

Отчёт по лабораторной работе №4

Дисциплина: Архитектура Компьютера

Дарина Андреевна Куокконен

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	8
5	Выводы	13
	Список литературы	14

Список иллюстраций

4.1	Создание каталога	8
4.2	Создание файла	8
4.3	открытие файла	8
4.4	Ввод текста	9
4.5	Компиляция текста программы	9
4.6	Компиляция текста программы	9
4.7	Передача объектного файла на обработку компановщику	10
4.8	Передача объектного файла на обработку компановщику	10
4.9	Запуск исполняемого файла	10
4.10	Создание копии файла	10
4.11	Создание копии файла	11
4.12	Компиляция текста программы	11
4.13	Передача файла на обработку	11
4.14	Запуск файла	12
4.15	Добавление файлов	12
4.16	Отправка файлов	12

1 Цель работы

Целью данной лабораторной работы - приобретение практического опыта работы с программами, написанными на ассемблере NASM, а именно - освоение процедур компиляций и сборки.

2 Задание

1. Создание программы Hello World!
2. Работа с транслятором NASM
3. Работа с расширенным синтаксисом командной строки NASM
4. Работа с компоновщиком LD
5. Запуск исполняемого файла
6. Выполнение заданий для самостоятельной работы

3 Теоретическое введение

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. Но 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. Программы, написанные на языке ассемблера, не уступают в качестве и скорости программам, написанным на машинном языке, ибо 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. Для записи команд в NASM используются: 1) Мнемокод — непосредственно мнемоника инструкции процессору, которая является обязательной частью команды. 2) Операнды — числа, данные, адреса регистров или адреса оперативной памяти. 3) Метка — идентификатор, с которым ассемблер ассоциирует некоторое число, чаще всего адрес в памяти. (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 составляет 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.

4 Выполнение лабораторной работы

4.1) Программа Hello World! Создаю каталог для работы с программами на языке ассемблера NASM. И перехожу в него для дальнейшей работы (рис. 4.1).

```
[dakuokkonen@fedora work]$ mkdir -p ~/work/arch-pc/lab04  
[dakuokkonen@fedora work]$ cd arch-pc/lab04  
[dakuokkonen@fedora lab04]$
```

Рис. 4.1: Создание каталога

Создаю текстовый файл hello.asm. (рис. 4.2).

```
[dakuokkonen@fedora lab04]$ touch hello.asm  
[dakuokkonen@fedora lab04]$
```

Рис. 4.2: Создание файла

Открываю созданный файл. (рис. 4.3).

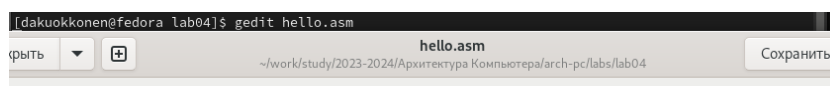


Рис. 4.3: открытие файла

Ввожу нужный текст. (рис. 4.4).

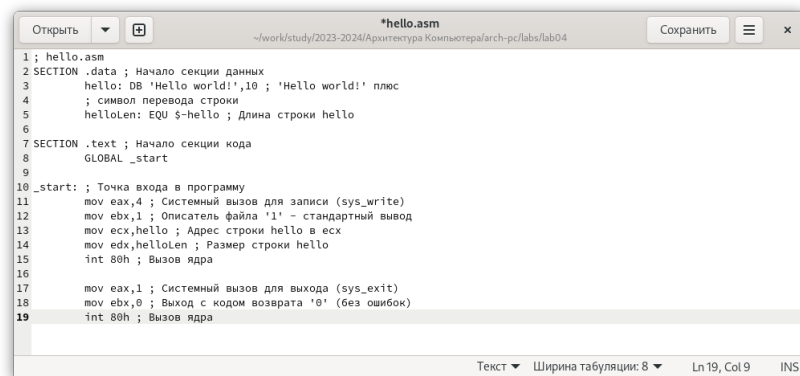


Рис. 4.4: Ввод текста

4.2) Работа с транслятором NASM

NASM превращает текст программы в объектный код. Выполним компиляцию приведённого выше текста программы “Hello World”. Сделаем проверку. (рис. 4.5).

```

[dakuokkonen@fedora lab04]$ nasm -f elf hello.asm
[dakuokkonen@fedora lab04]$ ls
hello.asm hello.o presentation report
[dakuokkonen@fedora lab04]$

