

Отчет по лабораорной работе № 4

Дисциплина: архитектура компьютеров

Выслоух Алиса Александровна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	10
4.1	Создание программы Hello world	10
4.2	Работа с транслятором NASM	11
4.3	Работа с расширенным синтаксисом командной строки NASM . . .	12
4.4	Работа с компоновщиком LD	12
4.5	Запуск исполняемого файла	12
5	Задания для самостоятельной работы	13
6	Выводы	15

Список иллюстраций

4.1	Создание каталога	10
4.2	Перемещение между дерикториями.	10
4.3	Создание файла.	10
4.4	Открытие файла и вставка программы.	11
4.5	Компиляция файла.	11
4.6	Проверка	11
4.7	Компиляция	12
4.8	Передача на обработку	12
4.9	Запуск программы	12
5.1	Создание копии	13
5.2	Открытие файла и его изменение	13
5.3	Компиляция	14
5.4	Передача файла	14
5.5	Запуск программы	14
5.6	Удаление файлов и проверка	14
5.7	Загрузка файлов на гитхаб.	14

Список таблиц

1 Цель работы

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

2 Задание

1. Создание программы Hello world!
2. Работа с транслятором NASM
3. Работа с расширенным синтаксисом командной строки NASM
4. Работа с компоновщиком LD
5. Запуск исполняемого файла
6. Выполнение заданий для самостоятельной работы.

3 Теоретическое введение

Основными элементами любой ЭВМ являются центральный процессор, память и периферийные устройства. Эти устройства взаимодействуют через общую шину, к которой они подключены. Физическая шина состоит из множества проводников, соединяющих устройства. В современных компьютерах проводники реализованы как электропроводящие дорожки на материнской плате. Главной задачей процессора является обработка данных и координация работы всех узлов компьютера. Центральный процессор включает следующие компоненты:

- арифметико-логическое устройство (АЛУ) — выполняет арифметические и логические операции, необходимые для обработки информации в памяти;
- устройство управления (УУ) — отвечает за управление и контроль всех компьютерных устройств;
- регистры — сверхбыстрая оперативная память небольшого объема для временного хранения промежуточных результатов выполнения команд; регистры делятся на общего назначения и специальные. Знание о регистрах процессора необходимо для программирования на ассемблере. Большинство команд ассемблерных программ используют регистры как операнды. Все команды часто преобразуют данные, хранящиеся в регистрах, что включает пересылку данных между регистрами и памятью или выполнение арифметических и логических операций. Доступ к регистрам осуществляется по именам, а не по адресам, как к основной памяти.

Каждый регистр архитектуры x86 имеет название из 2 или 3 латинских букв. Примеры основных регистров общего назначения:

- RAX, RCX, RDX, RBX, RSI, RDI — 64-битные;
- EAX, ECX, EDX, EBX, ESI, EDI — 32-битные;
- AX, CX, DX, BX, SI, DI — 16-битные;
- AH, AL, CH, CL, DH, DL, BH, BL — 8-битные.

Другим важным компонентом ЭВМ является оперативное запоминающее устройство (ОЗУ). Оно — энергозависимое, быстродействующее запоминающее устройство, обеспечивающее взаимодействие с процессором для хранения текущих программ и данных. ОЗУ состоит из пронумерованных ячеек памяти, адреса которых соответствуют хранимым в них данным. Периферийные устройства ЭВМ включают:

- устройства внешней памяти для долговременного хранения больших объемов данных;
- устройства ввода-вывода для обеспечения взаимодействия ЦП с внешней средой.

Вычислительный процесс ЭВМ основан на принципе программного управления, что предполагает, что задачи решаются как последовательность действий, записанных в программе.

Коды команд представляют собой двоичные комбинации из 0 и 1. У каждого машинного кода есть операционная и адресная части. Операционная часть хранит код команды, а адресная — данные или адреса, необходимые для ее выполнения. При каждом выполнении команды процессор выполняет стандартный набор действий, что называется командным циклом. Этот цикл состоит из следующих этапов:

- формирование адреса следующей команды в памяти;
- считывание кода команды из памяти и ее дешифрация;
- выполнение команды;
- переход к следующей команде.

Язык ассемблера (assembly language, сокращенно asm) — это низкоуровневый, машинно-ориентированный язык. NASM — открытый проект ассемблера, версии которого доступны для различных ОС и который позволяет создавать объектные файлы для этих систем. NASM использует Intel-синтаксис и поддерживает инструкции x86-64.

4 Выполнение лабораторной работы

4.1 Создание программы Hello world

Создаю каталог для работы с программами на языке ассемблера NASM. (рис. 4.1).

```
aavihsloukh@dk1n22 ~ $ mkdir -p ~/work/arch-pc/lab04
```

Рис. 4.1: Создание каталога

С помощью утилиты `cd` перемещаюсь в каталог, в котором буду работать (рис. 4.2).

```
aavihsloukh@dk1n22 ~ $ cd ~/work/arch-pc/lab04
```

Рис. 4.2: Перемещение между дерикториями.

