

Отчет по лабораторной работе №5

Архитектура компьютера

Рогожина Надежда Александровна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
3.1	Основные принципы работы компьютера	7
3.2	Ассемблер и язык ассемблера	9
3.3	Процесс создания и обработки программы на языке ассемблера .	9
4	Выполнение лабораторной работы	11
5	Выводы	15

Список иллюстраций

4.1	Создадим и перейдем в каталог ~/work/arch-pc/lab05	11
4.2	Создадим текстовый файл с именем hello.asm	12
4.3	Откроем этот файл с помощью текстового редактора	12
4.4	Введем в него следующий текст:	13
4.5	Скомпилируем программу	13
4.6	Проверим компиляцию программы и создание файла с объектным кодом	14

Список таблиц

1 Цель работы

Целью данной работы является изучение языка Assembler, освоение процедуры компиляции и сборки программ, написанных на ассемблере NASM.

2 Задание

1. В каталоге `~/work/arch-рс/lab05` с помощью команды `ср` создайте копию файла `hello.asm` с именем `lab5.asm`
2. С помощью любого текстового редактора внесите изменения в текст программы в файле `lab5.asm` так, чтобы вместо `Hello world!` на экран выводилась строка с вашими фамилией и именем.
3. Оттранслируйте полученный текст программы `lab5.asm` в объектный файл. Выполните компоновку объектного файла и запустите получившийся исполняемый файл.
4. Скопируйте файлы `hello.asm` и `lab5.asm` в Ваш локальный репозиторий в каталог `~/work/study/2022-2023/“Архитектура компьютера”/arch-рс/labs/lab05/`. Загрузите файлы на Github.

3 Теоретическое введение

3.1 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*.

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.

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**.

3.3 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`.

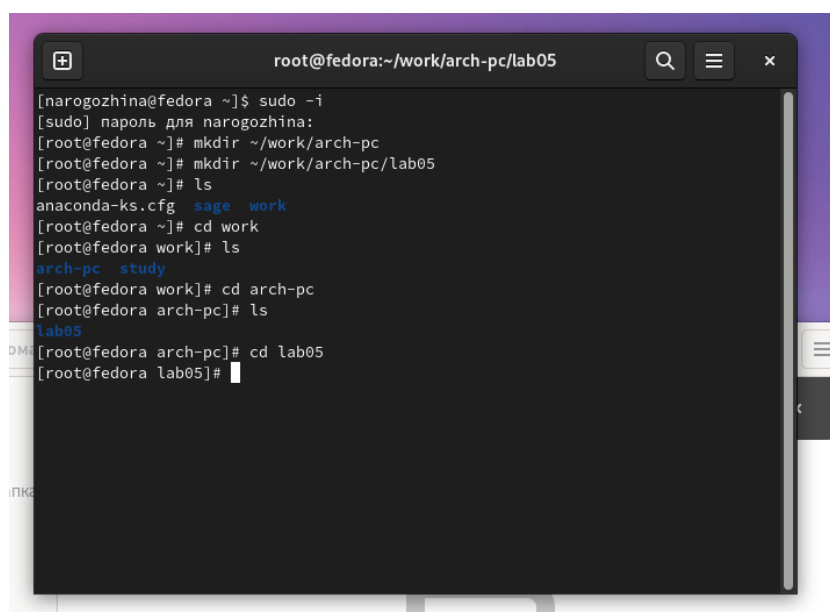
- **Трансляция** — преобразование с помощью транслятора, например `nasm`, текста программы в машинный код, называемый объектным. На данном этапе также может быть получен листинг программы, содержащий кроме текста программы различную дополнительную информацию, созданную транслятором. Тип объектного файла — `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.

