

Отчёт по лабораторной работе 4

**Создание и процесс обработки программ на языке ассемблера
NASM**

Бозорова Ануша

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
2.1	Программа Hello world!	6
2.2	Транслятор NASM	7
2.3	Расширенный синтаксис командной строки NASM	8
2.4	Компоновщик LD	8
2.5	Запуск исполняемого файла	9
2.6	Задание для самостоятельной работы	9
3	Выводы	12
4	Вопросы для самопроверки	13

Список иллюстраций

2.1	Создан каталог для работы и файл для программы	6
2.2	Программа в файле hello.asm	7
2.3	Трансляция программы	8
2.4	Трансляция программы с дополнительными опциями	8
2.5	Компоновка программы	9
2.6	Компоновка программы	9
2.7	Запуск программы	9
2.8	Скопировала файл	10
2.9	Программа в файле lab4.asm	10
2.10	Проверка программы lab4.asm	11

Список таблиц

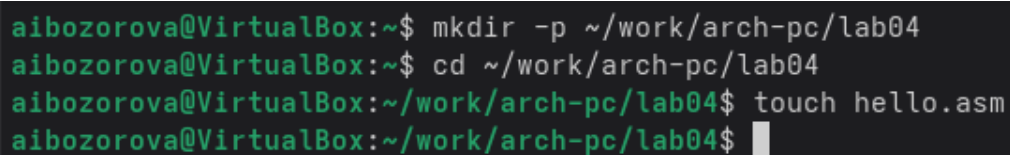
1 Цель работы

Целью работы является освоение процедуры компиляции и сборки программ, написанных на ассемблере NASM.

2 Выполнение лабораторной работы

2.1 Программа Hello world!

Создала каталог lab04 командой mkdir, перешла в него с помощью команды cd и создала файл hello.asm, в который напишу программу. Убеждаюсь с помощью команды ls, что создал файл.

A screenshot of a terminal window with a dark background and green text. It shows four lines of commands and their execution in a VirtualBox environment. The user 'aibozorova' is at the prompt. The commands are: 'mkdir -p ~/work/arch-pc/lab04', 'cd ~/work/arch-pc/lab04', 'touch hello.asm', and a final prompt line.

```
aibozorova@VirtualBox:~$ mkdir -p ~/work/arch-pc/lab04
aibozorova@VirtualBox:~$ cd ~/work/arch-pc/lab04
aibozorova@VirtualBox:~/work/arch-pc/lab04$ touch hello.asm
aibozorova@VirtualBox:~/work/arch-pc/lab04$
```

Рисунок 2.1: Создан каталог для работы и файл для программы

Написала программу по заданию на языке ассемблера.

```
SECTION .data
    hello:      db "Hello, world!",0xa
               helloLen: equ $ - hello

SECTION .text
    global _start

_start:
    mov eax, 4
    mov ebx, 1
    mov ecx, hello
    mov edx, helloLen
    int 0x80

    mov eax, 1
    mov ebx, 0
    int 0x80
```

Рисунок 2.2: Программа в файле hello.asm

2.2 Транслятор NASM

NASM превращает текст программы в объектный код. Если текст программы набран без ошибок, то транслятор преобразует текст программы из файла hello.asm в объектный код, который запишется в файл hello.o.

Транслировала файл командой `nasm`. Получился объектный файл hello.o.

```

aibozorova@VirtualBox:~/work/arch-pc/lab04$
aibozorova@VirtualBox:~/work/arch-pc/lab04$ nasm -f elf hello.asm
aibozorova@VirtualBox:~/work/arch-pc/lab04$ ls
hello.asm  hello.o
aibozorova@VirtualBox:~/work/arch-pc/lab04$

```

Рисунок 2.3: Трансляция программы

2.3 Расширенный синтаксис командной строки NASM

Полный вариант командной строки `nasm` выглядит следующим образом:

```

nasm [-@ косвенный_файл_настроек] [-o объектный_файл] [-f формат_объектного_файла] [-

