

Отчет по лабораторной работе №5

Создание и процесс обработки программ на языке ассемблера NASM

Виктор Максимович Кадров

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	7
5	Задания для самостоятельной работы	12
6	Ответы на вопросы для самопроверки	14
7	Выводы	17

Список иллюстраций

4.1	Создание папки и файла	7
4.2	Проверка с помощью ls	8
4.3	Код программы	8
4.4	Вызов транслятора nasm	9
4.5	Nasm с различными ключами	9
4.6	Компоновка	10
4.7	Задание другого имени	10
4.8	Выполнение программы	11
5.1	Измененный код	12
5.2	Выполнение программы	13
5.3	Копирование файлов	13

1 Цель работы

Освоение процедуры компиляции и сборки программ, написанных на ассемблере NASM.

2 Задание

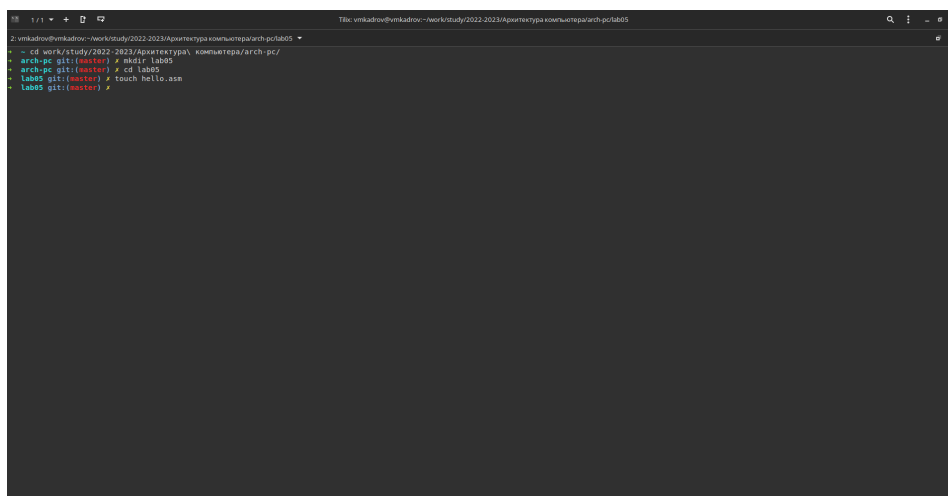
1. Написание программ Hello World!, ее трансляция, компоновка и выполнение.
2. Выполнение заданий для самостоятельной работы.
3. Ответы на вопросы для самопроверки.

3 Теоретическое введение

Язык ассемблера (assembly language, сокращённо asm) — машинноориентированный язык низкого уровня. Можно считать, что он больше любых других языков приближен к архитектуре ЭВМ и её аппаратным возможностям, что позволяет получить к ним более полный доступ, нежели в языках высокого уровня, таких как C/C++, Perl, Python и пр. Программы, написанные на языке ассемблера, не уступают в качестве и скорости программам, написанным на машинном языке, так как транслятор просто переводит мнемонические обозначения команд в последовательности бит (нулей и единиц). Используемые мнемоники обычно одинаковы для всех процессоров одной архитектуры или семейства архитектур (среди широко известных — мнемоники процессоров и контроллеров x86, ARM, SPARC, PowerPC, M68k). Таким образом для каждой архитектуры существует свой ассемблер и, соответственно, свой язык ассемблера.