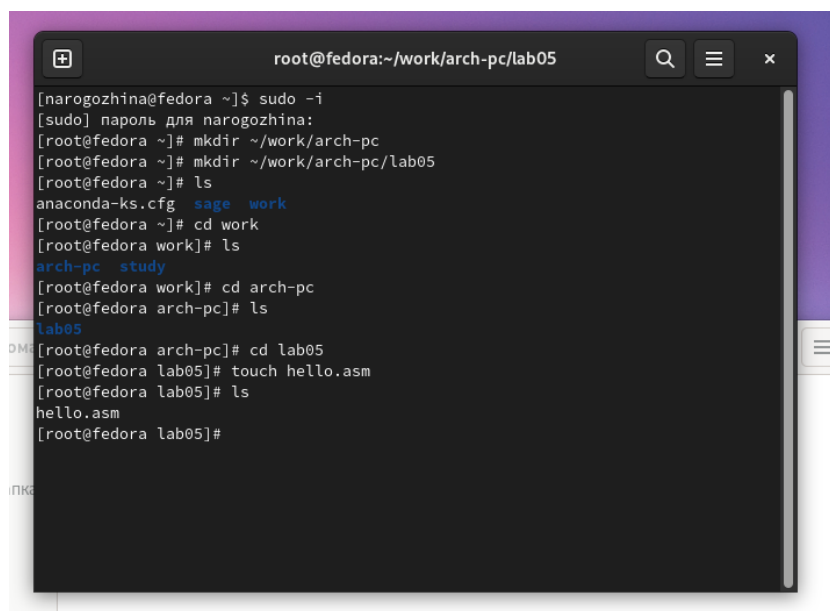
4 Выполнение лабораторной работы

Описываются проведённые действия, в качестве иллюстрации даётся ссылка на иллюстрацию (рис. 4.1)



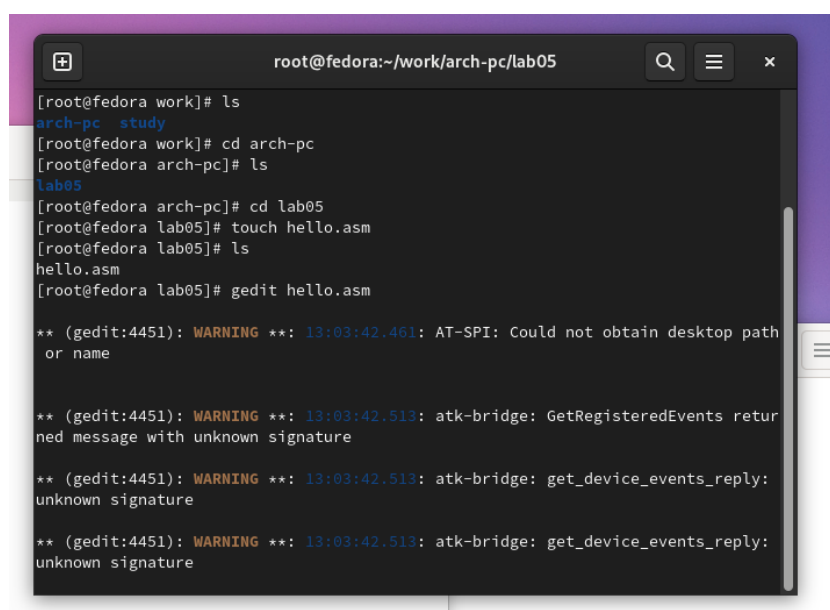
```
root@fedora:~/work/arch-pc/lab05
[narogozhina@fedora ~]$ sudo -i
[sudo] пароль для narogozhina:
[root@fedora ~]# mkdir ~/work/arch-pc
[root@fedora ~]# mkdir ~/work/arch-pc/lab05
[root@fedora ~]# ls
anaconda-ks.cfg  sage  work
[root@fedora ~]# cd work
[root@fedora work]# ls
arch-pc  study
[root@fedora work]# cd arch-pc
[root@fedora arch-pc]# ls
lab05
[root@fedora arch-pc]# cd lab05
[root@fedora lab05]#
```

Рис. 4.1: Создадим и перейдем в каталог ~/work/arch-pc/lab05



```
root@fedora:~/work/arch-pc/lab05
[narogozhina@fedora ~]$ sudo -i
[sudo] пароль для narogozhina:
[root@fedora ~]# mkdir ~/work/arch-pc
[root@fedora ~]# mkdir ~/work/arch-pc/lab05
[root@fedora ~]# ls
anaconda-ks.cfg  sage  work
[root@fedora ~]# cd work
[root@fedora work]# ls
arch-pc  study
[root@fedora work]# cd arch-pc
[root@fedora arch-pc]# ls
lab05
[root@fedora arch-pc]# cd lab05
[root@fedora lab05]# touch hello.asm
[root@fedora lab05]# ls
hello.asm
[root@fedora lab05]#
```

