

Отчёт по лабораторной работе №4

Дисциплина: Архитектура компьютера

Панина Жанна Валерьевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	10
4.1	Программа Hello world!	10
4.2	Транслятор NASM	11
4.3	Расширенный синтаксис командной строки NASM	11
4.4	Компоновщик LD	12
4.5	Запуск исполняемого файла	12
4.6	Выполнение заданий для самостоятельной работы	13
5	Выводы	15
	Список литературы	16

Список иллюстраций

4.1	Создание текстового файла	10
4.2	Файл в gedit	10
4.3	Создание объектного файла	11
4.4	Создание файлов	11
4.5	Передача файла на обработку	12
4.6	Исполняемый файл main	12
4.7	Запуск исполняемого файла	12
4.8	Копия файла hello.asm	13
4.9	Копия файла hello.asm	13
4.10	Копия файла hello.asm	13
4.11	Передача файла на обработку	14
4.12	Запуск файла	14
4.13	Запуск файла	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 функциональными элементами любой ЭВМ являются центральный процессор, память и периферийные устройства. Взаимодействие этих устройств осуществляется через общую шину, к которой они подключены. Физически шина представляет собой большое количество проводников, соединяющих устройства друг с другом. В современных компьютерах проводники выполнены в виде электропроводящих дорожек на материнской плате. Основной задачей процессора является обработка информации, а также организация координации всех узлов компьютера. В состав центрального процессора входят следующие устройства:

арифметико-логическое устройство (АЛУ) — выполняет логические и арифметические действия, необходимые для обработки информации, хранящейся в памяти; устройство управления (УУ) — обеспечивает управление и контроль всех устройств компьютера; регистры — сверхбыстрая оперативная память небольшого объёма, входящая в состав процессора, для временного хранения промежуточных результатов выполнения инструкций; регистры процессора делятся на два типа: регистры общего назначения и специальные регистры. Для того, чтобы писать программы на ассемблере, необходимо знать, какие регистры процессора существуют и как их можно использовать. Большинство команд в программах написанных на ассемблере используют регистры в качестве операндов. Практически все команды представляют собой преобразование данных хранящихся в регистрах процессора, это например пересылка данных между регистрами или между регистрами и памятью, преобразование (арифметические или логические

операции) данных хранящихся в регистрах. Доступ к регистрам осуществляется не по адресам, как к основной памяти, а по именам. Каждый регистр процессора архитектуры x86 имеет свое название, состоящее из 2 или 3 букв латинского алфавита. В качестве примера приведем названия основных регистров общего назначения (именно эти регистры чаще всего используются при написании программ): 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 узлом ЭВМ является оперативное запоминающее устройство (ОЗУ). ОЗУ — это быстродействующее энергозависимое запоминающее устройство, которое напрямую взаимодействует с узлами процессора, предназначенное для хранения программ и данных, с которыми процессор непосредственно работает в текущий момент. ОЗУ состоит из одинаковых пронумерованных ячеек памяти. Номер ячейки памяти — это адрес хранящихся в ней данных. Периферийные устройства в составе ЭВМ:

устройства внешней памяти, которые предназначены для долговременного хранения больших объёмов данных. устройства ввода-вывода, которые обеспечивают взаимодействие ЦП с внешней средой. В основе вычислительного процесса ЭВМ лежит принцип программного управления. Это означает, что компьютер решает поставленную задачу как последовательность действий, записанных в виде программы.

Коды команд представляют собой многоразрядные двоичные комбинации из 0 и 1. В коде машинной команды можно выделить две части: операционную и адресную. В операционной части хранится код команды, которую необходимо выполнить. В адресной части хранятся данные или адреса данных, которые участвуют в выполнении данной операции. При выполнении каждой команды процессор выполняет определённую последовательность стандартных действий, которая называется командным циклом процессора. Он заключается в следующем:

формирование адреса в памяти очередной команды; считывание кода коман-

ды из памяти и её дешифрация; выполнение команды; переход к следующей команде. Язык ассемблера (assembly language, сокращённо asm) — машинно-ориентированный язык низкого уровня. NASM — это открытый проект ассемблера, версии которого доступны под различные операционные системы и который позволяет получать объектные файлы для этих систем. В NASM используется Intel-синтаксис и поддерживаются инструкции x86-64.