```

Транслировала файл командой `nasm` с дополнительными опциями. С опцией `-l` Получил файл листинга `list.lst`, с опцией `-f` объектный файл `obj.o`, с опцией `-g` в программу добавилась отладочная информация.

```

aibozorova@VirtualBox:~/work/arch-pc/lab04$
aibozorova@VirtualBox:~/work/arch-pc/lab04$ nasm -o obj.o -f elf -g -l list.lst hello.asm
aibozorova@VirtualBox:~/work/arch-pc/lab04$ ls
hello.asm  hello.o  list.lst  obj.o
aibozorova@VirtualBox:~/work/arch-pc/lab04$

```

Рисунок 2.4: Трансляция программы с дополнительными опциями

2.4 Компоновщик LD

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

Выполнила команду `ld` и получила исполняемый файл `hello` из объектного файла `hello.o`.


```

aibozorova@VirtualBox:~/work/arch-pc/lab04$
aibozorova@VirtualBox:~/work/arch-pc/lab04$ ld -m elf_i386 hello.o -o hello
aibozorova@VirtualBox:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  list.lst  obj.o
aibozorova@VirtualBox:~/work/arch-pc/lab04$

```

Рисунок 2.5: Компоновка программы

Еще раз выполнила команду `ld` для объектного файла `obj.o` и получила исполняемый файл `main`.

```

aibozorova@VirtualBox:~/work/arch-pc/lab04$
aibozorova@VirtualBox:~/work/arch-pc/lab04$ ld -m elf_i386 obj.o -o main
aibozorova@VirtualBox:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  list.lst  main  obj.o
aibozorova@VirtualBox:~/work/arch-pc/lab04$

```

Рисунок 2.6: Компоновка программы

2.5 Запуск исполняемого файла

Запустила исполняемые файлы.

```

aibozorova@VirtualBox:~/work/arch-pc/lab04$
aibozorova@VirtualBox:~/work/arch-pc/lab04$ ./hello
Hello, world!
aibozorova@VirtualBox:~/work/arch-pc/lab04$ ./main
Hello, world!
aibozorova@VirtualBox:~/work/arch-pc/lab04$

```

Рисунок 2.7: Запуск программы

2.6 Задание для самостоятельной работы

Скопировала файл `hello.asm` в файл `lab4.asm`.

```
aibozorova@VirtualBox:~/work/arch-pc/lab04$  
aibozorova@VirtualBox:~/work/arch-pc/lab04$ cp hello.asm lab4.asm  
aibozorova@VirtualBox:~/work/arch-pc/lab04$ ls  
hello  hello.asm  hello.o  lab4.asm  list.lst  main  obj.o  
aibozorova@VirtualBox:~/work/arch-pc/lab04$
```

Рисунок 2.8: Скопировала файл

Изменила сообщение Hello world на свое имя.

```
SECTION .data  
    hello:      db "Бозорова Ануша",0ха  
                helloLen: equ $ - hello  
  
SECTION .text  
    global _start  
  
_start:  
    mov eax, 4  
    mov ebx, 1  
    mov ecx, hello  
    mov edx, helloLen  
    int 0x80  
  
    mov eax, 1  
    mov ebx, 0  
    int 0x80
```

Рисунок 2.9: Программа в файле lab4.asm

Запустила программу и проверила.

```
aibozorova@VirtualBox:~/work/arch-pc/lab04$ nasm -f elf lab4.asm
aibozorova@VirtualBox:~/work/arch-pc/lab04$ ld -m elf_i386 lab4.o -o lab4
aibozorova@VirtualBox:~/work/arch-pc/lab04$ ./lab4
Бозорова Ануша
aibozorova@VirtualBox:~/work/arch-pc/lab04$
```

Рисунок 2.10: Проверка программы lab4.asm

3 Выводы

Освоили процесс компиляции и сборки программ, написанных на ассемблере `nasm`.

4 Вопросы для самопроверки

1. Какие основные отличия ассемблерных программ от программ на языках высокого уровня?