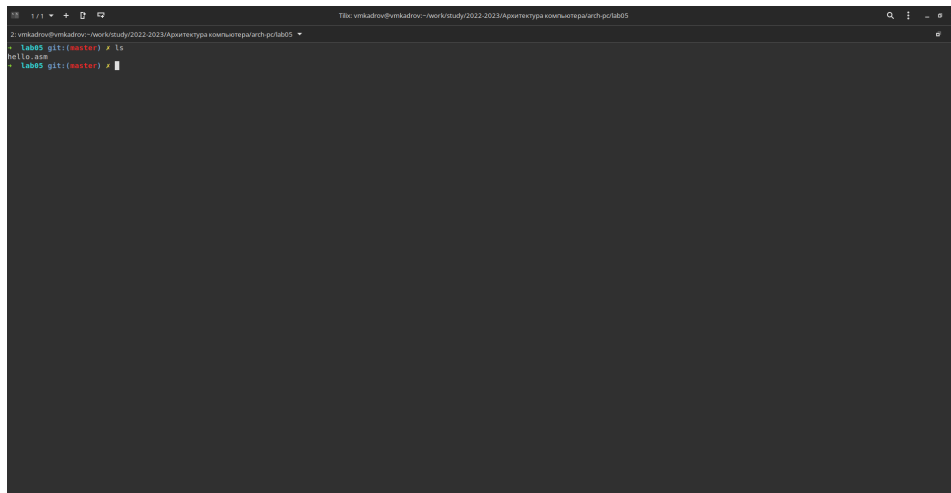
4 Выполнение лабораторной работы

Создаем папку lab05 и файл hello.asm в ней (рис. 4.1). После этого проверяем, что файл действительно был создан (рис. 4.2). Пишем код программы в созданном файле (рис. 4.3). Транслируем его при помощи NASM в объектный файл (hello.o) и проверяем наличие (рис. 4.4). Выплняем команду с дополнительными ключами (рис. 4.5). Компонуем и получаем на выходе исполняемый файл (рис. 4.6), выполняем команду с заданием другого имени (obj.o → main) (рис. 4.7). Исполняем его (рис. 4.8).



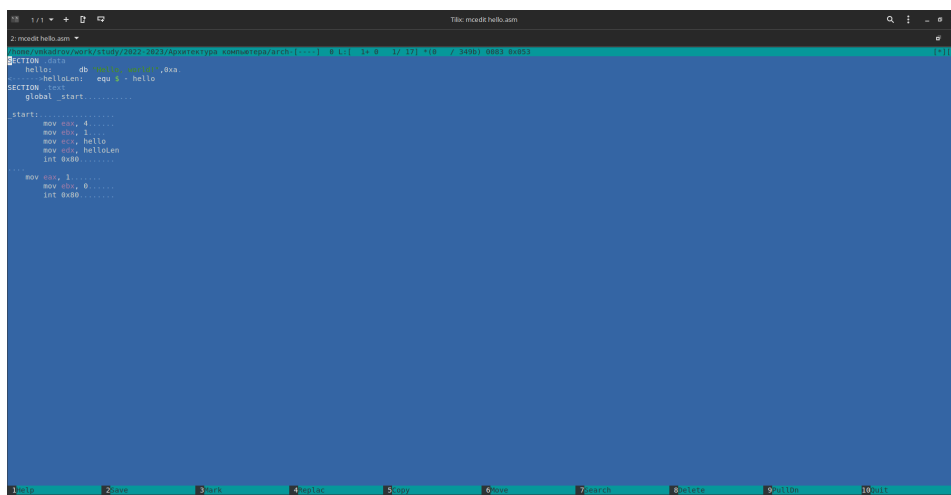
```
2 vmladov@vmladov:~/work/study/2022-2023/Архитектура компьютера/arch-pc/lab05
└─ cd work/study/2022-2023/Архитектура компьютера/arch-pc/
└─ arch-pc git:(master) # mkdir lab05
└─ arch-pc git:(master) # cd lab05
└─ lab05 git:(master) # touch hello.asm
└─ lab05 git:(master) #
```

Рис. 4.1: Создание папки и файла



```
2 vmkadrov@vmkadrov:~/work/study/2022-2023/Архитектура компьютерарх-pc/lab05  
- Lab05 git:(master) * ls  
hello.asm  
- Lab05 git:(master) *
```

Рис. 4.2: Проверка с помощью ls



```
2 moadt@helo.asm  
moadt@helo.asm:~/work/study/2022-2023/Архитектура компьютерарх-pc/lab05  
SECTION .data  
hello: db "hello, world!\n",0xa  
hellolen: equ $ - hello  
SECTION .text  
global _start  
_start:  
mov eax, 4  
mov ebx, 1  
mov ecx, hello  
mov edx, hellolen  
int 0x80  
mov ebx, 1  
mov ecx, 0  
int 0x80
```