4 Выполнение лабораторной работы

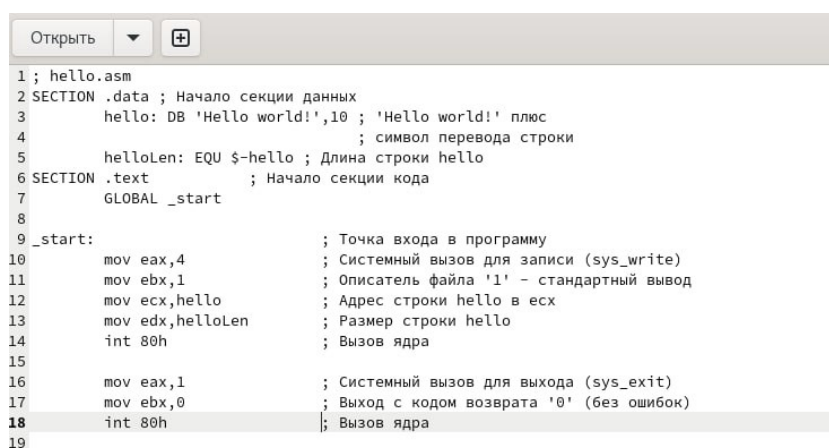
4.1 Программа Hello world!

Создаю каталог для работы с программами на языке ассемблера NASM, перехожу в созданный каталог. Создаю текстовый файл с именем hello.asm (рис. 4.1).

```
zvpanina@fedora:~$ mkdir -p ~/work/arch-pc/lab04
zvpanina@fedora:~$ cd ~/work/arch-pc/lab04
zvpanina@fedora:~/work/arch-pc/lab04$ touch hello.asm
zvpanina@fedora:~/work/arch-pc/lab04$ gedit hello.asm
zvpanina@fedora:~/work/arch-pc/lab04$ nasm -f elf hello.asm
```

Рис. 4.1: Создание текстового файла

Открываю этот файл с помощью любого текстового редактора gedit и ввожу в него следующий текст:(рис. 4.2).



```
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
9 _start: ; Точка входа в программу
10        mov eax,4 ; Системный вызов для записи (sys_write)
11        mov ebx,1 ; Описатель файла '1' - стандартный вывод
12        mov ecx,hello ; Адрес строки hello в ecx
13        mov edx,helloLen ; Размер строки hello
14        int 80h ; Вызов ядра
15
16        mov eax,1 ; Системный вызов для выхода (sys_exit)
17        mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
18        int 80h ; Вызов ядра
19
```

Рис. 4.2: Файл в gedit

4.2 Транслятор NASM

NASM превращает текст программы в объектный код. Например, для компиляции приведённого выше текста программы «Hello World» пишу: `nasm -f elf hello.asm`

Если текст программы набран без ошибок, то транслятор преобразует текст программы из файла `hello.asm` в объектный код, который запишется в файл `hello.o`. С помощью команды `ls` проверяю, что объектный файл действительно был создан под именем `hello.o`. (рис. 4.3).

```
zvpanina@fedora:~/work/arch-pc/lab04$ ls  
hello.asm hello.o
```

Рис. 4.3: Создание объектного файла

4.3 Расширенный синтаксис командной строки NASM

Полный вариант командной строки `nasm` выглядит следующим образом: `nasm [-@ косвенный_файл_настроек] [-о объектный_файл] [-f ↔ формат_объектного_файла] [-l листинг] [параметры...] [-] исходный_файл`. Выполняю следующую команду, которая скомпилирует исходный файл `hello.asm` в `obj.o` (опция `-o` позволяет задать имя объектного файла, в данном случае `obj.o`), при этом формат выходного файла будет `elf`, и в него будут включены символы для отладки (опция `-g`), кроме того, будет создан файл листинга `list.lst` (опция `-l`). С помощью команды `ls` проверяю, что файлы были созданы (рис. 4.4).

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

Рис. 4.4: Создание файлов

4.4 Компоновщик LD

Чтобы получить исполняемую программу, передаю объектный файл на обработку компоновщику и проверяю, что исполняемый файл hello был создан (рис. 4.5).

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

Рис. 4.5: Передача файла на обработку

Компоновщик ld не предполагает по умолчанию расширений для файлов, но принято использовать следующие расширения: • o – для объектных файлов; • без расширения – для исполняемых файлов; • tar – для файлов схемы программы; • lib – для библиотек.

