Лабораторная работа №4

НПИбд-01-25 №1032252598

Иванова Ангелина Олеговна

Содержание

# 1. Цель работы

Целью данной лабораторной работы является освоение процедуры компиляции и сборки программ, которые написаны на ассемблере NASM

# 2. Теоретическое введение

### 2.0.1 Основные принципы работы компьютера

Основными функциональными элементами любой электронно-вычислительной машины (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 (системной) плате.

Основной задачей процессора является обработка информации, а также организация координации всех узлов компьютера. В состав *центрального процессора (ЦП)* 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 (половинки 16-битных регистров). Например, AH (high AX) — старшие 8 бит регистра AX, AL (low AX) — младшие 8 бит регистра AX.

Таким образом можно отметить, что вы можете написать в своей программе, например, такие команды (mov – команда пересылки данных на языке ассемблера):

Обе команды поместят в регистр AX число 1. Разница будет заключаться только в том, что вторая команда обнулит старшие разряды регистра EAX, то есть после выполнения второй команды в регистре EAX будет число 1. А первая команда оставит в старших разрядах регистра EAX старые данные. И если там были данные, отличные от нуля, то после выполнения первой команды в регистре EAX будет какое-то число, но не 1. А вот в регистре AX будет число 1.

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.

В состав ЭВМ также входят периферийные устройства, которые можно разделить на: - устройства внешней памяти, которые предназначены для долговременного хранения больших объёмов данных (жёсткие диски, твердотельные накопители, магнитные ленты); - 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. Программа состоит из машинных команд, которые указывают, какие операции и над какими данными (или операндами), в какой последовательности необходимо выполнить.

Набор машинных команд определяется устройством конкретного процессора. 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. В самом общем виде 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.

Данный алгоритм позволяет выполнить хранящуюся в ОЗУ программу. Кроме того, в зависимости от команды при её выполнении могут проходить не все этапы.

## 2.1 Ассемблер и язык ассемблера

Язык ассемблера (assembly language, сокращённо asm) — машинно-ориентированный язык низкого уровня. Можно считать, что он больше любых других языков приближен к архитектуре ЭВМ и её аппаратным возможностям, что позволяет получить к ним более полный доступ, нежели в языках высокого уровня, таких как C/C++, Perl, Python и пр. Заметим, что получить полный доступ к ресурсам компьютера в современных архитектурах нельзя, самым низким уровнем работы прикладной программы является обращение напрямую к ядру операционной системы. Именно на этом уровне и работают программы, написанные на ассемблере. Но в отличие от языков высокого уровня ассемблерная программа содержит только тот код, который ввёл программист. Таким образом язык ассемблера — это язык, с помощью которого понятным для человека образом пишутся команды для процессора.

Следует отметить, что процессор понимает не команды ассемблера, а последовательности из нулей и единиц — машинные коды. До появления языков ассемблера программистам приходилось писать программы, используя только лишь машинные коды, которые были крайне сложны для запоминания, так как представляли собой числа, записанные в двоичной или шестнадцатеричной системе счисления. Преобразование или трансляция команд с языка ассемблера в исполняемый машинный код осуществляется специальной программой транслятором — Ассемблер. Программы, написанные на языке ассемблера, не уступают в качестве и скорости программам, написанным на машинном языке, так как транслятор просто переводит мнемонические обозначения команд в последовательности бит (нулей и единиц).

Используемые мнемоники обычно одинаковы для всех процессоров одной архитектуры или семейства архитектур (среди широко известных — мнемоники процессоров и контроллеров x86, ARM, SPARC, PowerPC,M68k). Таким образом для каждой архитектуры существует свой ассемблер и, соответственно, свой язык ассемблера. Наиболее распространёнными ассемблерами для архитектуры x86 являются: - для DOS/Windows: Borland Turbo Assembler (TASM), Microsoft Macro Assembler (MASM) и Watcom assembler (WASM); - для GNU/Linux: gas (GNU Assembler), использующий AT&T-синтаксис, в отличие от большинства других популярных ассемблеров, которые используют Intel-синтаксис.

В курсе Архитектуры компьютера будет использоваться ассемблер NASM (Netwide Assembler)

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.

Типичный формат записи команд NASM имеет вид: [метка:] мнемокод [операнд {, операнд}] [; комментарий]

Здесь мнемокод — непосредственно мнемоника инструкции процессору, которая является обязательной частью команды. Операндами могут быть числа, данные, адреса регистров или адреса оперативной памяти. Метка — это идентификатор, с которым ассемблер ассоциирует некоторое число, чаще всего адрес в памяти. Т.о. метка перед командой связана с адресом данной команды.

Допустимыми символами в метках являются буквы, цифры, а также следующие символы: \_, $, #, @,~,. и ?.

Начинаться метка или идентификатор могут с буквы, ., \_ и ?. Перед идентификаторами, которые пишутся как зарезервированные слова, нужно писать $, чтобы компилятор трактовал его верно (так называемое экранирование). Максимальная длина идентификатора 4095 символов.

Программа на языке ассемблера также может содержать директивы — инструкции, не переводящиеся непосредственно в машинные команды, а управляющие работой транслятора.Например, директивы используются для определения данных (констант и переменных) и обычно пишутся большими буквами.

