Отчет по лабораторной работе №4

Архитектура компьютеров

Бердыев Даянч НММбд-04-23

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

2.0.1 1

Создаю каталог для работы с программами на языке ассемблера NASM.

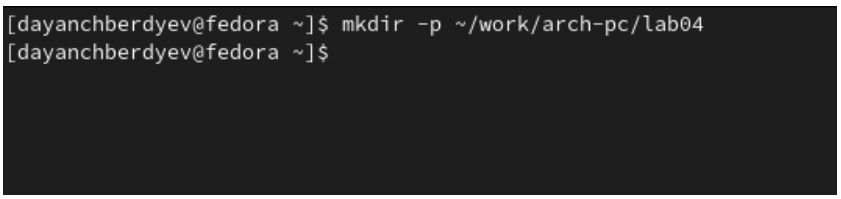

Рис. 2.1: Создание каталога с помощью команд mkdir -p ~/work/arch-pc/lab04

2.0.2 2

Перехожу в созданный каталог.

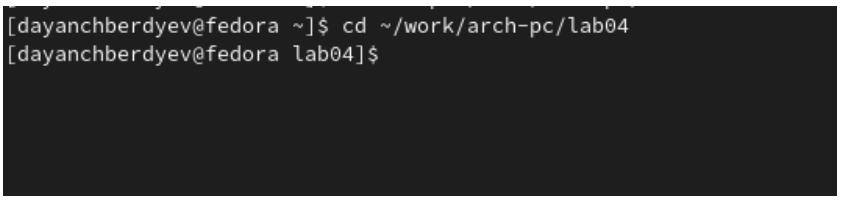

Рис. 2.2: Переход в созданный каталог с помощью команд cd ~/work/arch-pc/lab04

2.0.3 3

Создаю текстовый файл с именем hello.asm

Рис. 2.3: Создание текстового файла с помощью команд touch hello.asm

2.0.4 4

Открываю этот файл с помощью текстового редактора gedit.

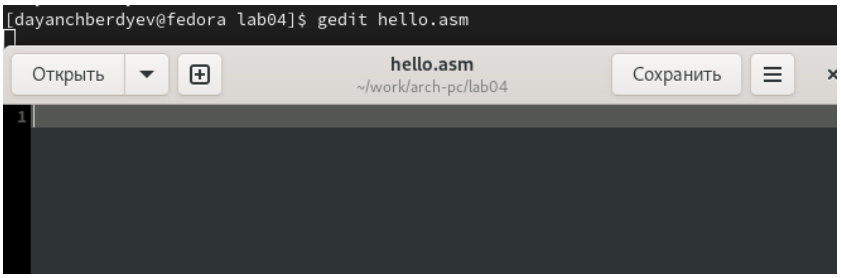

Рис. 2.4: Открытие текстового редактора gedit с помощью команды gedit hello.asm

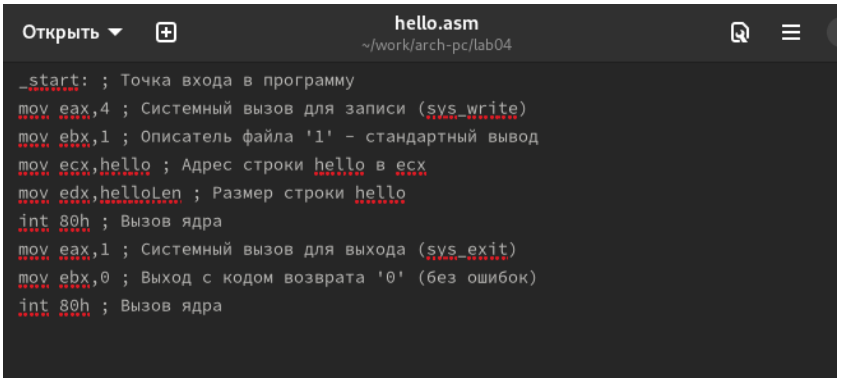

Рис. 2.5: И ввожу в него следующий текст.

