

Отчёт по лабораторной работе №7

Дисциплина: Архитектура компьютера

Канева Екатерина Павловна

Содержание

| | | |
|----------|---------------------------------------|-----------|
| 1 | Цель работы | 5 |
| 2 | Задание | 6 |
| 3 | Теоретическое введение | 7 |
| 4 | Выполнение лабораторной работы | 10 |
| 4.1 | Лабораторная работа | 10 |
| 4.2 | Самостоятельная работа | 18 |
| 5 | Выводы | 22 |

Список иллюстраций

| | | |
|------|---|----|
| 4.1 | Создание каталога и файла. | 10 |
| 4.2 | Ввод текста программы. | 10 |
| 4.3 | Запуск программы. | 11 |
| 4.4 | Изменение текста программы. | 11 |
| 4.5 | Запуск программы. | 12 |
| 4.6 | Создание файла. | 12 |
| 4.7 | Ввод текста программы. | 12 |
| 4.8 | Запуск программы. | 13 |
| 4.9 | Изменение текста программы. | 13 |
| 4.10 | Запуск программы. | 13 |
| 4.11 | Изменение текста программы. | 14 |
| 4.12 | Запуск программы. | 14 |
| 4.13 | Создание файла. | 14 |
| 4.14 | Ввод текста программы. | 15 |
| 4.15 | Запуск программы. | 15 |
| 4.16 | Создание файла. | 15 |
| 4.17 | Ввод текста программы. | 16 |
| 4.18 | Запуск программы. | 16 |
| 4.19 | Создание файла. | 18 |
| 4.20 | Ввод текста программы. | 20 |
| 4.21 | Запуск программы со значением 8. | 20 |
| 4.22 | Запуск программы со значением 64. | 21 |

Список таблиц

1 Цель работы

Целью лабораторной работы является освоение арифметических инструкций языка ассемблера NASM.

2 Задание

1. 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 , выводить результат вычислений. Вид функции $f(x)$ выбрать из таблицы вариантов заданий в соответствии с номером полученным при выполнении лабораторной работы.
2. Создать исполняемый файл и проверить его работу для значений x_1 и x_2 .

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.

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 <операнд_1>, <операнд_2>
```

Команда целочисленного вычитания `sub` (от англ. *subtraction* – вычитание) работает аналогично команде `add` и выглядит следующим образом:

sub <операнд_1>, <операнд_2>

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 <операнд>

dec <операнд>

Еще одна команда, которую можно отнести к арифметическим командам, это команда изменения знака neg:

neg <операнд>

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 <операнд>

Для знакового умножения используется команда imul:

imul <операнд>

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

div <делитель> ; Беззнаковое деление

idiv <делитель> ; Знаковое деление

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,<int>`).
- `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,<int>`).

4 Выполнение лабораторной работы

4.1 Лабораторная работа

Создадим каталог для программ лабораторной работы №7, перейдём в него и создадим файл `lab7-1.asm` (рис. 4.1):

```
mkdir work/study/2022-2023/"Архитектура компьютера"/arh-pc/lab07
cd work/study/2022-2023/"Архитектура компьютера"/arh-pc/lab07
touch lab7-1.asm
```

```
[epkaneva@fedora ~]$ mkdir work/study/2022-2023/Архитектура\ компьютера/arh-pc/lab07
[epkaneva@fedora ~]$ cd work/study/2022-2023/Архитектура\ компьютера/arh-pc/lab07
[epkaneva@fedora lab07]$ touch lab7-1.asm
```

Рис. 4.1: Создание каталога и файла.

В файл `lab7-1.asm` введём текст программы, указанный в лабораторной работе (рис. 4.2):



The screenshot shows a text editor window titled "lab7-1.asm" with the file path "~/work/study/2022-2023/Архитектура компьютера/arh-pc/lab07". The code inside is as follows:

```
%include 'in_out.asm'

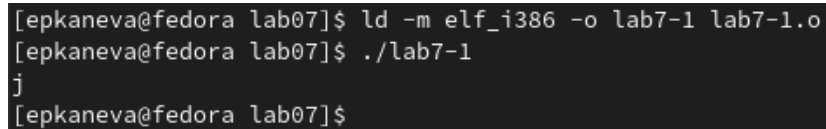
SECTION .bss
buf1:  RESB 80
SECTION .text
GLOBAL _start
_start:

mov eax, '6'
mov ebx, '4'
add eax, ebx
mov [buf1], eax
mov eax, buf1
call sprintf
call quit|
```

Рис. 4.2: Ввод текста программы.

Создадим исполняемый файл и запустим его (рис. 4.3):

