Отчёта по лабораторной работе №4

Дисциплина: архитектура компьютера

Закиров Нурислам Дамирович

Содержание

Список иллюстраций

1 Цель работы

Цель данной лабораторной работы состоит в том, чтобы научиться компиляции и сборке программ в 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 каче- стве 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: 1. формирование адреса в памяти очередной команды; 2. считывание кода команды из памяти и её дешифрация; 3. выполнение команды; 4. 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 Создание программы Hello world!

Создаем новую директорию с помощью mkdir и перемещаемся туда, для дальнейшей работы в ней. (рис. 4.1).

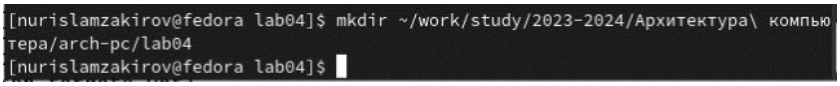

Рис. 4.1: Создание директории

Я использую Touch для создания пустого текстового файла hello.asm в текущем каталоге. (рис. 4.2).

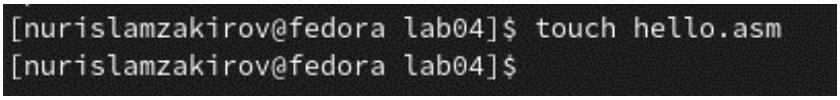

Рис. 4.2: Создание пустого файла

Открываю созданный файл в текстовом редакторе mousepad (рис. 4.3).

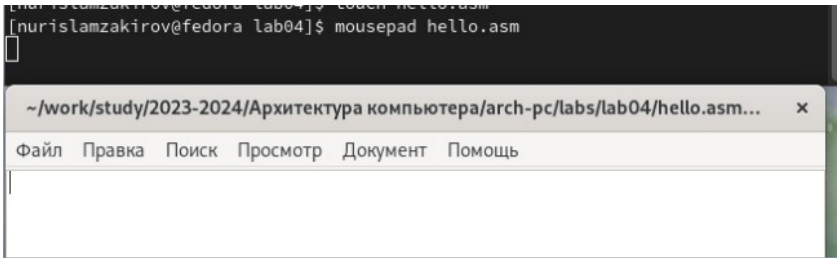

Рис. 4.3: Открытие файла в текстовом редакторе

Заполняю файл, вставляя в него программу для вывода “Hello word!” (рис. 4.4).

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

Рис. 4.4: Заполнение файла

4.2 Работа с транслятором NASM

Я использую команду nasm -f elf hello.asm для перевода текста программы «Hello world!» в объектный код. Ключ -f указывает транслятору NASM создать бинарный файл в формате ELF (рис. 4.5). Далее проверяю корректность выполнения команды с помощью утилиты ls.

```
[nurislamzakirov@fedora lab04]$ nasm -f elf hello.asm
[nurislamzakirov@fedora lab04]$ ls
hello.asm  hello.o  presentation  report
[nurislamzakirov@fedora lab04]$
```

Рис. 4.5: Компиляция текста программы

4.3 Работа с расширенным синтаксисом командной строки NASM

Вводим команду, которая скомпилирует файл hello.asm в файл obj.o, кроме того в файл будут включены символы для отладки (ключ -g), и с помощью ключа -l будет создан файл листинга list.lst (рис. 4.6). Далее проверяю с помощью утилиты ls корректность выполнения команды.

Рис. 4.6: Компиляция текста программы

4.4 Работа с компоновщиком LD

Я передаю объектный файл hello.o компоновщику LD, чтобы он мог получить исполняемый файл hello (рис. 4.7). Ключ -o задает имя создаваемого исполняемого файла. Далее проверяю с помощью утилиты ls корректность выполнения команды.

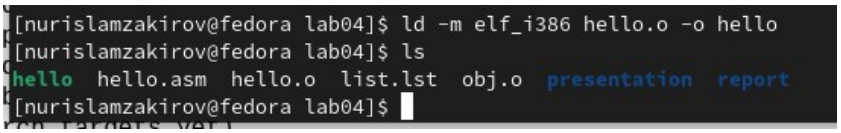

Рис. 4.7: Передача объектного файла на обработку компоновщику

Выполняю следующую команду (рис. 4.8). Файл, который выполняется, будет иметь имя главного, потому что значение главного было задано после ключа -o. Obj.o — это имя объектного файла, из которого состоит этот исполняемый файл.

Рис. 4.8: Передача объектного файла на обработку компоновщику

4.5 Запуск исполняемого файла

Запускаем созданный файл hello (рис. 4.9).

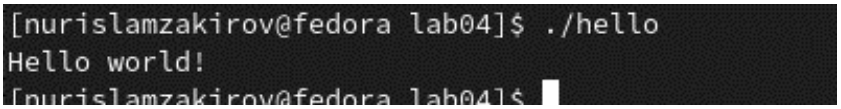

Рис. 4.9: Запуск исполняемого файла

4.6 Выполнение заданий для самостоятельной работы.

С помощью утилиты cp создаю копию файла hello.asm с именем lab4.asm в этом каталоге. (рис. 4.10).

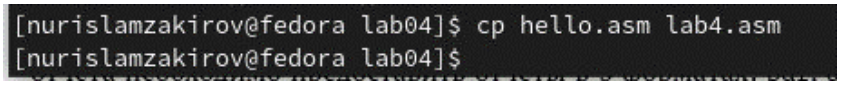

Рис. 4.10: Создание копии файла

Я открываю файл lab4.asm с помощью текстового редактора Mousepad и вношу изменения в программу, чтобы она выдавала моё имя и фамилию. (рис. 4.11).