```

Рис. 4.5: Компиляция текста программы

4.3) Работа с расширенным синтаксисом командной строки NASM

Далее, ввожу команду, которая скомпилирует файл hello.asm в файл obj.o, при этом в файл будут включены символы для отладки (ключ -g), также с помощью ключа -l будет создан файл листинга list.lst. Проверяю правильность выполнения команды. (рис. 4.6).

```

[dakuokkonen@fedora lab04]$ nasm -o obj.o -f elf -g -l list.lst hello.asm
[dakuokkonen@fedora lab04]$ ls
hello.asm hello.o list.lst obj.o presentation report
[dakuokkonen@fedora lab04]$

```

Рис. 4.6: Компиляция текста программы

4.4) Работа с компоновщиком LD

Передаю объектный файл hello.o на обработку компоновщику LD, чтобы получить исполняемый файл hello. Ключ -o задает имя создаваемого исполняемого файла. С помощью команды “ls” проверяю правильность выполнения.(рис. 4.7).

```
[dakuokkonen@fedora lab04]$ ld -m elf_i386 hello.o -o hello
[dakuokkonen@fedora lab04]$ ls
hello hello.asm hello.o list.lst obj.o presentation report
[dakuokkonen@fedora lab04]$
```

Рис. 4.7: Передача объектного файла на обработку компоновщику

Выполняю ту же самую команду со значением main. Объектный файл, из которого собран этот исполняемый файл, имеет имя obj.o (рис. 4.8)

```
[dakuokkonen@fedora lab04]$ ld -m elf_i386 obj.o -o main
[dakuokkonen@fedora lab04]$ ls
hello hello.asm hello.o list.lst main obj.o presentation report
[dakuokkonen@fedora lab04]$
```

Рис. 4.8: Передача объектного файла на обработку компоновщику

4.5) Запуск исполняемого файла

Запускаю на выполнение созданный исполняемый файл hello. (рис. 4.9).

```
[dakuokkonen@fedora lab04]$ ./hello
Hello world!
[dakuokkonen@fedora lab04]$
```

Рис. 4.9: Запуск исполняемого файла

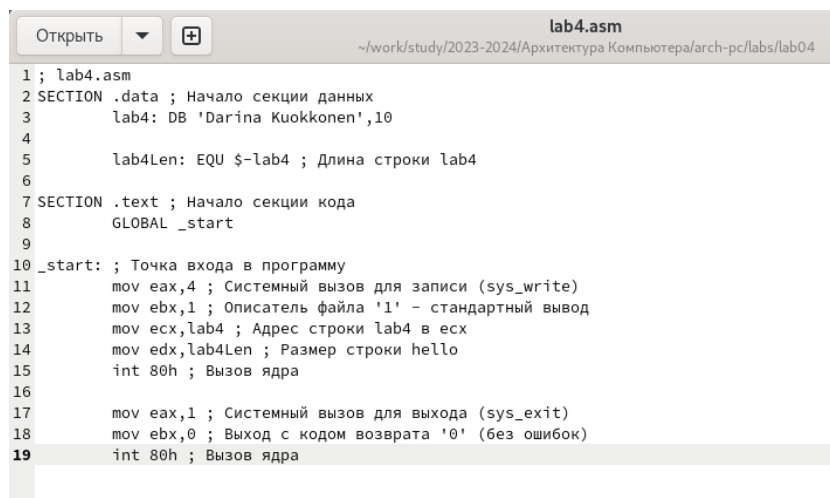
4.6) Выполнение заданий для самостоятельной работы

С помощью утилиты “cp” создаю в текущем каталоге копию файла hello.asm с именем lab4.asm.(рис. 4.10).

```
[dakuokkonen@fedora lab04]$ cp hello.asm lab4.asm
```