```
nasm -f elf lab7-1.asm
ld -m elf_i386 -o lab7-1 lab7-1.o
./lab7-1
```



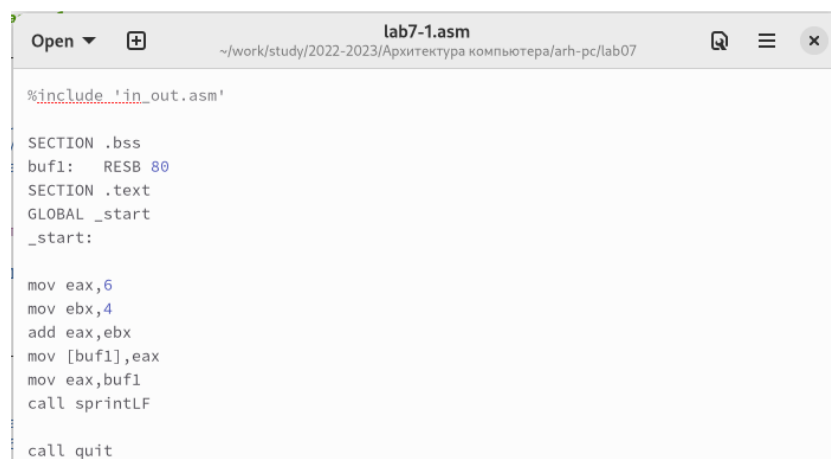
```
[epkaneva@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[epkaneva@fedora lab07]$ ./lab7-1
j
[epkaneva@fedora lab07]$
```

Рис. 4.3: Запуск программы.

Видим, что программа вывела символ j.

Далее изменим текст программы и вместо символов запишем в регистры числа.

Исправим текст программы (рис. 4.4):



```
lab7-1.asm
~/work/study/2022-2023/Архитектура компьютера/arh-pc/lab07

#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. 4.4: Изменение текста программы.

Создадим исполняемый файл и запустим его (рис. 4.5):

```
nasm -f elf lab7-1.asm
ld -m elf_i386 -o lab7-1 lab7-1.o
./lab7-1
```

```
[epkaneva@fedora lab07]$ nasm -f elf lab7-1.asm
[epkaneva@fedora lab07]$ ld -m elf_i386 -o lab7-1 lab7-1.o
[epkaneva@fedora lab07]$ ./lab7-1

[epkaneva@fedora lab07]$
```

Рис. 4.5: Запуск программы.

Программа выводит пустой символ. Это соответствует десятому символу в таблице ASCII.

Создадим файл `lab7-2.asm` (рис. 4.6) и введём текст программы (рис. 4.7):

```
touch lab7-2.asm
```

```
[epkaneva@fedora lab07]$ touch lab7-2.asm
```

Рис. 4.6: Создание файла.



```
lab7-2.asm
~/work/study/2022-2023/Архитектура компьютера/arh-pc/lab07

#include 'in_out.asm'

SECTION .text
GLOBAL _start
_start:

mov eax, '6'
mov ebx, '4'
add eax, ebx
call iprintLF
call quit
```

Рис. 4.7: Ввод текста программы.

Создадим исполняемый файл и запустим его (рис. 4.8):

```
nasm -f elf lab7-2.asm
ld -m elf_i386 -o lab7-2 lab7-2.o
./lab7-2
```

```
[epkaneva@fedora lab07]$ nasm -f elf lab7-2.asm
[epkaneva@fedora lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[epkaneva@fedora lab07]$ ./lab7-2
106
[epkaneva@fedora lab07]$
```

Рис. 4.8: Запуск программы.

На экран вывелось число 106.

Теперь изменим символы на числа, как ранее (рис. 4.9) и запустим программу (рис. 4.10):

```
nasm -f elf lab7-2.asm
ld -m elf_i386 -o lab7-2 lab7-2.o
./lab7-2
```



Рис. 4.9: Изменение текста программы.

```
[epkaneva@fedora lab07]$ nasm -f elf lab7-2.asm
[epkaneva@fedora lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[epkaneva@fedora lab07]$ ./lab7-2
10
[epkaneva@fedora lab07]$
```

Рис. 4.10: Запуск программы.