2.0.5 5

NASM превращает текст программы в объектный код.

Рис. 2.6: Ввожу команду nasm -f elf hello.asm

2.0.6 6

Полный вариант командной строки nasm выглядит следующим образом:

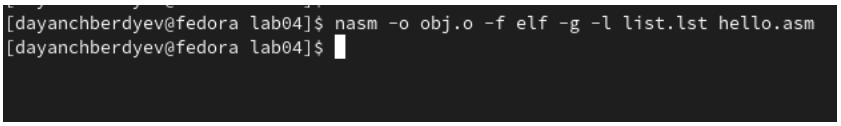

Рис. 2.7: Расширенный синтаксис командной строки NASM.

2.0.7 7

Чтобы получить исполняемую программу, объектный файл необходимо передать на обработку компоновщику:

Рис. 2.8: Компоновщик LD.

2.0.8 8

Ключ -o с последующим значением задаёт в данном случае имя создаваемого исполняемого файла.

Рис. 2.9: Ввожу команду ld -m elf_i386 obj.o -o main

2.0.9 9

Запуск исполняемого файла.

Рис. 2.10: Ввожу команду ./hello

3 Самостоятельная работа

3.0.1 1

В каталоге ~/work/arch-pc/lab04 с помощью команды cp

Рис. 3.1: Создаю копию файла hello.asm с именем lab04.asm

3.0.2 2

С помощью текстового редактора gedit ввожу изменения в тексте программы в файле lab04.asm вместо Hello world! ввожу Бердыев Даянч.

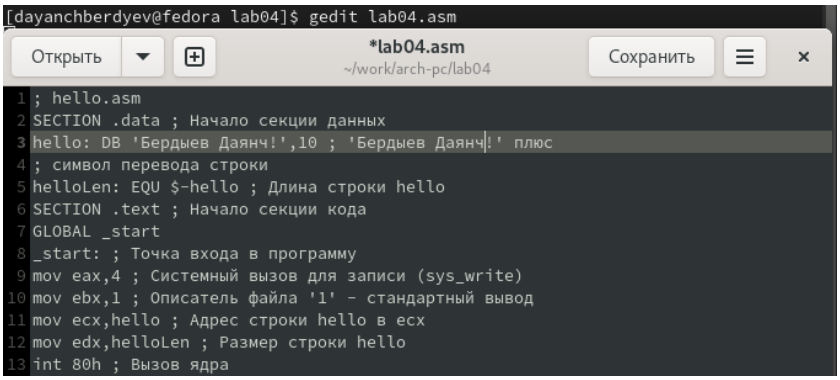

Рис. 3.2: Ввожу свое имя фамилию.

3.0.3 3

Оттранслирую полученный текст программы lab04.asm в объектный файл. Выполняю компоновку объектного файла.

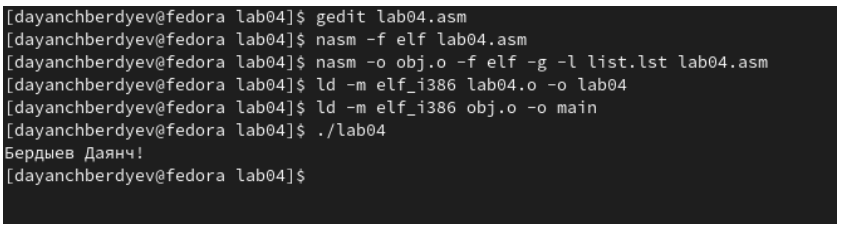

Рис. 3.3: Запускаю получившийся исполняемый файл.