Рис. 4.3: Код программы


```
2 vmkadrov@vmkadrov:~/work/study/2022-2023/Архитектура компьютерарх-pc/lab05
* Lab05 git:(master) x nasm -f elf hello.asm
* Lab05 git:(master) x ls
hello.asm  hello.o
* Lab05 git:(master) x
```

Рис. 4.4: Вызов транслятора nasm

```
2 vmkadrov@vmkadrov:~/work/study/2022-2023/Архитектура компьютерарх-pc/lab05
* Lab05 git:(master) x nasm -o obj.o -f elf -g -l list.lst hello.asm
* Lab05 git:(master) x ls
hello.asm  hello.o  list.lst  obj.o
* Lab05 git:(master) x
```

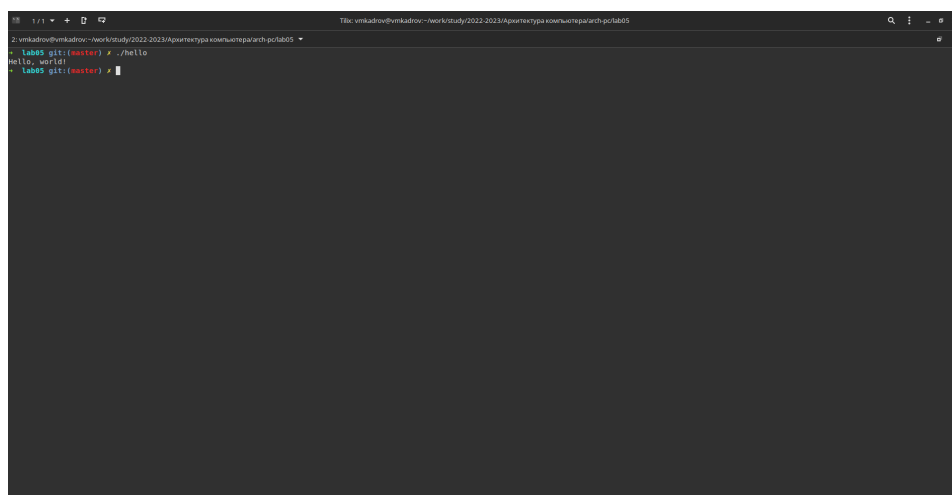
Рис. 4.5: Nasm с различными ключами

```
2 vmkadrov@vmkadrov:~/work/study/2022-2023/Архитектура компьютерарх-pc/lab05
+ lab05 git:(master) x ld -m elf_i386 hello.o -o hello
+ lab05 git:(master) x ls
hello hello.asm hello.o list.lst obj.o
+ lab05 git:(master) x
```

Рис. 4.6: Компоновка

```
2 vmkadrov@vmkadrov:~/work/study/2022-2023/Архитектура компьютерарх-pc/lab05
+ lab05 git:(master) x ld -m elf_i386 obj.o -o main
+ lab05 git:(master) x ls
hello hello.asm hello.o list.lst main obj.o
+ lab05 git:(master) x
```

Рис. 4.7: Задание другого имени



A terminal window with a dark background. The title bar at the top reads "Tlac vmkadrov@vmkadrov:~/work/study/2022-2023/Apuntektyga kompyutera/arch-pc/lab05". The terminal content shows a prompt "2 vmkadrov@vmkadrov:~/work/study/2022-2023/Apuntektyga kompyutera/arch-pc/lab05" followed by a command "Lab05 git:(master) x ./hello" and its output "hello, world!". The prompt "Lab05 git:(master) x" is repeated on the next line.

```
Tlac vmkadrov@vmkadrov:~/work/study/2022-2023/Apuntektyga kompyutera/arch-pc/lab05
2 vmkadrov@vmkadrov:~/work/study/2022-2023/Apuntektyga kompyutera/arch-pc/lab05
Lab05 git:(master) x ./hello
hello, world!
Lab05 git:(master) x
```