Рис. 4.2: Создадим текстовый файл с именем hello.asm



```
root@fedora:~/work/arch-pc/lab05
[root@fedora work]# ls
arch-pc  study
[root@fedora work]# cd arch-pc
[root@fedora arch-pc]# ls
lab05
[root@fedora arch-pc]# cd lab05
[root@fedora lab05]# touch hello.asm
[root@fedora lab05]# ls
hello.asm
[root@fedora lab05]# gedit hello.asm

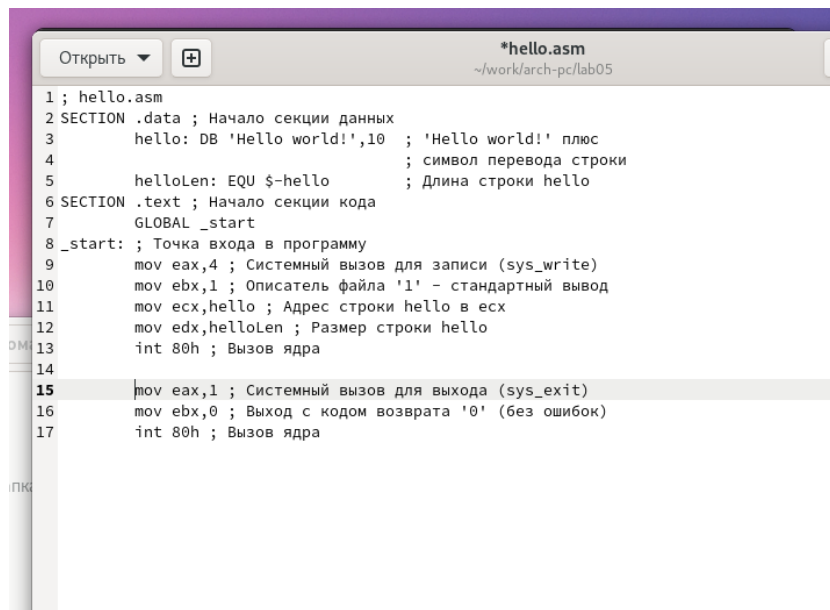
** (gedit:4451): WARNING **: 13:03:42.461: AT-SPI: Could not obtain desktop path
or name

** (gedit:4451): WARNING **: 13:03:42.513: atk-bridge: GetRegisteredEvents retur
ned message with unknown signature

** (gedit:4451): WARNING **: 13:03:42.513: atk-bridge: get_device_events_reply:
unknown signature

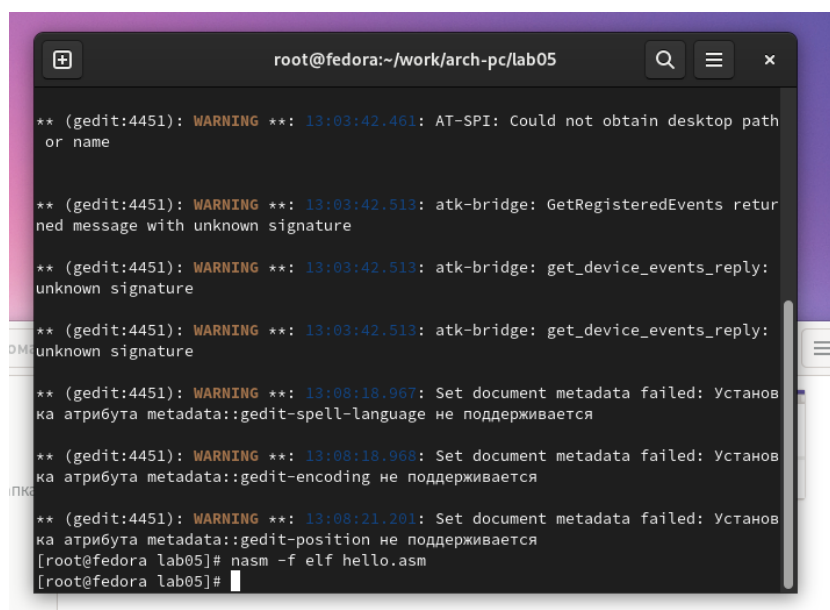
** (gedit:4451): WARNING **: 13:03:42.513: atk-bridge: get_device_events_reply:
unknown signature
```

