

Отчёт по лабораторной работе №5

Простейший вариант

Поляков Глеб

Содержание

1	Ц е л ь р а б о т ы	1
2	Задание	1
3	Теоретическое введение	1
4	Выполнение лабораторной работы	7
5	Выводы	8
	Список литературы.....	8

1 Цель работы

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

2 Задание

Рассмотрим пример простой программы на языке ассемблера NASM. Традиционно первая программа выводит приветственное сообщение Hello world! на экран.

3 Теоретическое введение

**5.2.1 Основные принципы работы компьютера

Основными функциональными элементами любой электронно-вычислительной машины (ЭВМ) являются центральный процессор, память и периферийные устройства (рис. 5.1). Взаимодействие этих устройств осуществляется через общую шину, к которой они подключены. Физически шина представляет собой большое количество про- водников, соединяющих устройства друг с другом. В современных компьютерах проводники выполнены в виде электропроводящих дорожек на материнской (системной) плате.

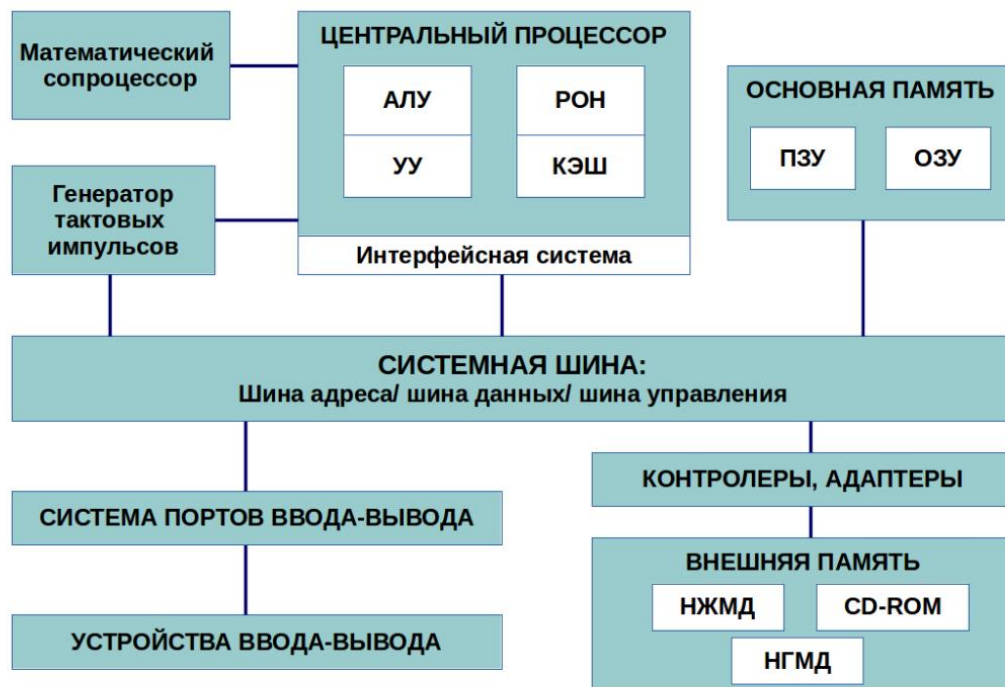


Рис. 5.1. Структурная схема ЭВМ

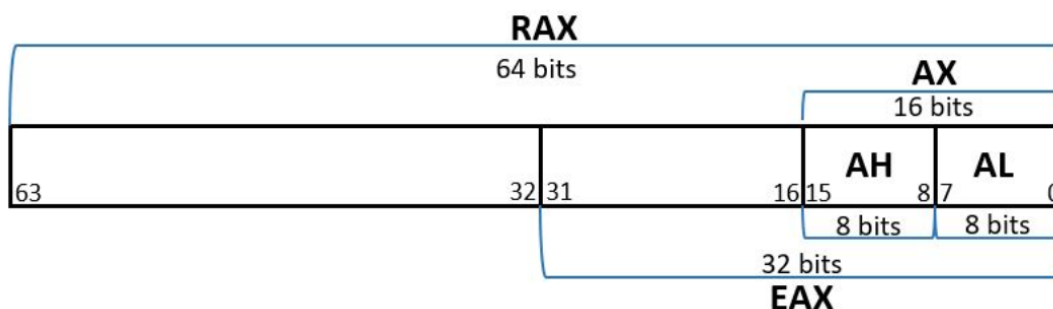
Основной задачей процессора является обработка информации, а также организация координации всех узлов компьютера. В состав **центрального процессора (ЦП)** входят следующие устройства:

- **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** — сверх быстрая 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.



2: Рис. 5.1. 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
```

Рис. 5.2. 64-битный регистр процессора 'RAX'

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 длительного 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.

5.2.2. 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:

- 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.

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.

5.2.3. Процесс создания и обработки программы на языке ассемблера

Процесс создания ассемблерной программы можно изобразить в виде следующей схемы (рис. 4).



Рис. 4: **Рис. 5.3.** Процесс создания ассемблерной программы

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).

4 Выполнение лабораторной работы

Создал каталог для работы с программами на NASM

Перешел в созданный каталог

```
gspolyakov@dk8n60 ~/work/arch-pc/lab05 $ touch hello.asm
gspolyakov@dk8n60 ~/work/arch-pc/lab05 $ gedit hello.asm
gspolyakov@dk8n60 ~/work/arch-pc/lab05 $
```

Рис. 4

Создал текстовый файл с именем hello.asm

Открыл этот файл в текстовом редакторе и вписал текст (рис. 4)

```
gspolyakov@dk8n60 ~/work/arch-pc/lab05 $ nasm -f elf hello.asm
gspolyakov@dk8n60 ~/work/arch-pc/lab05 $ ls
hello.asm  hello.o
gspolyakov@dk8n60 ~/work/arch-pc/lab05 $
```

Рис. 5

Скомпилировал файл в hello.o (рис. 5)

```
gspolyakov@dk8n60 ~/work/arch-pc/lab05 $ nasm -o obj.o -f elf -g -l list.lst hello.asm
gspolyakov@dk8n60 ~/work/arch-pc/lab05 $ ls
hello.asm  hello.o  list.lst  obj.o
gspolyakov@dk8n60 ~/work/arch-pc/lab05 $
```

Рис. 6

Скомпилировал файл в obj.o (рис. 6)

```
gspolyakov@dk8n60 ~/work/arch-pc/lab05 $ ld -m elf_i386 hello.o -o hello
gspolyakov@dk8n60 ~/work/arch-pc/lab05 $ ls
hello  hello.asm  hello.o  list.lst  obj.o
gspolyakov@dk8n60 ~/work/arch-pc/lab05 $
```

Рис. 7

Создал исполняемый файл (рис. 7)

```
gspolyakov@dk8n60 ~/work/arch-pc/lab05 $ ld -m elf_i386 hello.o -o hello
gspolyakov@dk8n60 ~/work/arch-pc/lab05 $ ls
hello  hello.asm  hello.o  list.lst  obj.o
gspolyakov@dk8n60 ~/work/arch-pc/lab05 $
```

Рис. 8

Скомпоновал файл в main (рис. 8)

5 Выводы

Assembler nasm – низкоуровневый ЯП, для работы с которым необходимо вручную компилировать и создавать исполняемые файлы.

Список литературы