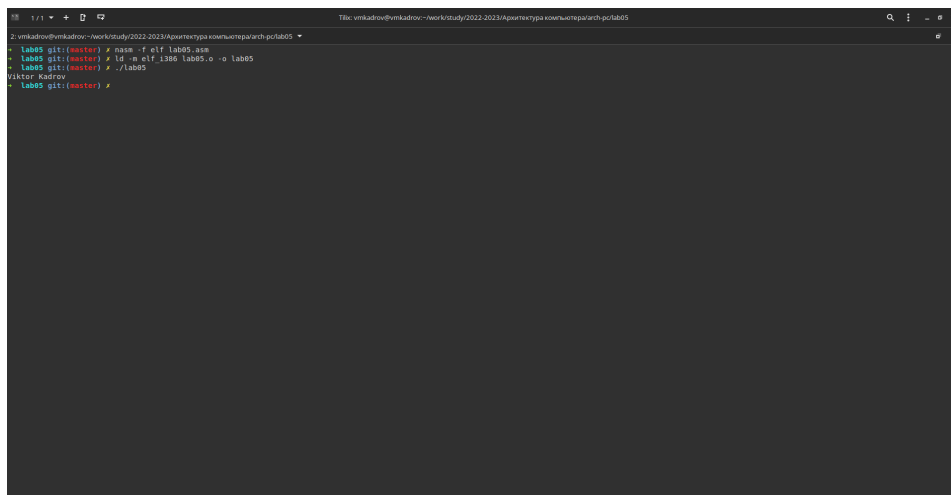
Рис. 4.8: Выполнение программы

5 Задания для самостоятельной работы

Изменяем код программы в копии файла (рис. 5.1). Транслируем, компонуем и проверяем, что программа выполняется корректно (рис. 5.2). Копируем файлы в папку лаб. работы (рис. 5.3). После заполнения отчета загружаем файлы на github.

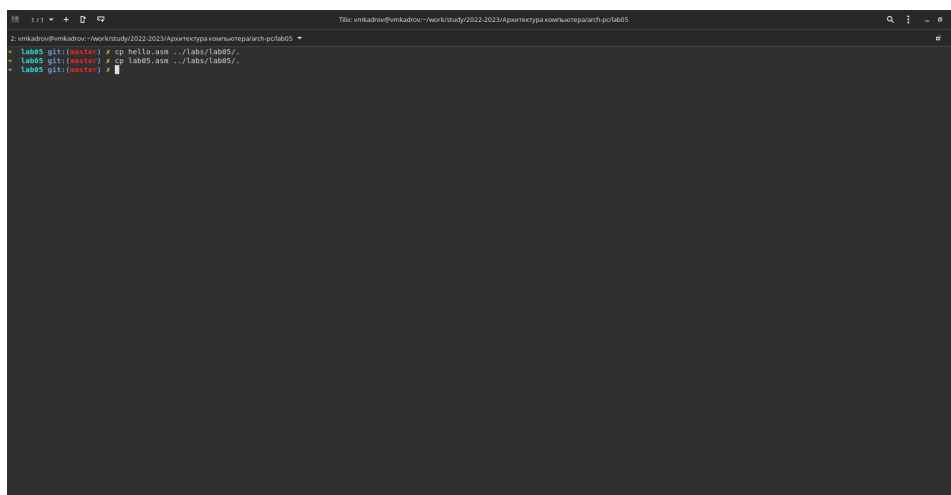
```
Tik-mcride lab05.asm  
#  
2: mcride lab05.asm  
  
[...]  
SECTION .text  
hello: db "hello\n",0x0a  
        helloLen equ $ - hello  
SECTION .start  
global start  
  
start:  
    mov rax, 4  
    mov rdi, 1  
    mov rsi, hello  
    mov rdx, helloLen  
    int 0x80  
  
    mov rax, 1  
    mov rdi, 0  
    int 0x80
```

Рис. 5.1: Измененный код



```
2 vmkadrov@vmkadrov:~/work/study/2022-2023/Apnet/textyga_kompyuternaya_arch/lab05
* lab05 git:(master) x nasm -f elf lab05.asm
* lab05 git:(master) x ld -o elf_lab05 lab05.o -o lab05
* lab05 git:(master) x ./lab05
Viktor Andrey
* lab05 git:(master) x
```

Рис. 5.2: Выполнение программы



```
2 vmkadrov@vmkadrov:~/work/study/2022-2023/Apnet/textyga_kompyuternaya_arch/lab05
* lab05 git:(master) x cp hello.asm ../labs/lab05/
* lab05 git:(master) x cp lab05.asm ../labs/lab05/
* lab05 git:(master) x
```

Рис. 5.3: Копирование файлов

6 Ответы на вопросы для самопроверки

1. Язык уровня ассемблера: язык низкого уровня, который позволяет пользователям писать программы, используя буквенно-цифровые мнемонические коды вместо числового кода для набора инструкций.