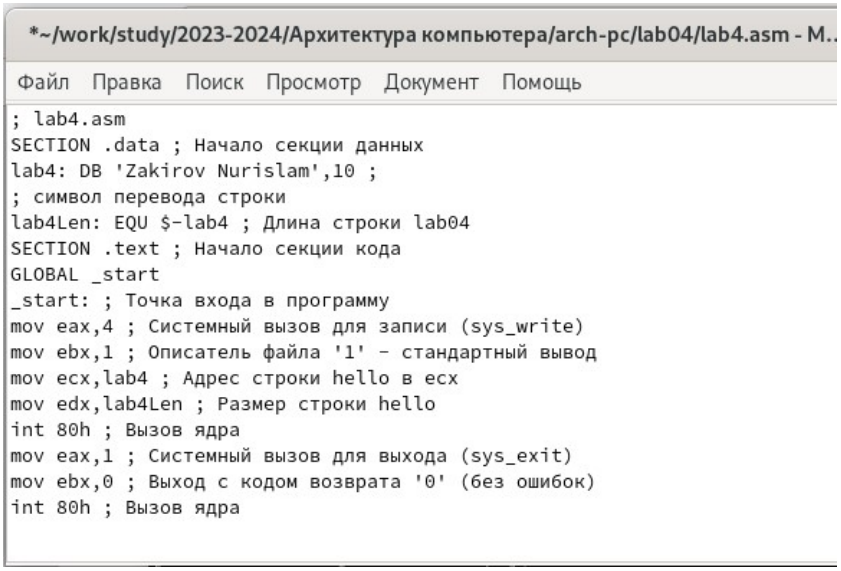

Рис. 4.11: Изменение программы

Компилирую текст программы в объектный файл (рис. 4.12). Проверяю с помощью утилиты ls, что файл lab4.o создан.

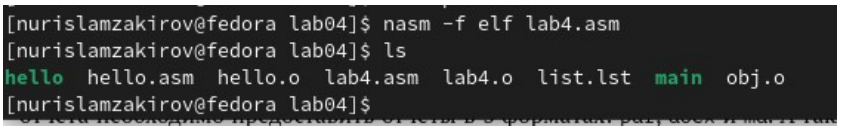

Рис. 4.12: Компиляция текста программы

Я передаю объектный файл lab4.o компоновщику LD, чтобы он мог получить исполняемый файл lab4. (рис. 4.13).

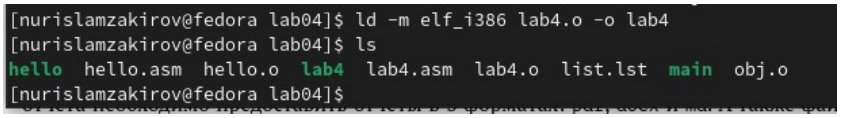

Рис. 4.13: Передача объектного файла на обработку компоновщику

Запускаю исполняемый файл lab4. Выводятся мои имя и фамилия (рис. 4.14).

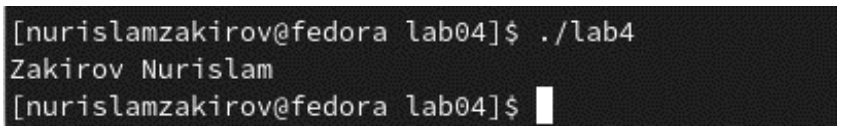

Рис. 4.14: Запуск исполняемого файла

С помощью команд git add . и git commit добовляем файлы на GitHub. (рис. 4.15).

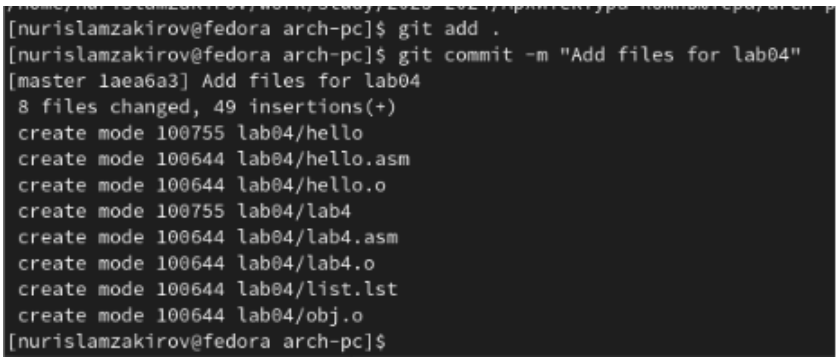

Рис. 4.15: Добавление файлов на GitHub

Отправляю файлы на сервер с помощью команды git push (рис. 4.16).

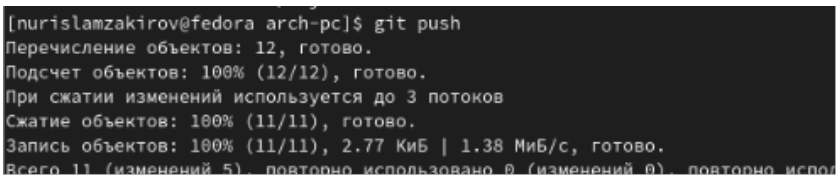

Рис. 4.16: Отправка файлов

5 Выводы

Во время выполнения данной лабораторной работы я научился компиляции и сборке программ в ассемблере NASM.

6 Список литературы

1. https://esystem.rudn.ru/pluginfile.php/1584628/mod_resource/content/1/%D0%9B%D0%B0%D