Ключ -o с последующим значением задаёт в данном случае имя создаваемого исполняемого файла. Выполняю следующую команду (исполняемый файл будет иметь имя main, т.к. с помощью ключа -o мы задали имя main. Объектный файл, из которого собран этот исполняемый файл, имеет имя obj.o): (рис. 4.6).

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

Рис. 4.6: Исполняемый файл main

4.5 Запуск исполняемого файла

Запускаю на выполнение созданный исполняемый файл, находящийся в текущем каталоге, набрав в командной строке: ./hello (рис. 4.7).

```
zvpanina@fedora:~/work/arch-pc/lab04$ ./hello
Hello world!
```

Рис. 4.7: Запуск исполняемого файла

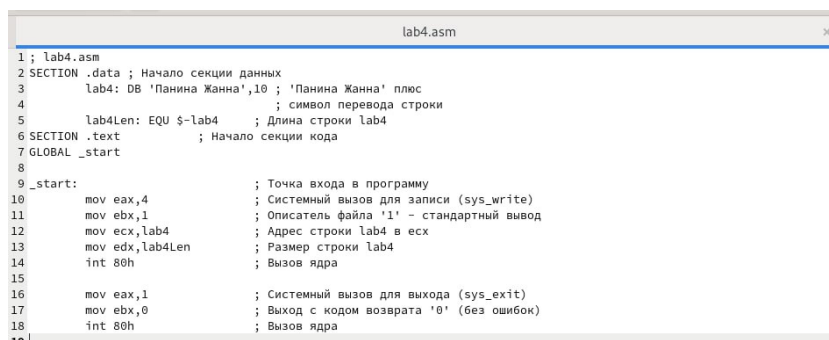
4.6 Выполнение заданий для самостоятельной работы

1. В каталоге ~/work/arch-pc/lab04 с помощью команды cp создаю копию файла hello.asm с именем lab4.asm (рис. 4.8).

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

Рис. 4.8: Копия файла hello.asm

2. С помощью текстового редактора gedit вношу изменения в текст программы в файле lab4.asm так, чтобы вместо Hello world! на экран выводилась строка с моими фамилией и именем (рис. 4.9).



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

Рис. 4.9: Копия файла hello.asm

3. Транслирую текст программы в объектный файл (рис. 4.10). Проверяю с помощью ls, что файл lab4.o создан.

```
zypanina@fedora:~/work/arch-pc/lab04$ nasm -f elf lab4.asm
zypanina@fedora:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  lab4.asm  lab4.o  list.lst  main  obj.o
zypanina@fedora:~/work/arch-pc/lab04$
```

Рис. 4.10: Копия файла hello.asm

Передаю объектный файл lab4.o на обработку компоновщику LD, чтобы получить исполняемый файл lab4 (рис. 4.11).

```

zvpanina@fedora:~/work/arch-pc/lab04$ ld -m elf_i386 lab4.o -o lab4
zvpanina@fedora:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  lab4  lab4.asm  lab4.o  list.lst  main  obj.o
zvpanina@fedora:~/work/arch-pc/lab04$

```

Рис. 4.11: Передача файла на обработку

Запускаю исполняемый файл lab4, на экране вижу свои имя и фамилию (рис. 4.12).

```

zvpanina@fedora:~/work/arch-pc/lab04$ ./lab4
Панина Жанна
zvpanina@fedora:~/work/arch-pc/lab04$

```

Рис. 4.12: Запуск файла

4. Копирую файлы hello.asm и lab4.asm в свой локальный репозиторий в каталог ~/work/study/2024-2025/“Архитектура компьютера”/arch-pc/labs/lab04/ (рис. 4.13).

```

zvpanina@fedora: ~/work/arch-pc/lab04$ cp hello.asm lab4.asm ~/work/study/2024-2025/"Архитектура компьютера"/arch-pc/labs/lab04/
zvpanina@fedora: ~/work/arch-pc/lab04$ ls ~/work/study/2024-2025/"Архитектура компьютера"/arch-pc/labs/lab04/
hello.asm  lab4.asm  presentation  report
zvpanina@fedora: ~/work/arch-pc/lab04$

```

Рис. 4.13: Запуск файла

Загружаю файлы на Github.

5 Выводы

Во время выполнения лабораторной работы я освоила процедуры компиляции и сборки программ, написанных на ассемблере NASM.

Список литературы

Архитектура ЭВМ