Уровень абстракции: Ассемблерные программы работают напрямую с машинными инструкциями, что требует от программиста понимания архитектуры процессора, адресации памяти и регистров. Программы на языках высокого уровня (например, C++, Python) предоставляют более абстрактные конструкции (циклы, функции, классы), скрывающие детали работы процессора.

Число инструкций: Ассемблерные программы требуют написания большего количества инструкций для выполнения простых задач, в то время как в языках высокого уровня многие операции выполняются одной строчкой кода.

Портируемость: Ассемблерный код сильно зависит от архитектуры конкретного процессора (например, x86, ARM), в то время как программы на языках высокого уровня могут быть легче перенесены на другую платформу при наличии соответствующего компилятора.

Оптимизация: Ассемблерные программы могут быть сильно оптимизированы для работы с конкретным оборудованием, что делает их более эффективными по производительности. Языки высокого уровня полагаются на компиляторы, которые автоматически производят оптимизацию, но не всегда с таким уровнем эффективности.

2. В чём состоит отличие инструкции от директивы на языке ассемблера?

Инструкция — это команда, которая будет непосредственно преобразована в машинный код и выполнена процессором. Например, инструкции для работы с регистрами, памятью, или выполнения арифметических операций.

Директива — это команда для ассемблера, которая не превращается в машинный код, а управляет процессом компиляции или настройкой программы. Например, директивы указывают, как организовать память, где начинается программа, или как подключить внешние библиотеки (например, section, global).

3. Перечислите основные правила оформления программ на языке ассемблера.

Чёткая структура кода: Обычно программа состоит из секций (например, .text, .data), где каждая секция отвечает за разные типы данных (инструкции, данные, стек).

Комментарии: Программы на ассемблере обычно сложны для понимания, поэтому комментарии играют важную роль в объяснении назначения каждой строки кода.

Использование регистров и памяти: Необходимо чётко управлять регистрами и адресами памяти, чтобы избежать ошибок (например, выход за границы памяти или ошибки при работе с регистрами).

Именованые меток и переменных: Метки должны быть уникальными и легко читаемыми, чтобы можно было быстро понять структуру программы и её логику.

4. Каковы этапы получения исполняемого файла?

Написание исходного кода на языке программирования.

Трансляция (ассемблирование): Ассемблер преобразует исходный код в машинный код (объектный файл).

Компоновка (линковка): Компоновщик объединяет объектные файлы и подключаемые библиотеки в исполняемый файл.

Загрузка и выполнение: Исполняемый файл загружается в память и выполняется операционной системой.

5. Каково назначение этапа трансляции?

Трансляция — это процесс, при котором исходный код программы на языке ассемблера переводится в машинный код (объектный файл). На этом этапе проверяется синтаксис программы, распределяются регистры, память и создаются машинные инструкции.

6. Каково назначение этапа компоновки?

Компоновка (линковка) — это процесс объединения нескольких объектных файлов и библиотек в единый исполняемый файл. Компоновщик проверяет наличие всех внешних ссылок и функций, а также распределяет адресное пространство для программы.

7. Какие файлы могут создаваться при трансляции программы, какие из них создаются по умолчанию?

Объектные файлы (.o или .obj): Это промежуточные файлы, созданные ассемблером на основе исходного кода. Они содержат машинные инструкции и ссылки на внешние библиотеки.

Исполняемые файлы (.exe, .out): Финальные файлы, которые создаются после компоновки и могут быть выполнены операционной системой.

Отладочные файлы (.debug): Создаются по желанию для целей отладки программы.

8. Каковы форматы файлов для nasm и ld?

NASM (Netwide Assembler): Использует объектные форматы, такие как ELF (для Linux), COFF (для Windows), Mach-O (для macOS), а также плоский бинарный формат (.bin).

LD (GNU Linker): Поддерживает различные форматы объектных файлов и исполняемых файлов, такие как ELF и PE (для Windows).