Создаю в текущем каталоге пустой текстовый файл `hello.asm` с помощью команды `touch` (рис. 4.3)

```
aavihsloukh@dk1n22 ~/work/arch-pc/lab04 $ touch hello.asm
```

Рис. 4.3: Создание файла.

Открываю созданный файл в текстовом редакторе `gedit` и аполняю файл, вставляя в него программу для вывода “Hello word!”(рис. 4.4).

```

1 ; hello.asm
2 SECTION .data ; Начало секции данных
3 hello: DB 'Hello world!',10 ; 'Hello world!' плюс
4 ; символ перевода строки
5 helloLen: EQU $-hello ; Длина строки hello
6 SECTION .text ; Начало секции кода
7 GLOBAL _start
8 _start: ; Точка входа в программу
9 mov eax,4 ; Системный вызов для записи (sys_write)
10 mov ebx,1 ; Описатель файла '1' - стандартный вывод
11 mov ecx,hello ; Адрес строки hello в ecx
12 mov edx,helloLen ; Размер строки hello
13 int 80h ; Вызов ядра
14 mov eax,1 ; Системный вызов для выхода (sys_exit)
15 mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
16 int 80h ; Вызов ядра

```

Рис. 4.4: Открытие файла и вставка программы.

4.2 Работа с транслятором NASM

Я преобразую текст программы для вывода “Hello world!” в объектный код с помощью транслятора NASM, используя команду `nasm -f elf hello.asm`. Параметр `-f` сообщает транслятору `nasm`, что необходимо создать бинарный файл в формате ELF (рис. 4.5). Затем я проверяю успешность выполнения команды с помощью утилиты `ls` и вижу, что файл “hello.o” действительно создан. (рис. 4.6)

```

aavihsloukh@dk1n22 ~/work/arch-pc/lab04 $ nasm -f elf hello.asm

```

Рис. 4.5: Компиляция файла.

```

aavihsloukh@dk1n22 ~/work/arch-pc/lab04 $ ls
hello.asm hello.o

```

Рис. 4.6: Проверка

4.3 Работа с расширенным синтаксисом командной строки NASM

Ввожу команду, которая скомпилирует файл `hello.asm` в файл `obj.o`, при этом в файл будут включены символы для отладки (ключ `-g`), также с помощью ключа `-l` будет создан файл листинга `list.lst` (рис. 4.7). Далее проверяю с помощью утилиты `ls` правильность выполнения команды.

```
aavihsloukh@dk1n22 ~/work/arch-pc/lab04 $ nasm -o obj.o -f elf -g -l list.lst hello.asm
aavihsloukh@dk1n22 ~/work/arch-pc/lab04 $ ls
hello.asm  hello.o  list.lst  obj.o
```

Рис. 4.7: Компиляция

4.4 Работа с компоновщиком LD

Передаю объектный файл `hello.o` на обработку компоновщику `LD`, чтобы получить исполняемый файл `hello` (рис. 4.8). Ключ `-o` задает имя создаваемого исполняемого файла. Далее проверяю с помощью утилиты `ls` правильность выполнения команды.

```
aavihsloukh@dk1n22 ~/work/arch-pc/lab04 $ ld -m elf_i386 hello.o -o hello
aavihsloukh@dk1n22 ~/work/arch-pc/lab04 $ ls
hello  hello.asm  hello.o  list.lst  obj.o
```

Рис. 4.8: Передача на обработку

4.5 Запуск исполняемого файла

Запускаю на выполнение созданный исполняемый файл `hello` (рис. 4.9).

```
aavihsloukh@dk1n22 ~/work/arch-pc/lab04 $ ./hello
Hello world!
```

