

Отчёт по лабораторной работе №5

Дисциплина: Архитектура компьютера

Канева Екатерина Павловна

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	8
3.3	Процесс создания и обработки программы на языке ассемблера .	9
4	Выполнение лабораторной работы	11
4.1	Лабораторная работа	11
4.2	Самостоятельная работа	15
5	Выводы	18

Список иллюстраций

4.1	Создание каталога programs.	11
4.2	Создание и открытие файла hello.asm.	11
4.3	Создание программы.	13
4.4	Компиляция программы и проверка создания объектного файла.	13
4.5	Компиляция файла с присвоением другого имени, создание файла листинга.	13
4.6	Передача объектного файла hello.o компоновщику.	14
4.7	Передача объектного файла obj.o компоновщику.	14
4.8	Выполнение исполняемого файла hello.	14
4.9	Копирование файла hello.asm как lab5.asm.	15
4.10	Открытие файла lab5.asm.	15
4.11	Изменённый текст программы.	16
4.12	Создание объектного и исполняемого файлов, запуск программы.	17
4.13	Копирование файлов, удаление ненужных файлов и каталогов.	17

Список таблиц

1 Цель работы

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

2 Задание

1. В каталоге `~/work/study/2022-2023/"Архитектура компьютера"/arch-
pc/labs/lab05/programs` с помощью команды `cp` 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-
pc/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 — **ассемблером**.

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 эта-

пе 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.

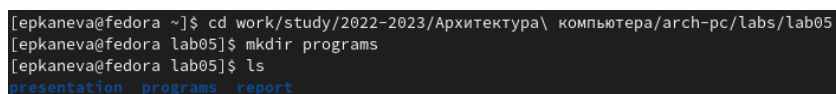
- **Запуск программы.** Конечной целью является работоспособный исполняемый файл. Ошибки на предыдущих этапах могут привести к некорректной работе программы, поэтому может присутствовать этап отладки программы при помощи специальной программы — отладчика. При нахождении ошибки необходимо провести коррекцию программы, начиная с первого шага.

4 Выполнение лабораторной работы

4.1 Лабораторная работа

Сначала создадим папку, в которой будут созданы программы на языке ассемблера NASM. Для этого введём следующие команды (рис. 4.1):

```
cd work/study/2022-2023/"Архитектура компьютера"/arch-pc/labs/lab05
mkdir programs
ls
```

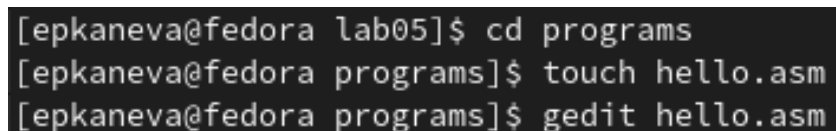


```
[epkaneva@fedora ~]$ cd work/study/2022-2023/Архитектура\ компьютера/arch-pc/labs/lab05
[epkaneva@fedora lab05]$ mkdir programs
[epkaneva@fedora lab05]$ ls
presentation  programs  report
```

Рис. 4.1: Создание каталога programs.

Затем перейдём в созданный каталог, создадим файл `hello.asm` и откроем его в `gedit` (рис. 4.2):

```
cd programs
touch hello.asm
gedit hello.asm
```



```
[epkaneva@fedora lab05]$ cd programs
[epkaneva@fedora programs]$ touch hello.asm
[epkaneva@fedora programs]$ gedit hello.asm
```

Рис. 4.2: Создание и открытие файла `hello.asm`.

В созданном файле введём нужный текст (рис. 4.3):

```
; hello.asm
```

```
SECTION .data                ; Начало секции данных
    hello:    DB 'Hello, world!',10    ; 'Hello world!' плюс
                                           ; символ перевода строки
    helloLen: EQU $-hello            ; Длина строки hello
```

```
SECTION .text                ; Начало секции кода
    GLOBAL _start
```

```
_start:                    ; Точка входа в программу

    mov eax,4              ; Системный вызов для записи (sys_write)
    mov ebx,1              ; Описатель файла '1' - стандартный вывод
    mov ecx,hello          ; Адрес строки hello в ecx
    mov edx,helloLen       ; Размер строки hello
    int 0x80               ; Вызов ядра

    mov eax,1              ; Системный вызов для выхода (sys_exit)
    mov ebx,0              ; Выход с кодом возврата '0' (без ошибок)
    int 0x80               ; Вызов ядра
```

```

1 ; hello.asm
2
3 SECTION .data                ; Начало секции данных
4     hello:    DB 'Hello, world!',10 ; 'Hello world!' плюс
5                                     ; символ перевода строки
6     helloLen: EQU $-hello        ; Длина строки hello
7
8 SECTION .text                ; Начало секции кода
9     GLOBAL _start
10
11 _start:                    ; Точка входа в программу
12     mov eax,4              ; Системный вызов для записи (sys_write)
13     mov ebx,1              ; Описатель файла '1' - стандартный вывод
14     mov ecx,hello          ; Адрес строки hello в ecx
15     mov edx,helloLen       ; Размер строки hello
16     int 0x80              ; Вызов ядра
17
18     mov eax,1              ; Системный вызов для выхода (sys_exit)
19     mov ebx,0              ; Выход с кодом возврата '0' (без ошибок)
20     int 0x80              ; Вызов ядра

