Отчёт по лабораторной работе №4

Дисциплина: Архитектура компьютера

Зарицкая Марина Петровна

Содержание

# 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 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!

С помощью команды cd перемещаюсь в каталог, в котором буду работать (рис. -4.1).

fig:
Перемещение между директориями

Перемещение между директориями

Создаю в текущем каталоге пустой текстовый файл hello.asm, используя touch (рис. -4.2).

fig:
Создание пустого текстового файла

Создание пустого текстового файла

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

fig:
Открытие файла в текстовом редакторе

Открытие файла в текстовом редакторе

Ввожу в файл программу для вывода “Hello word!” (рис. -4.4).

fig:
Заполнение файла

Заполнение файла

## 4.2 Работа с транслятором NASM

Компилирую текст программы для вывода “Hello world!” в объектный код с помощью транслятора NASM, далее проверяю правильность выполнения команды с помощью ls.(рис. -4.5)

fig:
Компиляция текста программы

Компиляция текста программы

## 4.3 Работа с расширенным синтаксисом командной строки NASM

Выполняю команду, которая скомпилирует файл hello.asm в 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. -4.6).

fig:
Компиляция текста программы

Компиляция текста программы

## 4.4 Работа с компоновщиком LD

Передаю объектный файл hello.o на обработку компоновщику LD, чтобы получить исполняемый файл hello. Затем с помощью ls проверяю, что исполняемый файл hello был создан (рис. -4.7).

fig:
Передача объектного файла на обработку компоновщику

Передача объектного файла на обработку компоновщику

Выполняю следующую команду (рис. -4.8). Исполняемый файл будет иметь имя main, т.к. после ключа -о было задано значение main. Объектный файл, из которого собран этот исполняемый файл, имеет имя obj.o

fig:
Передача объектного файла на обработку компоновщику

Передача объектного файла на обработку компоновщику

## 4.5 Запуск исполняемого файла

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

fig:
Запуск исполняемого файла

Запуск исполняемого файла

## 4.6 Выполнение заданий для самостоятельной работы.

С помощью команды cp создаю в текущем каталоге копию файла hello.asm с именем lab4.asm. Проверяю с помощью ls (рис. -4.10).

fig:
Создание копии файла

Создание копии файла

С помощью текстового редактора gedit открываю файл lab4.asm и вношу изменения в программу так, чтобы вместо Hello world! на экран выводилась строка с моими фамилией и именем. (рис. -4.11).

fig:
Изменение программы

Изменение программы

Транслирую текст программы в объектный файл и проверяю с помощью утилиты ls, что файл lab4.o создан. (рис. -4.12).

fig:
Компиляция текста программы

Компиляция текста программы

Передаю объектный файл lab4.o на обработку компоновщику LD, чтобы получить исполняемый файл lab4 (рис. -4.13).

fig:
Передача объектного файла на обработку компоновщику LD

Передача объектного файла на обработку компоновщику LD

Запускаю исполняемый файл lab4, на экран действительно выводятся мои фамилия и имя (рис. -4.14).

fig:
Запуск исполняемого файла

Запуск исполняемого файла

Копирую файлы hello.asm и lab4.asm в свой локальный репозиторий в каталог ~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04 (рис. -4.15).

fig:
Копия файлов в локальный репозиторий

Копия файлов в локальный репозиторий

С помощью команд git add . и git commit добавляю файлы на GitHub (рис. -4.16).

fig:
Добавление файлов на GitHub

Добавление файлов на GitHub

Отправляю файлы на сервер с помощью команды git push (рис. -4.17).

fig:
Отправка файлов

Отправка файлов

# 5 Выводы

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