Заменим функцию iprintLF на iprint (рис. 4.11) и запустим программу (рис. 4.12):

```
nasm -f elf lab7-2.asm
ld -m elf_i386 -o lab7-2 lab7-2.o
./lab7-2
```



Рис. 4.11: Изменение текста программы.

```
[epkaneva@fedora lab07]$ nasm -f elf lab7-2.asm
[epkaneva@fedora lab07]$ ld -m elf_i386 -o lab7-2 lab7-2.o
[epkaneva@fedora lab07]$ ./lab7-2
10[epkaneva@fedora lab07]$
```

Рис. 4.12: Запуск программы.

Программа вывела число 10 и в той же строке приглашение для следующей команды. Вывод функций отличается тем, что функция `iprintLF` выполняет переход на новую строку после вывода, а `iprint` нет.

Создадим файл `lab7-3.asm` (рис. 4.13), в него введём текст программы из лабораторной работы (рис. 4.14):

```
touch lab7-3.asm
```

```
[epkaneva@fedora lab07]$ touch lab7-3.asm
```

Рис. 4.13: Создание файла.

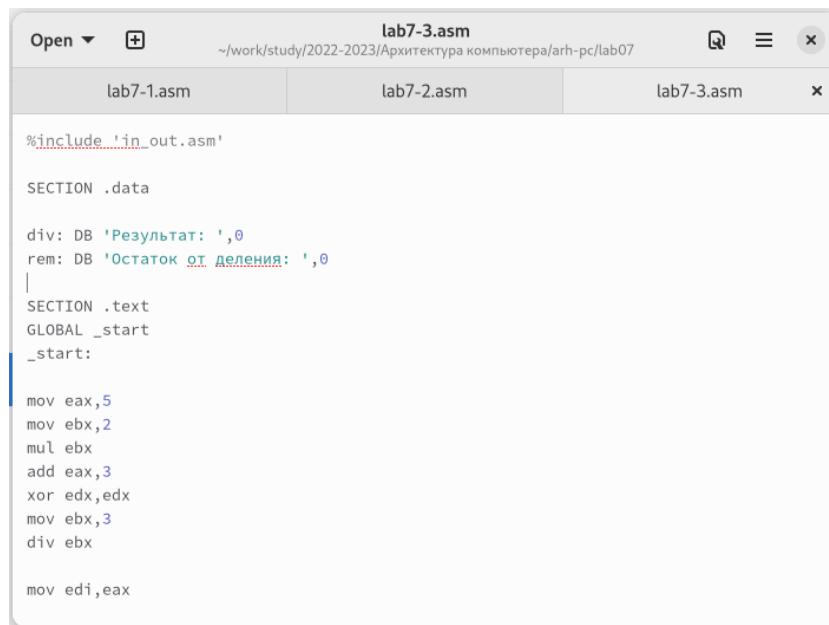


Рис. 4.14: Ввод текста программы.

Создадим исполняемый файл и запустим его (рис. 4.15):

```
nasm -f elf lab7-3.asm
ld -m elf_i386 -o lab7-3 lab7-3.o
./lab7-3
```

```
[epkaneva@fedora lab07]$ nasm -f elf lab7-3.asm
[epkaneva@fedora lab07]$ ld -m elf_i386 -o lab7-3 lab7-3.o
[epkaneva@fedora lab07]$ ./lab7-3
Результат: 4
Остаток от деления: 1
[epkaneva@fedora lab07]$
```

Рис. 4.15: Запуск программы.

Программа работает корректно.

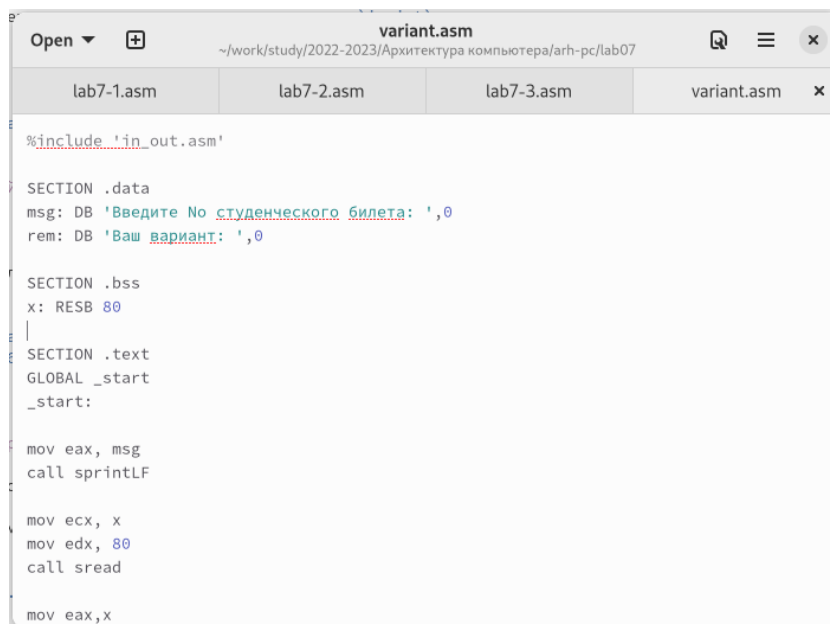
Далее создадим файл `variant.asm` (рис. 4.16):

```
touch variant.asm
```

```
[epkaneva@fedora lab07]$ touch variant.asm
```