Рис. 4.10: Создание копии файла

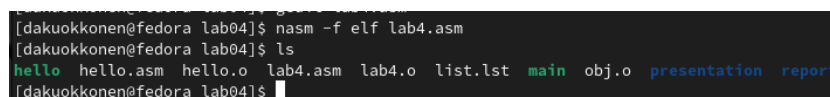
С помощью текстового редактора открываю файл lab4.asm и вношу изменения в программу так, чтобы она выводила мои имя и фамилию. (рис. 4.11)



```
1; lab4.asm
2SECTION .data ; Начало секции данных
3    lab4: DB 'Darina Kuokkonen',10
4
5    lab4Len: EQU $-lab4 ; Длина строки lab4
6
7SECTION .text ; Начало секции кода
8    GLOBAL _start
9
10_start: ; Точка входа в программу
11    mov eax,4 ; Системный вызов для записи (sys_write)
12    mov ebx,1 ; Описатель файла '1' - стандартный вывод
13    mov ecx,lab4 ; Адрес строки lab4 в ecx
14    mov edx,lab4Len ; Размер строки hello
15    int 80h ; Вызов ядра
16
17    mov eax,1 ; Системный вызов для выхода (sys_exit)
18    mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
19    int 80h ; Вызов ядра
```

Рис. 4.11: Создание копии файла

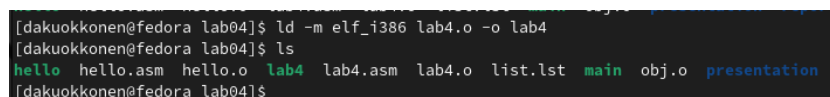
Компилирую текст программы в объектный файл, проверяю что файл создан.(рис. 4.12)



```
[dakuokkonen@fedora lab04]$ nasm -f elf lab4.asm
[dakuokkonen@fedora lab04]$ ls
hello hello.asm hello.o lab4.asm lab4.o list.lst main obj.o presentation report
[dakuokkonen@fedora lab04]$
```

Рис. 4.12: Компиляция текста программы

Передаю файл lab4.asm, на обработку компоновщику LD, чтобы получить исполняемый файл.(рис. 4.13)



```
[dakuokkonen@fedora lab04]$ ld -m elf_i386 lab4.o -o lab4
[dakuokkonen@fedora lab04]$ ls
hello hello.asm hello.o lab4 lab4.asm lab4.o list.lst main obj.o presentation report
[dakuokkonen@fedora lab04]$
```

Рис. 4.13: Передача файла на обработку

Затем, запускаю исполняемый файл lab4.asm. Убеждаюсь, что на экран действительно выводятся мои имя и фамилия.(рис. 4.14)

```
[dakuokkonen@fedora lab04]$ ./lab4
Darina Kuokkonen
[dakuokkonen@fedora lab04]$
```

Рис. 4.14: Запуск файла

Добавляю файлы в GitHub, используя команду “git add.” и “git commit”, комментируя свое действие как добавление файлов для лабораторной работы №4(рис. 4.15)

```
[dakuokkonen@fedora lab04]$ git add .
[dakuokkonen@fedora lab04]$ git commit -m "Add files for lab04"
[master d3e048d] Add files for lab04
 2 files changed, 38 insertions(+)
```

Рис. 4.15: Добавление файлов

Отправляю файлы на сервер с помощью команды “git push”.(рис. 4.16)

```
[dakuokkonen@fedora lab04]$ git push
Перечисление объектов: 12, готово.
Подсчет объектов: 100% (12/12), готово.
Сжатие объектов: 100% (11/11), готово.
Запись объектов: 100% (11/11), 3.76 КиБ | 295.00 КиБ/с, готово.
Всего 11 (изменений 5), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (5/5), completed with 1 local object.
To github.com:dakuokkonen1/study_2023-2024_arh-pc.git
 7766152..25802e6 master -> master
[dakuokkonen@fedora lab04]$
```

Рис. 4.16: Отправка файлов

5 Выводы

При выполнении данной лабораторной работы я приобрела практический опыт работы с программами, написанными на ассемблере NASM, конкретнее - освоила процедуры компиляций и сборки.

Список литературы

Архитектура ЭВМ