## 2.2 Процесс создания и обработки программы на языке ассемблера

В процессе создания ассемблерной программы можно выделить четыре шага:

* Набор текста программы в текстовом редакторе и сохранение её в отдельном файле. Каждый файл имеет свой тип (или расширение), который определяет назначение файла. Файлы с исходным текстом программ на языке ассемблера имеют тип asm.
* Трансляция — преобразование с помощью транслятора, например nasm, текста программы в машинный код, называемый объектным. На данном этапе также может быть получен листинг программы, содержащий кроме текста программы различную дополнительную информацию, созданную транслятором. Тип объектного файла — o, файла листинга — lst.
* Компоновка или линковка — этап обработки объектного кода компоновщиком (ld), который принимает на вход объектные файлы и собирает по ним исполняемый файл. Исполняемый файл обычно не имеет расширения. Кроме того, можно получить файл карты загрузки программы в ОЗУ, имеющий расширение map.
* Запуск программы. Конечной целью является работоспособный исполняемый файл. Ошибки на предыдущих этапах могут привести к некорректной работе программы, поэтому может присутствовать этап отладки программы при помощи специальной программы — отладчика. При нахождении ошибки необходимо провести коррекцию программы, начиная с первого шага.

Из-за специфики программирования, а также по традиции для создания программ на языке ассемблера обычно пользуются утилитами командной строки (хотя поддержка ассемблера есть в некоторых универсальных интегрированных средах)

# 3. Выполнение лабораторной работы

## 3.1 Задания лабораторной работы

### 3.1.1 Программа Hello world!

Создали каталог для работы с программами на языке ассемблера NASM и перешли в него

|  |
| --- |
| Создание и переход к каталогу  Рисунок 1: Создание и переход к каталогу |

Создали текстовый файл с именем hello.asm и открыли его с помощью любого текстового редактора gedit.

|  |
| --- |
| Создание файла и его открытие  Рисунок 2: Создание файла и его открытие |

Ввели в файл предоставленный нам текст.

|  |
| --- |
| Отредактированный файл  Рисунок 3: Отредактированный файл |

### 3.1.2 Транслятор NASM

Преобразовали с помощью NASM текст программы из файла hello.asm в объектный код, который записался в файл hello.o. С помощью команды ls проверили корректность созданных файлов.

|  |
| --- |
| Преобразование текст программы в объектный код и проверка файлов  Рисунок 4: Преобразование текст программы в объектный код и проверка файлов |

Объектный файл носит имя hello.o

### 3.1.3 Расширенный синтаксис командной строки NASM

Выполнили следующую команду: nasm -o obj.o -f elf -g -l list.lst hello.asm Данная команда скомпилировала исходный файл hello.asm в obj.o (опция -o позволяет задать имя объектного файла, в данном случае obj.o), при этом формат выходного файла является elf, и в него включены символы для отладки (опция -g), кроме того, создался файл листинга list.lst (опция -l). С помощью команды ls проверили, что файлы были созданы.

|  |
| --- |
| Использование команды и проверка файлов  Рисунок 5: Использование команды и проверка файлов |

### 3.1.4 Компоновщик LD

Чтобы получить исполняемую программу, объектный файл необходимо передать на обработку компоновщику с помощью команды: ld -m elf\_i386 hello.o -o hello С помощью команды ls проверили, что исполняемый файл hello был создан.

|  |
| --- |
| Передача файла на обработку компоновщику и проверка создания файла hello  Рисунок 6: Передача файла на обработку компоновщику и проверка создания файла hello |

Выполнили следующую команду: ld -m elf\_i386 obj.o -o main

|  |
| --- |
| Использование команды и проверка файлов  Рисунок 7: Использование команды и проверка файлов |

Исполняемый файл будет иметь имя main, а объектный файл из которого собран этот исполняемый файл – это obj.o

### 3.1.5 Запуск исполняемого файла

Запустили на выполнение созданный исполняемый файл

|  |
| --- |
| Запуск исполняемого файла  Рисунок 8: Запуск исполняемого файла |

## 3.2 Задание для самостоятельной работы

1. В каталоге ~/work/arch-pc/lab04 с помощью команды cp создади копию файла hello.asm с именем lab4.asm

|  |
| --- |
| Копирование файла  Рисунок 9: Копирование файла |

1. С помощью текстового редактора gedit внесли изменения в текст программы в файле lab4.asm так, чтобы вместо Hello world! на экран выводилась строка с моей фамилией и именем

|  |
| --- |
| Открытие и редоктирование файла  Рисунок 10: Открытие и редоктирование файла |

1. Оттранслировали полученный текст программы lab4.asm в объектный файл

|  |
| --- |
| Преобразование в объектный файл  Рисунок 11: Преобразование в объектный файл |

* Выполнили компоновку объектного файла

|  |
| --- |
| Компановка оъектного файла  Рисунок 12: Компановка оъектного файла |

* Запустили получившийся исполняемый файл и получили корректный вывод программы

|  |
| --- |
| Запуск и вывод программы  Рисунок 13: Запуск и вывод программы |

1. Скопировали файлы hello.asm и lab4.asm в локальный репозиторий в каталог ~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/.

|  |
| --- |
| Копирование файлов  Рисунок 14: Копирование файлов |

* Загрузили файлы на Github

# 4. Выводы

Освоили процедуры компиляции и сборки программ, которые написаны на ассемблере NASM. НАписали свою первою программу на асемблере и запустили её.