Рис. 4.16: Создание файла.

Введём текст программы вычисления номера варианта по номеру студенческого билета (рис. 4.17):



```
variant.asm
~/work/study/2022-2023/Архитектура компьютера/argh-pc/lab07

lab7-1.asm lab7-2.asm lab7-3.asm variant.asm x

%include 'in_out.asm'

SECTION .data
msg: DB 'Введите No студенческого билета: ',0
rem: DB 'Ваш вариант: ',0

SECTION .bss
x: RESB 80

SECTION .text
GLOBAL _start
_start:

mov eax, msg
call sprintLF

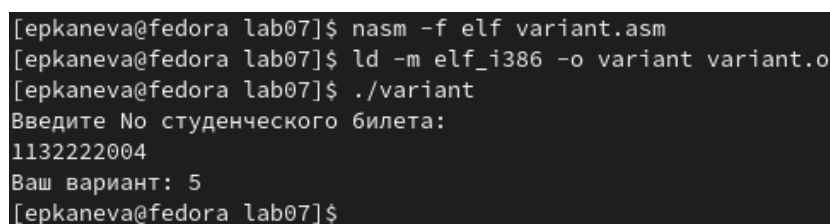
mov ecx, x
mov edx, 80
call sread

mov eax, x
```

Рис. 4.17: Ввод текста программы.

Создадим исполняемый файл и запустим его (рис. 4.18):

```
nasm -f elf variant.asm
ld -m elf_i386 -o variant variant.o
./variant
```



```
[epkaneva@fedora lab07]$ nasm -f elf variant.asm
[epkaneva@fedora lab07]$ ld -m elf_i386 -o variant variant.o
[epkaneva@fedora lab07]$ ./variant
Введите No студенческого билета:
1132222004
Ваш вариант: 5
[epkaneva@fedora lab07]$
```

Рис. 4.18: Запуск программы.

Программа вывела номер 5. Программа вычисляет вариант, прибавляя к остатку от деления номера варианта на 20 1 – результат действительно должен получиться таким.

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

Строки, отвечающие за вывод этого сообщения на экран:

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

2. Для чего используются следующие инструкции?

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

Эти строки отвечают за чтение вводимого номера студенческого билета.

3. Для чего используется инструкция “call atoi”?

Инструкция `call atoi` вызывает функцию `atoi`, которая преобразует ASCII-код символа в целое число и записывает результат в регистр `eax`.

4. Какие строки отвечают за вычисление варианта?

Строки, отвечающие за вычисление варианта:

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

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

В регистр `edx`.

6. Для чего используется инструкция “inc edx”?

Для увеличения значения, полученного при взятии остатка, на 1.

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

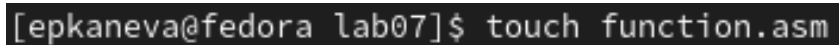
Строки, отвечающие за вывод результата со словами “Ваш вариант:”:

```
mov eax,rem  
call sprint  
mov eax,edx  
call iprintLF
```

4.2 Самостоятельная работа

Для написания программы создадим файл `function.asm` (рис. 4.19):

```
touch function.asm
```



```
[epkaneva@fedora lab07]$ touch function.asm
```

Рис. 4.19: Создание файла.

