Отчёт по лабораторной работе №4

Дисциплина: Архитектура компьютера

Долгаев Евгений Сергеевич НММбд-01-24

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Освоение процедуры компиляции и сборки программ, написанных на ассемблере NASM.

2 Задание

1) Выполнение лабораторной работы
    1) Программа Hello world!
    2) Транслятор NASM
    3) Расширенный синтаксис командной строки NASM
    4) Компоновщик LD
    5) Запуск исполняемого файла
2) Задания для самостоятельной работы

3 Теоретическое введение

3.1 Основные принципы работы компьютера

Основными функциональными элементами любой электронно-вычислительной машины(ЭВМ) являются центральный процессор, память и периферийные устройства (рис. 3.1).

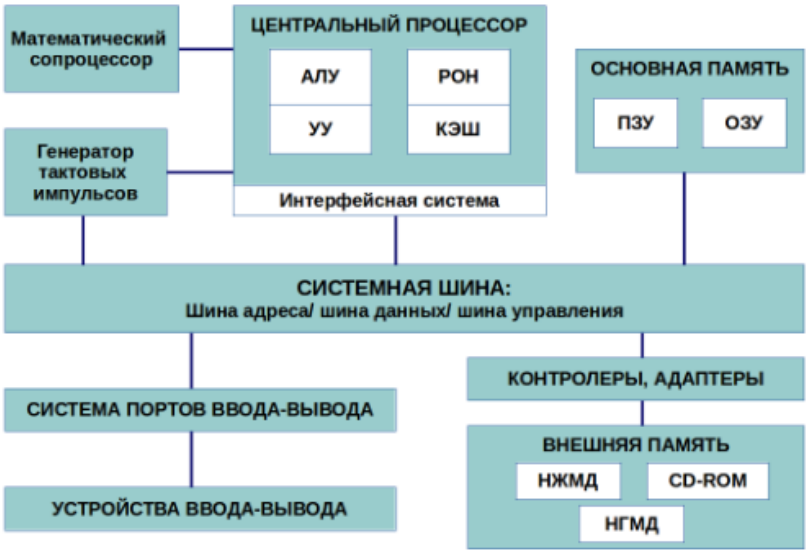

Рис. 3.1: Структурная схема ЭВМ

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:

1. арифметико-логическое устройство (АЛУ) — выполняет логические и арифметические действия, необходимые для обработки информации, хранящейся в памяти;

2. устройство управления (УУ) — обеспечивает управление и контроль всех устройств компьютера;
3. 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):

1. RAX, RCX, RDX, RBX, RSI, RDI — 64-битные

2. EAX, ECX, EDX, EBX, ESI, EDI — 32-битные
3. AX, CX, DX, BX, SI, DI — 16-битные
4. 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.

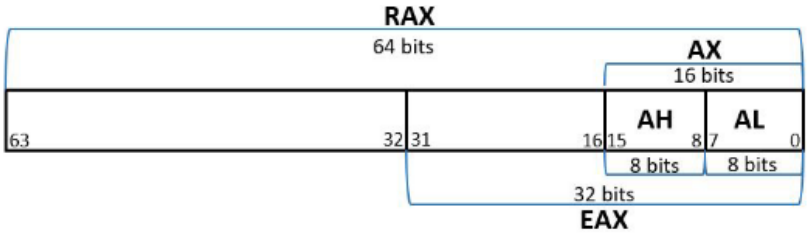

Рис. 3.2: 64-битный регистр процессора ‘RAX’

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

```
mov ax, 1

mov eax, 1
```

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:

1. устройства внешней памяти, которые предназначены для долговременного хранения больших объёмов данных (жёсткие диски, твердотельные накопители, магнитные ленты);
2. 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.

3.2 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:

1. для DOS/Windows: Borland Turbo Assembler (TASM), Microsoft Macro Assembler (MASM) и Watcom assembler (WASM);
2. для GNU/Linux: gas (GNU Assembler), использующий AT&T-синтаксис, в отличие от большинства других популярных ассемблеров, которые используют Intel-синтаксис.

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.

3.3 Процесс создания и обработки программы на языке ассемблера

Процесс создания ассемблерной программы можно изобразить в виде следующей схемы (рис. 3.3).

Рис. 3.3: Процесс создания ассемблерной программы

В процессе создания ассемблерной программы можно выделить четыре шага:

1. 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.
2. Трансляция — преобразование с помощью транслятора, например nasm, текста программы в машинный код, называемый объ-

ектным. 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.

3. 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.
4. 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).

4 Выполнение лабораторной работы

Рассмотрим пример простой программы на языке ассемблера NASM. Традиционно первая программа выводит приветственное сообщение Hello world! на экран. Создадим каталог для работы с программами на языке ассемблера NASM и перейдём в него (рис. 4.1):

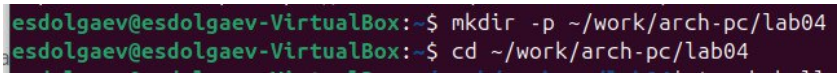

Рис. 4.1: Создание нужного каталога

Создим текстовый файл с именем hello.asm и откроем его (рис. 4.2):

Рис. 4.2: Создание файла hello.asm

и введём в него следующий текст(рис. 4.3):

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

Рис. 4.3: Код программы

Для компиляции приведённого выше текста программы «Hello World» необходимо написать (рис. 4.4):

Рис. 4.4: Компиляция программы

Если текст программы набран без ошибок, то транслятор преобразует текст программы из файла hello.asm в объектный код, который запишется в файл hello.o. Таким образом, имена всех файлов получаются из имени входного файла и расширения по умолчанию. Выполните следующую команду и проверим, что файлы были созданы (рис. 4.5):

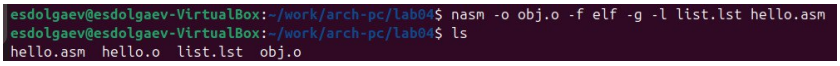

Рис. 4.5: Компиляция исходного файла в новый

Данная команда 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).

