Отчёта по лабораторной работе №4

Дисциплина: архитектура компьютера

Карачевцева Елизавета Васильевна

Содержание

Список иллюстраций

1 Цель работы

Цель данной лабораторной работы - освоить 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 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!

Создаю каталог, в котором буду работать

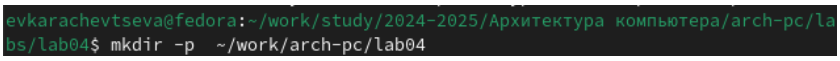

Рис. 4.1: Создание каталога

С помощью утилиты cd перемещаюсь в каталог, в котором буду работать.

Рис. 4.2: Перемещение между директориями

Создаю в текущем каталоге пустой текстовый файл hello.asm с помощью утилиты touch.

Рис. 4.3: Создание пустого файла

Открываю созданный файл в текстовом редакторе.

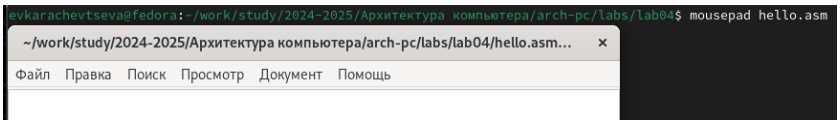

Рис. 4.4: Открытие файла в текстовом редакторе

Заполняю файл, вставляя в него программу для вывода “Hello word!”.

```
; 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. 4.5: Заполнение файла

4.2 Работа с транслятором NASM

Превращаю текст программы для вывода “Hello world!” в объектный код с помощью транслятора NASM, используя команду nasm -f elf hello.asm, ключ -f указывает транслятору nasm, что требуется создать бинарный файл в формате ELF. Далее проверяю правильность выполнения команды с помощью утилиты ls: действительно, создан файл “hello.o”.

```
evkarachevtseva@fedora:~/work/arch-pc/lab04$ nasm -f elf hello.asm
evkarachevtseva@fedora:~/work/arch-pc/lab04$ ls
hello.asm  hello.o
```

Рис. 4.6: Компиляция текста программы

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. Далее проверяю с помощью утилиты ls правильность выполнения команды.

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

Рис. 4.7: Компиляция текста программы

4.4 Работа с компоновщиком LD

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

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

Рис. 4.8: Передача объектного файла на обработку компоновщику

Выполняю следующую команду. Исполняемый файл будет иметь имя main, т.к. после ключа -o было задано значение main. Объектный файл, из которого собран этот исполняемый файл, имеет имя obj.o

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

Рис. 4.9: Передача объектного файла на обработку компоновщику

4.5 Запуск исполняемого файла

Запускаю на выполнение созданный исполняемый файл hello.

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

Рис. 4.10: Запуск исполняемого файла

4.6 Выполнение заданий для самостоятельной работы.

С помощью утилиты cp создаю в текущем каталоге копию файла hello.asm с именем lab4.asm.

```
evkarachevtseva@fedora:~/work/arch-pc/lab04$ cp hello.asm lab4.asm
```

Рис. 4.11: Создание копии файла

С помощью текстового редактора mousepad открываю файл lab4.asm и вношу изменения в программу так, чтобы она выводила мои имя и фамилию.

```
; lab4.asm
SECTION .data ; Начало секции данных
        lab4: DB 'Elizaveta Karachevtseva',10

        lab4Len: EQU $-lab4 ; Длина строки lab4
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,lab4 ; Адрес строки lab4 в ecx
        mov edx,lab4Len ; Размер строки lab
        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. 4.12: Изменение программы

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

Рис. 4.13: Компиляция текста программы

Передаю объектный файл lab4.o на обработку компоновщику LD, чтобы получить исполняемый файл lab4.

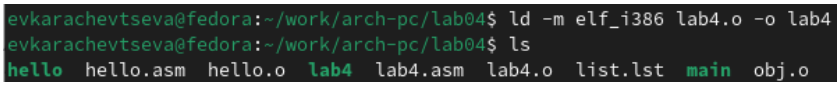

Рис. 4.14: Передача объектного файла на обработку компоновщику

Запускаю исполняемый файл lab4, на экран действительно выводятся мои имя и фамилия.

Рис. 4.15: Запуск исполняемого файла

Создаю копии файлов в нужном каталоге.

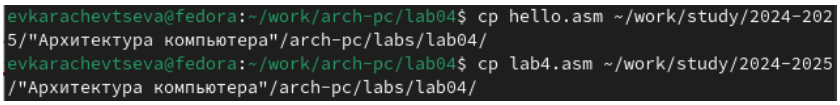

Рис. 4.16: Создание копий файлов в другом каталоге

Удаляю лишние файлы в текущем каталоге с помощью утилиты rm, ведь копии файлов остались в другой директории.

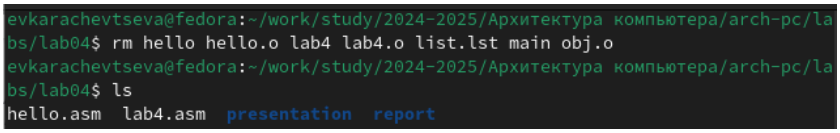

Рис. 4.17: Удаление лишних файлов в текущем каталоге

С помощью команд git add . и git commit добавляю файлы на GitHub, комментируя действие как добавление файлов для лабораторной работы №4.

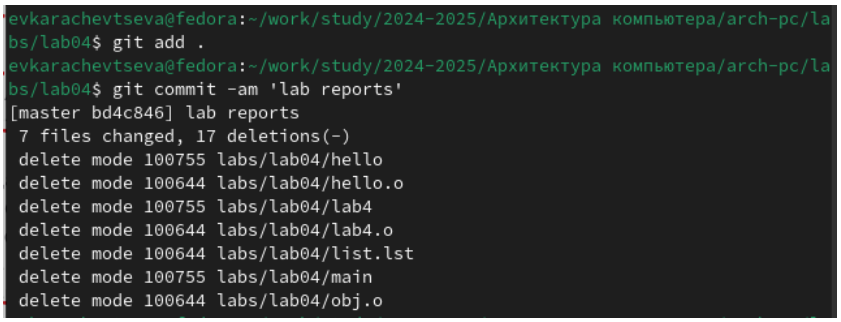

Рис. 4.18: Добавление файлов на GitHub

Отправляю файлы на сервер с помощью команды git push.

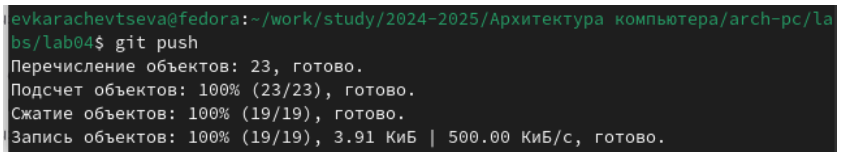

Рис. 4.19: Отправка файлов

5 Выводы

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

6 Список литературы

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