Введём текст программы (рис. 4.20):

```
%include 'in_out.asm'  
  
SECTION .data  
stm: DB 'y = (9x - 8) / 8', 0  
msg: DB 'Введите значение x: ', 0  
res: DB 'Результат вычислений: ', 0  
  
SECTION .bss
```

```
x: RESB 80
```

```
SECTION .text
```

```
GLOBAL _start
```

```
_start:
```

```
mov eax, stm
```

```
call sprintLF
```

```
mov eax, msg
```

```
call sprintLF
```

```
mov ecx, x
```

```
mov edx, 80
```

```
call sread
```

```
mov eax, x
```

```
call atoi
```

```
mov ebx, 12
```

```
mul ebx
```

```
add eax, 3
```

```
xor edx, edx
```

```
mov ebx, 5
```

```
idiv ebx
```

```
mov edi, eax
```

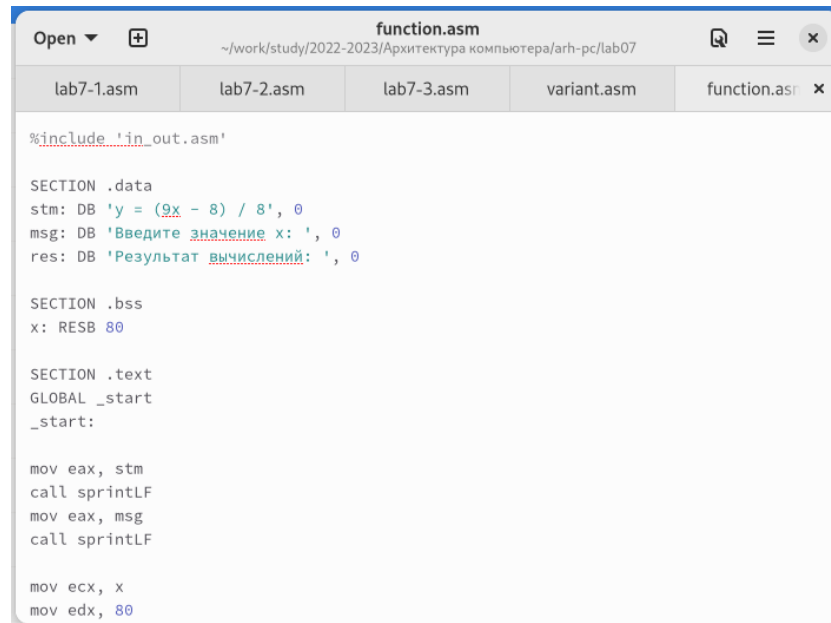
```
mov eax, res
```

```
call sprint
```

```
mov eax, edi
```

```
call iprintLF
```

```
call quit
```



```
function.asm
~/work/study/2022-2023/Архитектура компьютера/arh-pc/lab07

lab7-1.asm lab7-2.asm lab7-3.asm variant.asm function.asm x

%include 'in_out.asm'

SECTION .data
stm: DB 'y = (9x - 8) / 8', 0
msg: DB 'Введите значение x: ', 0
res: DB 'Результат вычислений: ', 0

SECTION .bss
x: RESB 80

SECTION .text
GLOBAL _start
_start:

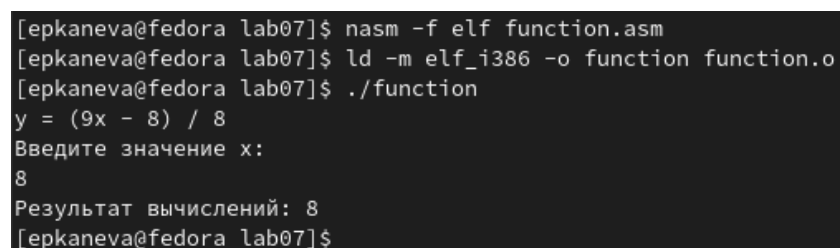
mov eax, stm
call sprintLF
mov eax, msg
call sprintLF

mov ecx, x
mov edx, 80
```

Рис. 4.20: Ввод текста программы.

Создадим исполняемый файл и запустим его (рис. 4.21 и 4.22):

```
nasm -f elf function.asm
ld -m elf_i386 -o function function.o
./function
```



```
[epkaneva@fedora lab07]$ nasm -f elf function.asm
[epkaneva@fedora lab07]$ ld -m elf_i386 -o function function.o
[epkaneva@fedora lab07]$ ./function
y = (9x - 8) / 8
Введите значение x:
8
Результат вычислений: 8
[epkaneva@fedora lab07]$
```

Рис. 4.21: Запуск программы со значением 8.

```
[epkaneva@fedora lab07]$ ./function
y = (9x - 8) / 8
Введите значение x:
64
Результат вычислений: 71
[epkaneva@fedora lab07]$
```

Рис. 4.22: Запуск программы со значением 64.

Проверим вычисления, выполненные программой:

$$(9 * 8 - 8) / 8 = 8 \quad (9 * 64 - 8) / 8 = 71$$

Программа работает корректно.

5 Выводы

Освоили арифметические инструкции языка ассемблера NASM.