```

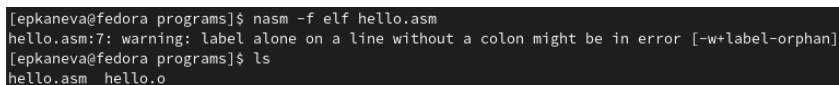
Рис. 4.3: Создание программы.

Для компиляции текста программы введём следующую команду (рис. 4.4):

```

nasm -f elf hello.asm
ls

```



```

[epkaneva@fedora programs]$ nasm -f elf hello.asm
hello.asm:7: warning: label alone on a line without a colon might be in error [-w+label-orphan]
[epkaneva@fedora programs]$ ls
hello.asm  hello.o

```

Рис. 4.4: Компиляция программы и проверка создания объектного файла.

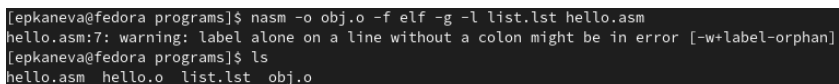
Видим, что объектный файл имеет имя `hello.o`.

Теперь скомпилируем файл с присвоением другого имени, а также создадим файл листинга; проверим создание файлов (рис. 4.5):

```

nasm -o obj.o -f elf -g -l list.lst hello.asm
ls

```



```

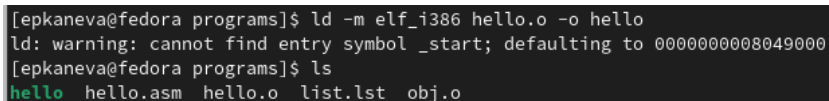
[epkaneva@fedora programs]$ nasm -o obj.o -f elf -g -l list.lst hello.asm
hello.asm:7: warning: label alone on a line without a colon might be in error [-w+label-orphan]
[epkaneva@fedora programs]$ ls
hello.asm  hello.o  list.lst  obj.o

```

Рис. 4.5: Компиляция файла с присвоением другого имени, создание файла листинга.

Далее передадим объектный файл `hello.o` на обработку компоновщику и проверим создание исполняемого файла (рис. 4.6):

```
ld -m elf_i386 hello.o -o hello
ls
```



```
[epkaneva@fedora programs]$ ld -m elf_i386 hello.o -o hello
ld: warning: cannot find entry symbol _start; defaulting to 0000000008049000
[epkaneva@fedora programs]$ ls
hello  hello.asm  hello.o  list.lst  obj.o
```

Рис. 4.6: Передача объектного файла `hello.o` компоновщику.

Теперь передадим компоновщику на обработку объектный файл `obj.o` и проверим создание исполняемого файла (рис. 4.7):

```
ld -m elf_i386 obj.o -o main
ls
```



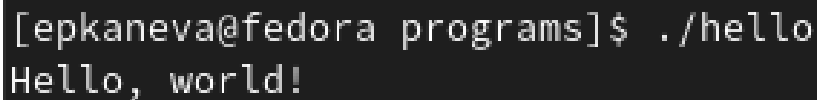
```
[epkaneva@fedora programs]$ ld -m elf_i386 obj.o -o main
ld: warning: cannot find entry symbol _start; defaulting to 0000000008049000
[epkaneva@fedora programs]$ ls
hello  hello.asm  hello.o  list.lst  main  obj.o
```

Рис. 4.7: Передача объектного файла `obj.o` компоновщику.

Исполняемый файл имеет имя `main`, объектный файл `obj.o`.

Теперь запустим на выполнение исполняемый файл `hello` (рис. 4.8):

```
./hello
```



```
[epkaneva@fedora programs]$ ./hello
Hello, world!
```

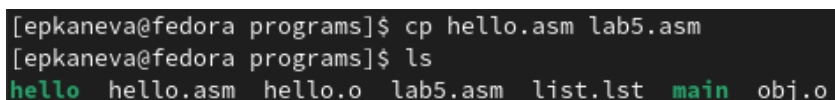
Рис. 4.8: Выполнение исполняемого файла `hello`.