Рис. 4.3: Откроем этот файл с помощью текстового редактора



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

Рис. 4.4: Введем в него следующий текст:



```
root@fedora:~/work/arch-pc/lab05

** (gedit:4451): WARNING **: 13:03:42.461: AT-SPI: Could not obtain desktop path
or name

** (gedit:4451): WARNING **: 13:03:42.513: atk-bridge: GetRegisteredEvents retur
ned message with unknown signature

** (gedit:4451): WARNING **: 13:03:42.513: atk-bridge: get_device_events_reply:
unknown signature

** (gedit:4451): WARNING **: 13:03:42.513: atk-bridge: get_device_events_reply:
unknown signature

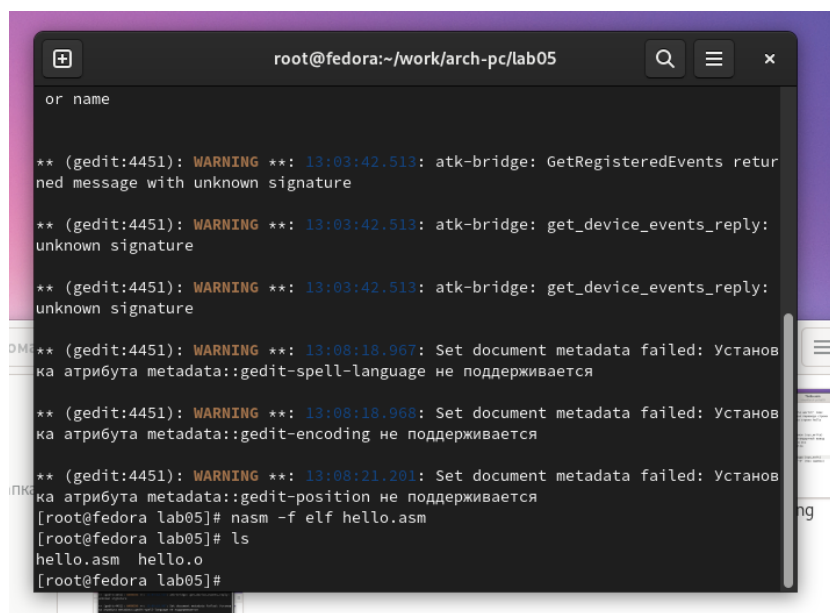
** (gedit:4451): WARNING **: 13:08:18.967: Set document metadata failed: Установ
ка атрибута metadata::gedit-spell-language не поддерживается

** (gedit:4451): WARNING **: 13:08:18.968: Set document metadata failed: Установ
ка атрибута metadata::gedit-encoding не поддерживается

** (gedit:4451): WARNING **: 13:08:21.201: Set document metadata failed: Установ
ка атрибута metadata::gedit-position не поддерживается

[root@fedora lab05]# nasm -f elf hello.asm
[root@fedora lab05]#
```

Рис. 4.5: Скомпилируем программу



A terminal window titled 'root@fedora:~/work/arch-pc/lab05' with search and menu icons. The terminal output shows several warning messages from gedit and atk-bridge, followed by the execution of 'nasm -f elf hello.asm' and 'ls', which results in the creation of 'hello.o'.

```
or name

** (gedit:4451): WARNING **: 13:03:42.513: atk-bridge: GetRegisteredEvents returned message with unknown signature
** (gedit:4451): WARNING **: 13:03:42.513: atk-bridge: get_device_events_reply: unknown signature
** (gedit:4451): WARNING **: 13:03:42.513: atk-bridge: get_device_events_reply: unknown signature

** (gedit:4451): WARNING **: 13:08:18.967: Set document metadata failed: Установка атрибута metadata::gedit-spell-language не поддерживается
** (gedit:4451): WARNING **: 13:08:18.968: Set document metadata failed: Установка атрибута metadata::gedit-encoding не поддерживается
** (gedit:4451): WARNING **: 13:08:21.201: Set document metadata failed: Установка атрибута metadata::gedit-position не поддерживается
[root@fedora lab05]# nasm -f elf hello.asm
[root@fedora lab05]# ls
hello.asm  hello.o
[root@fedora lab05]#
```

Рис. 4.6: Проверим компиляцию программы и создание файла с объектным кодом

5 Выводы

Таким образом, мы изучение языка Assembler, освоение процедуры компиляции и сборки программ, написанных на ассемблере NASM.