Рис. 4.9: Запуск программы

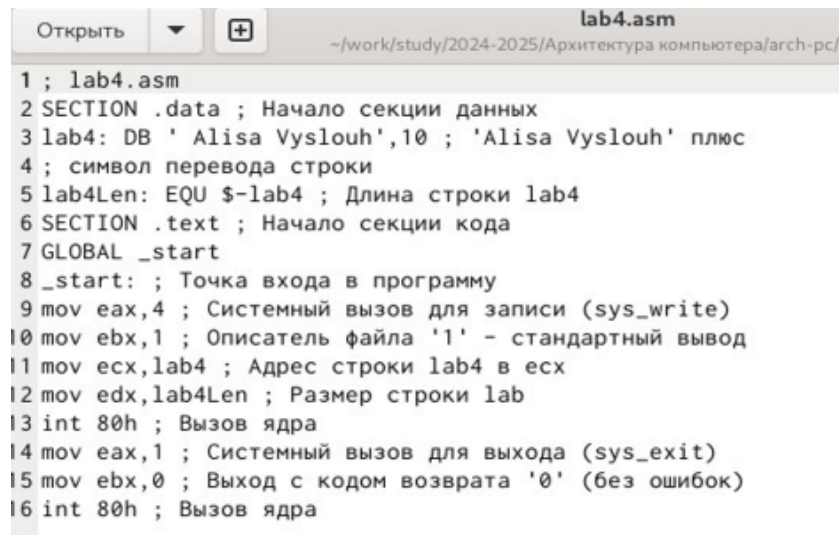
5 Задания для самостоятельной работы

С помощью утилиты `cp` создаю в текущем каталоге копию файла `hello.asm` с именем `lab4.asm` (рис. 5.1).

```
aavihsloukh@dk1n22 ~/work/arch-pc/lab04 $ cp hello.asm lab4.asm
```

Рис. 5.1: Создание копии

С помощью текстового редактора `gedit` открываю файл `lab4.asm` и вношу изменения в программу так, чтобы она выводила мои имя и фамилию (рис. 5.2).



```
lab4.asm
~/work/study/2024-2025/Архитектура компьютера/arch-pc/

1 ; lab4.asm
2 SECTION .data ; Начало секции данных
3 lab4: DB 'Alisa Vyslouh',10 ; 'Alisa Vyslouh' плюс
4 ; символ перевода строки
5 lab4Len: EQU $-lab4 ; Длина строки lab4
6 SECTION .text ; Начало секции кода
7 GLOBAL _start
8 _start: ; Точка входа в программу
9 mov eax,4 ; Системный вызов для записи (sys_write)
10 mov ebx,1 ; Описатель файла '1' - стандартный вывод
11 mov ecx,lab4 ; Адрес строки lab4 в ecx
12 mov edx,lab4Len ; Размер строки lab
13 int 80h ; Вызов ядра
14 mov eax,1 ; Системный вызов для выхода (sys_exit)
15 mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
16 int 80h ; Вызов ядра
```

Рис. 5.2: Открытие файла и его изменение

Компилирую текст программы в объектный файл. Проверяю с помощью утилиты `ls`, что файл `lab4.o` создан (рис. 5.3).

```
aavihsloukh@dk1n22 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04 $ ls
hello hello.asm hello.o lab4 lab4.asm lab4.o list.lst main obj.o presentation report
```

Рис. 5.3: Компиляция

Передаю объектный файл lab4.o на обработку компоновщику LD, чтобы получить исполняемый файл lab4 (рис. 5.4).

```
aavihsloukh@dk1n22 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04 $ ls
hello hello.asm hello.o lab4 lab4.asm lab4.o list.lst main obj.o presentation report
```

Рис. 5.4: Передача файла

Запускаю исполняемый файл lab5, на экран действительно выводятся мои имя и фамилия (рис. 5.5).

```
aavihsloukh@dk1n22 ~/work/arch-pc/lab04 $ ./lab5
Alisa Vyslouh
```

Рис. 5.5: Запуск программы

Копирую папку и соединяю ее с lab04. Удаляю лишние файлы после копирования и проверяю с помощью утилиты ls.(рис. 5.6).

```
aavihsloukh@dk1n22 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04 $ rm hello hello.o lab4 lab4.o list.lst main obj.o
rm: невозможно удалить 'hello': Нет такого файла или каталога
rm: невозможно удалить 'hello.o': Нет такого файла или каталога
rm: невозможно удалить 'list.lst': Нет такого файла или каталога
rm: невозможно удалить 'main': Нет такого файла или каталога
rm: невозможно удалить 'obj.o': Нет такого файла или каталога
aavihsloukh@dk1n22 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04 $ ls
hello.asm lab4.asm presentation report
```

Рис. 5.6: Удаление файлов и проверка

Ввожу последовательность команд git add . git commit и git push.(рис. 5.7).

```
aavihsloukh@dk1n22 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04 $ git add .
aavihsloukh@dk1n22 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04 $ git commit -m "Add fales for lab04"
[master 8997bd6] Add fales for lab04
2 files changed, 32 insertions(+)
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/lab4.asm
aavihsloukh@dk1n22 ~/work/study/2024-2025/Архитектура компьютера/arch-pc/labs/lab04 $ git push
Перечисление объектов: 9, готово.
Подсчет объектов: 100% (9/9), готово.
При сжатии изменений используется до 6 потоков
Сжатие объектов: 100% (6/6), готово.
Запись объектов: 100% (6/6), 1.00 KiB | 1.00 MiB/s, готово.
Total 6 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), completed with 2 local objects.
To github.com:aavihsloukh/study_2024-2025_arh-pc.git
97cd4a4..8997bd6 master -> master
```

Рис. 5.7: Загрузка файлов на гитхаб.

6 Выводы

При выполнении данной лабораторной работы я освоила процедуры компиляции и сборки программ, написанных на ассемблере NASM.