4.2 Самостоятельная работа

Теперь всё то же самое сделаем для того, чтобы на экран были выведены фамилия и имя. Скопируем файл `hello.asm` под именем `lab5.asm`, проверим выполнение действия (рис. 4.9):

```
cp hello.asm lab5.asm
```

```
ls
```

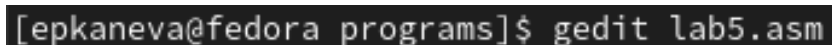


```
[epkaneva@fedora programs]$ cp hello.asm lab5.asm
[epkaneva@fedora programs]$ ls
hello  hello.asm  hello.o  lab5.asm  list.lst  main  obj.o
```

Рис. 4.9: Копирование файла `hello.asm` как `lab5.asm`.

Откроем файл `lab5.asm` в `gedit` (рис. 4.10) и изменим текст программы так, чтобы при исполнении файла на экран выводились фамилия и имя (рис. 4.11):

```
gedit lab5.asm
```



```
[epkaneva@fedora programs]$ gedit lab5.asm
```

Рис. 4.10: Открытие файла `lab5.asm`.

```
; lab5.asm
```

```
SECTION .data                                ; Начало секции данных
    lab5:      DB 'Ekaterina Kaneva',10      ; 'Ekaterina Kaneva' плюс
                                                ; символ перевода строки
    lab5Len:    EQU $-lab5                    ; Длина строки lab5

SECTION .text                                ; Начало секции кода
    GLOBAL _start
```

```

_start:          ; Точка входа в программу

    mov eax,4      ; Системный вызов для записи (sys_write)
    mov ebx,1      ; Описатель файла '1' - стандартный вывод
    mov ecx,lab5   ; Адрес строки lab5 в ecx
    mov edx,lab5Len ; Размер строки lab5
    int 0x80       ; Вызов ядра

    mov eax,1      ; Системный вызов для выхода (sys_exit)
    mov ebx,0      ; Выход с кодом возврата '0' (без ошибок)
    int 0x80       ; Вызов ядра

```

```

1 ; lab5.asm
2
3 SECTION .data          ; Начало секции данных
4     lab5:              DB 'Ekaterina Kaneva',10 ; 'Ekaterina Kaneva' плюс
5                                     ; символ перевода строки
6     lab5Len:           EQU $-lab5           ; Длина строки lab5
7
8 SECTION .text          ; Начало секции кода
9     GLOBAL _start
10
11 _start:                ; Точка входа в программу
12     mov eax,4          ; Системный вызов для записи (sys_write)
13     mov ebx,1          ; Описатель файла '1' - стандартный вывод
14     mov ecx,lab5       ; Адрес строки lab5 в ecx
15     mov edx,lab5Len    ; Размер строки lab5
16     int 0x80           ; Вызов ядра
17
18     mov eax,1          ; Системный вызов для выхода (sys_exit)
19     mov ebx,0          ; Выход с кодом возврата '0' (без ошибок)
20     int 0x80           ; Вызов ядра

```

Рис. 4.11: Изменённый текст программы.

Используя те же команды, что и ранее, создадим объектный файл, исполняемый файл и запустим программу в терминале (рис. 4.12):

```

nasm -f elf lab5.asm
ld -m elf_i386 lab5.o -o lab5
./lab5

```



```
[epkaneva@fedora programs]$ nasm -f elf lab5.asm
lab5.asm:9: warning: label alone on a line without a colon might be in error [-w+label-orphan]
[epkaneva@fedora programs]$ ld -m elf_i386 lab5.o -o lab5
ld: warning: cannot find entry symbol _start; defaulting to 0000000008049000
[epkaneva@fedora programs]$ ./lab5
Ekaterina Kaneva
```

Рис. 4.12: Создание объектного и исполняемого файлов, запуск программы.

Файлы lab5.asm и hello.asm перенесём в каталог ~/work/study/2022-2023/"Архитектура компьютера"/arch-pc/labs/lab05/, ненужные файлы и каталог удалим, проверим корректность выполнения команд (рис. 4.13), загрузим всё на GitHub.

```
cd ..
cp programs/hello.asm hello.asm
cp programs/lab5.asm lab5.asm
ls
rm -r programs
ls
```

```
[epkaneva@fedora programs]$ cd ..
[epkaneva@fedora lab05]$ cp programs/hello.asm hello.asm
[epkaneva@fedora lab05]$ cp programs/lab5.asm lab5.asm
[epkaneva@fedora lab05]$ ls
hello.asm lab5.asm presentation programs report
[epkaneva@fedora lab05]$ rm -r programs
[epkaneva@fedora lab05]$ ls
hello.asm lab5.asm presentation report
```

Рис. 4.13: Копирование файлов, удаление ненужных файлов и каталогов

5 Выводы

Освоили процедуру компиляции и сборки программ, написанных на ассемблере NASM.