Чтобы получить исполняемую программу, передадим объектный файл на обработку компоновщику и проверим, что исполняемый файл был создан (рис. 4.6):

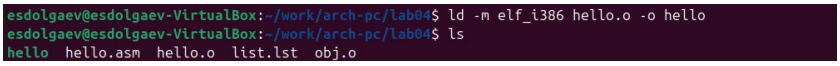

Рис. 4.6: Обработка компановщиком

Ключ -o с последующим значением задаёт в данном случае имя создаваемого исполняемого файла. Выполним следующую команду (рис. 4.7):

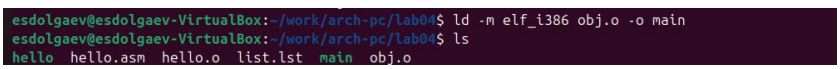

Рис. 4.7: Создание исполняемого файла

Таким образом, исполняемый файл имеет имя main, а объектный obj.o(или hello и hello.o (рис. 4.6)). Запустить на выполнение созданный исполняемый файл, находящийся в текущем каталоге, можно, набрав в командной строке (рис. 4.8):

Рис. 4.8: Запуск исполняемого файла

Привступим к выполнению заданий для самостоятельной работы. В каталоге ~/work/arch-pc/lab04 с помощью команды cp создим копию файла hello.asm с именем lab4.asm(рис. 4.9):

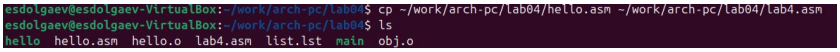

Рис. 4.9: Создание копии файла

С помощью текстового редактора внесём изменения в текст программы в файле lab4.asm так, чтобы вместо Hello world! на экран выводилась строка с фамилией и именем(рис. 4.10).

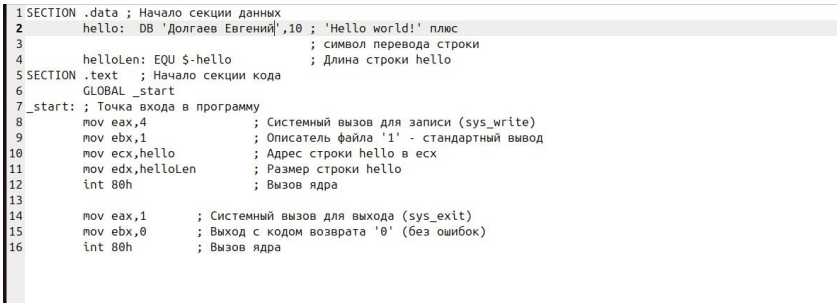

Рис. 4.10: Имзменение программы

Оттранслируем полученный текст программы lab4.asm в объектный файл. Выполним компоновку объектного файла и запустим получившийся исполняемый файл(рис. 4.11, 4.12, 4.12).

Рис. 4.11: Получение объектного файла

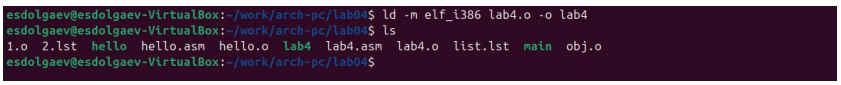

Рис. 4.12: Компановка объектного файла

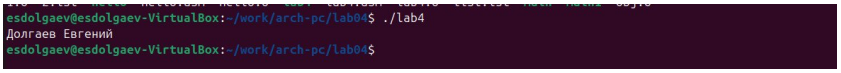

Рис. 4.13: Запуск исполняемого файла

Скопируем файлы hello.asm и lab4.asm в локальный репозиторий в каталог ~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/. Загрузим файлы на Github(рис. 4.14, 4.15).

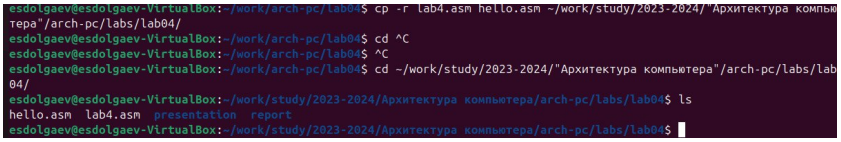

Рис. 4.14: Копирование в локальный репозиторий

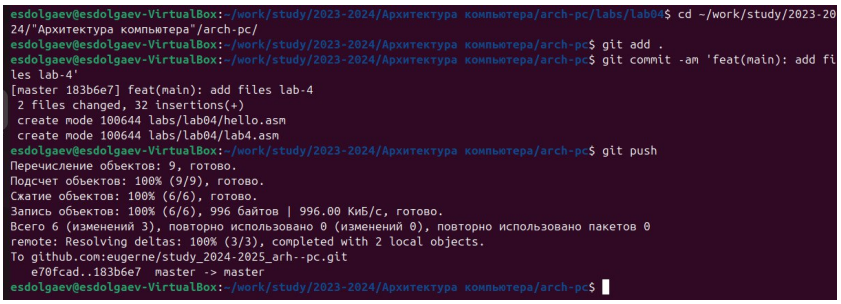

Рис. 4.15: Загрузка на Github

5 Выводы

Благодаря этой лабораторной работе, я освоил компиляцию и сборку программ, написанных на ассемблере NASM.