Язык высокого уровня: машинно-независимый язык. Он позволяет пользователю писать программы на языке, который напоминает английские слова и знакомые математические символы. Примерами языков высокого уровня являются python, c# и др.

Язык ассемблера (assembly language, сокращенно asm) — машинно-ориентированный язык низкого уровня. Можно считать, что он больше любых других языков приближен к архитектуре ЭВМ и её аппаратным возможностям, что позволяет получить к ним более полный доступ, нежели в языках высокого уровня, таких как C/C++, Perl, Python и пр. Заметим, что получить полный доступ к ресурсам компьютера в современных архитектурах нельзя, самым низким уровнем работы прикладной программы является обращение напрямую к ядру операционной системы. Именно на этом уровне и работают программы, написанные на ассемблере. Но в отличие от языков высокого уровня ассемблерная программа содержит только тот код, который ввел программист. Таким образом язык ассемблера — это язык, с помощью которого понятным для человека образом пишутся команды для процессора.

2. Программа на языке ассемблера также может содержать директивы — инструкции, не переводящиеся непосредственно в машинные команды, а

управляющие работой транслятора. Например, директивы используются для определения данных (констант и переменных) и обычно пишутся большими буквами.

3. Правила оформления ассемблерных программ При наборе программ на языке ассемблера придерживайтесь следующих правил:

- директивы набираются большими буквами, инструкции – малыми;
- текст пишется широко;
- не стоит выходить за край экрана – его неудобно будет редактировать и печатать;
- для отступов используется табуляция (клавиша TAB);
- блоки комментариев задаются с одинаковым отступом. Оптимальной считается такая строка: `moveax,ebx<(1-3)TAB>;` текст комментария Количество табуляций перед комментарием определяется длиной аргументов команды и может быть от 1 до 3.

4. Трансляция и компоновка.

5. Трансляция исходного текста программы состоит в преобразовании предложений исходного языка в коды машинных команд и выполняется с помощью транслятора с языка ассемблера (т. е. с помощью программы ассемблера).

6. Чтобы получить исполняемую программу, объектный файл необходимо передать на обработку компоновщику.

7. После того, как текст программы набран и записан на жесткий диск, необходимо произвести трансляцию программы. В процессе трансляции создается результирующий (объектный) файл, который представляет собой ту же программу, но в машинных кодах, предназначенную для записи в программную память микроконтроллера. Результирующий файл имеет расширение hex.

Кроме hex-файла транслятор создает еще несколько вспомогательных файлов. И главное, файл с расширением eep. Этот файл имеет точно такую же внутреннюю структуру, как файл hex. А содержит он информацию, предназначенную для записи в EEPROM. Такая информация появляется в том случае, когда в тексте программы переменным, размещенным в сегменте eeprom, присвоены начальные значения. В обоих случаях (hex и eep форматы) применяется так называемый HEX-формат, который практически является стандартом для записи результатов транслирования различных программ. Он поддерживается практически всеми трансляторами с любого языка программирования.

8. NASM поддерживает множество форматов выходных файлов, среди них:

- bin
- obj
- win32 и win64
- aout
- aoutb
- coff
- elf32 и elf64 Компоновщик ld не предполагает по умолчанию расширений для файлов. Но принято использовать следующие расширения:
 - o для объектных файлов;
 - без расширения для исполняемых файлов;
 - map для файлов схемы программы;
 - lib для библиотек.

7 Выводы

В ходе выполнения лабораторной работы были освоены процедуры компиляции и сборки программ, написанных на ассемблере NASM.