3.0.4 4

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

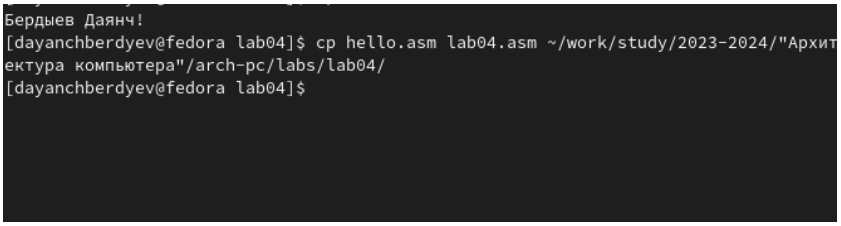

Рис. 3.4: Копирую файлы hello.asm и lab4.asm с помошью команды cp hello.asm lab04.asm ~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/

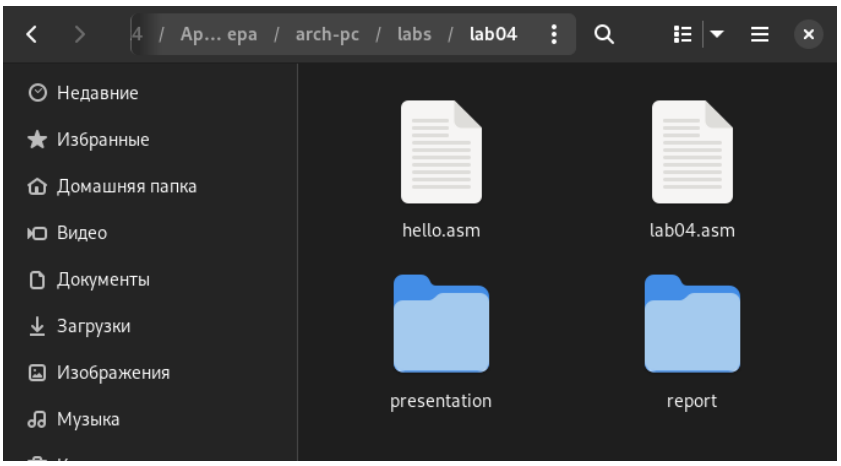

Рис. 3.5: Проверяю.

```
[dayanchberdyev@fedora lab04]$ cd ~/work/study/2023-2024/"Архитектура компьютера"/a
rch-pc
[dayanchberdyev@fedora arch-pc]$ git add .
[dayanchberdyev@fedora arch-pc]$ git commit -am 'feat(main): add files lab-4'
[master 9cac902] feat(main): add files lab-4
 2 files changed, 32 insertions(+)
 create mode 100644 labs/lab04/hello.asm
 create mode 100644 labs/lab04/lab04.asm
[dayanchberdyev@fedora arch-pc]$ git push
Перечисление объектов: 9, готово.
Подсчет объектов: 100% (9/9), готово.
При сжатии изменений используется до 2 потоков
Сжатие объектов: 100% (6/6), готово.
Запись объектов: 100% (6/6), 970 байтов | 485.00 КиБ/с, готово.
Всего 6 (изменений 3), повторно использовано 0 (изменений 0), повторно использовано
 пакетов 0
remote: Resolving deltas: 100% (3/3), completed with 2 local objects.
To github.com:Denchi05/den05.git
   3fb646a..9cac902  master -> master
[dayanchberdyev@fedora arch-pc]$
```

Рис. 3.6: Загржаю файлы на Github.

4 Ответы на вопросы

1. Основное отличие ассемблера от языков высокого уровня — Байт-код или байткóд (англ. byte-code), иногда также используется термин псевдокóд — машинно-независимый код низкого уровня, генерируемый транслятором и исполняемый интерпретатором. Большинство инструкций байт-кода эквивалентны одной или нескольким командам ассемблера. Трансляция в байт-код занимает промежуточное положение между компиляцией в машинный код и интерпретацией.

2. Инструкция ассемблера генерирует машинный код, таким образом, способствует размеру программы. Директива ассемблера не создает какого -либо машинного кода, таким образом, не способствует размеру программы. IT приказывает ассемблеру выполнять определенные действия на этапе сборки.

3. Правила написания программ на языке assembler Исходный текст программы на языке ассемблера имеет определенный формат. Каждая команда и директива представляет собой строку: Метка, операция,операнд(ы), комментарии.

4. Создание исполняемого файла издавна производилось в три этапа: (1) обработка исходного кода препроцессором, (2) компиляция в объектный код и (3) компоновка объектных модулей, включая модули из объектных библиотек, в исполняемый файл. Это классическая схема для компилируемых языков.

5. На этапе трансляции осуществляется перевод команд ассемблера в соответствующие машинные команды. В результате трансляции формируются файл объектного модуля и файл листинга.

6. Если в процессе ассемблирования не было выявлено ошибок в ассемблерном листинге, то программа-ассемблер создаст объектный файл (с расширением OBJ).

Затем необходимо воспользоваться компоновщиком (линковщиком), который входит в комплект программы-ассемблера. Данная процедура выполняется гораздо быстрее ассемблирования.

Именно компоновщик создает готовый к запуску файл (программу) с расширением COM или EXE из объектного файла (OBJ). Оба типа имеют отличия в структуре ассемблерной программы. Первый тип (COM) не может превышать 64 Кбайт и используется только в MS-DOS (и для совместимости поддерживается в Windows), однако он очень компактный и удобный для написания небольших программ и резидентов. В большинстве случаев, если программа написана на чистом ассемблере под MS-DOS, нет необходимости создавать EXE-файлы. В этой книге в части I рассматриваются именно программы типа COM.

В отличие от создания программ типа COM, при создании стандартных EXE-программ под MS-DOS нет необходимости указывать какие-либо параметры линковщику при компоновке. Дело в том, что компоновщик не может автоматически определить, какой тип подвергается компоновке.

Линковщик также проверяет, нет ли каких-либо ошибок в объектном файле, но не грамматических, а логических. Например, отсутствие необходимой объектной библиотеки, указанной в самом файле либо в командной строке (программаассемблер этого не делает).

Если ошибки не были обнаружены, компоновщик создает машинный код (программу типа COM или EXE), которую можно запускать на выполнение.

7. Для того чтобы выполнить пробный прогон ассемблерной программы, ее

необходимо сначала оттранслировать и скомпоновать. Пусть текст исходной программы хранится в файле с именем SIMPLE.ASM. Трансляцию можно осуществить вызовом турбо ассемблера TASM.EXE с помощью, например, следующей команды DOS:

```
tasm /l/z/zi/n simple.asm
```

8. NASM поддерживает множество форматов выходных файлов, среди них:

bin — файл произвольного формата, определяемого только исходным кодом. Пригоден как для файлов данных, так и для модулей с исполняемыми кодами — например, системных загрузчиков, образов ПЗУ, модулей операционных систем, драйверов .SYS в MS-DOS или исполняемых файлов .COM. obj — объектный модуль в формате OMF, совместимый с MASM и TASM. win32 и win64 — объектный модуль для 32- и 64-битного кода, совместимый с Win32- и Win64-компиляторами Microsoft. aout — объектный модуль в варианте формата a.out, использовавшегося в ранних Linux-системах. aoutb — версия формата a.out для BSD-совместимых операционных систем. coff — объектный модуль в формате COFF, совместимом с компоновщиком из DJGPP. elf32 и elf64 — объектный модуль в форматах ELF32 и ELF64, используемых в Linux и Unix System V, включая Solaris x86, UnixWare и SCO Unix. Формат выходного файла можно задать с помощью ключа командной строки -f. Форматы могут расширять синтаксис некоторых инструкций и добавлять собственные инструкции.

5 Выводы

В ходе выполнения этой лабораторной работы я освоил процедуру компиляции и сборки программ, написанных на ассемблере NASM.