

Отчёт по лабораторной работе №4

Дисциплина: Архитектура компьютера

Зарицкая Марина Петровна

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	9
4.1	Создание программы Hello world!	9
4.2	Работа с транслятором NASM	10
4.3	Работа с расширенным синтаксисом командной строки NASM . .	11
4.4	Работа с компоновщиком LD	11
4.5	Запуск исполняемого файла	12
4.6	Выполнение заданий для самостоятельной работы.	12
5	Выводы	15

Список иллюстраций

4.1	Перемещение между директориями	9
4.2	Создание пустого текстового файла	9
4.3	Открытие файла в текстовом редакторе	9
4.4	Заполнение файла	10
4.5	Компиляция текста программы	10
4.6	Компиляция текста программы	11
4.7	Передача объектного файла на обработку компоновщику	11
4.8	Передача объектного файла на обработку компоновщику	11
4.9	Запуск исполняемого файла	12
4.10	Создание копии файла	12
4.11	Изменение программы	13
4.12	Компиляция текста программы	13
4.13	Передача объектного файла на обработку компоновщику LD . . .	13
4.14	Запуск исполняемого файла	14
4.15	Копия файлов в локальный репозиторий	14
4.16	Добавление файлов на GitHub	14
4.17	Отправка файлов	14

1 Цель работы

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

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

Основными функциональными элементами любой 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 плате. Основной задачей процессора является обработка информации, а также организация координации всех узлов компьютера. В состав центрального процессора 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. 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

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. Периферийные устройства в составе ЭВМ: - устройства внешней памяти, которые предназначены для долговременного хранения больших объёмов данных. - 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.

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. 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.

Язык ассемблера (assembly language, сокращённо asm) — машинно-ориентированный язык низкого уровня. 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.

4 Выполнение лабораторной работы

4.1 Создание программы Hello world!

С помощью команды `cd` перемещаюсь в каталог, в котором буду работать (рис. -4.1).

```
[mpzarickaya@fedora ~]$ mkdir -p ~/work/arch-pc/lab04  
[mpzarickaya@fedora ~]$ cd ~/work/arch-pc/lab04  
[mpzarickaya@fedora lab04]$ touch hello.asm  
[mpzarickaya@fedora lab04]$ gedit hello.asm
```

Рис. 4.1: Перемещение между директориями

Создаю в текущем каталоге пустой текстовый файл `hello.asm`, используя `touch` (рис. -4.2).

```
[mpzarickaya@fedora ~]$ mkdir -p ~/work/arch-pc/lab04  
[mpzarickaya@fedora ~]$ cd ~/work/arch-pc/lab04  
[mpzarickaya@fedora lab04]$ touch hello.asm  
[mpzarickaya@fedora lab04]$ gedit hello.asm
```

Рис. 4.2: Создание пустого текстового файла

Открываю созданный файл в текстовом редакторе (рис. -4.3).

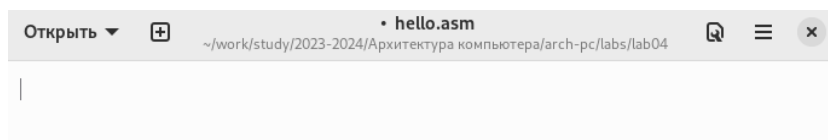


Рис. 4.3: Открытие файла в текстовом редакторе

Ввожу в файл программу для вывода “Hello word!” (рис. -4.4).



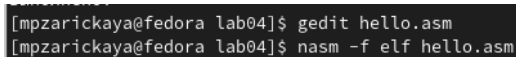
```
Открыть ▾ + hello.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04

; 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 80h ; Вызов ядра
mov eax,1 ; Системный вызов для выхода (sys_exit)
mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
int 80h ; Вызов ядра
```

Рис. 4.4: Заполнение файла

4.2 Работа с транслятором NASM

Компилирую текст программы для вывода “Hello world!” в объектный код с помощью транслятора NASM, далее проверяю правильность выполнения команды с помощью ls.(рис. -4.5)



```
[mpzarickaya@fedora lab04]$ gedit hello.asm
[mpzarickaya@fedora lab04]$ nasm -f elf hello.asm
```

Рис. 4.5: Компиляция текста программы

4.3 Работа с расширенным синтаксисом командной строки NASM

Выполняю команду, которая скомпилирует файл `hello.asm` в `obj.o`, при этом формат выходного файла будет `elf`, и в него будут включены символы для отладки (опция `-g`), кроме того, будет создан файл листинга `list.lst` (опция `-l`). С помощью команды `ls` проверяю, что файлы были созданы. (рис. -4.6).

```
[mpzarickaya@fedora lab04]$ nasm -f elf hello.asm
[mpzarickaya@fedora lab04]$ nasm -o obj.o -f elf -g -l list.lst hello.asm
[mpzarickaya@fedora lab04]$ ls
hello.asm  hello.o  list.lst  obj.o
[mpzarickaya@fedora lab04]$ ld -m elf_i386 hello.o -o hello
```

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`. Затем с помощью `ls` проверяю, что исполняемый файл `hello` был создан (рис. -4.7).

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

Рис. 4.7: Передача объектного файла на обработку компоновщику

Выполняю следующую команду (рис. -4.8). Исполняемый файл будет иметь имя `main`, т.к. после ключа `-o` было задано значение `main`. Объектный файл, из которого собран этот исполняемый файл, имеет имя `obj.o`

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

Рис. 4.8: Передача объектного файла на обработку компоновщику

4.5 Запуск исполняемого файла

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

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

Рис. 4.9: Запуск исполняемого файла

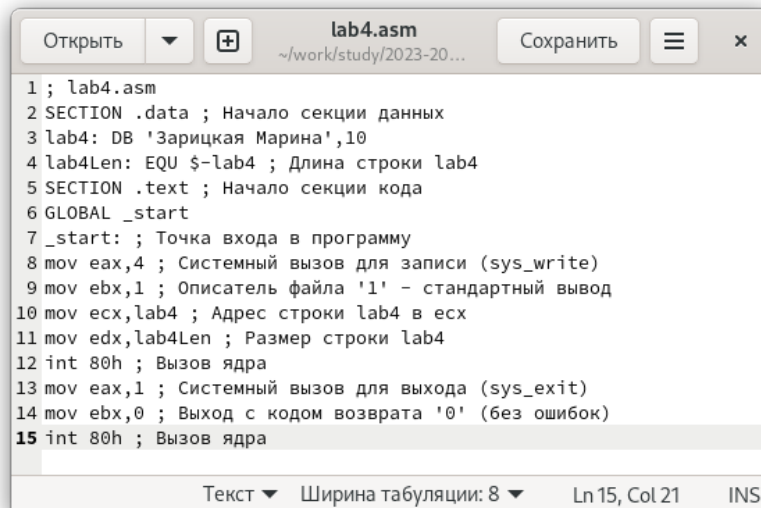
4.6 Выполнение заданий для самостоятельной работы.

С помощью команды cp создаю в текущем каталоге копию файла hello.asm с именем lab4.asm. Проверяю с помощью ls (рис. -4.10).

```
[mpzarickaya@fedora lab04]$ cp hello.asm lab4.asm  
[mpzarickaya@fedora lab04]$ ls  
hello      hello.o  list.lst  obj.o      report  
hello.asm  lab4.asm  main      presentation
```

Рис. 4.10: Создание копии файла

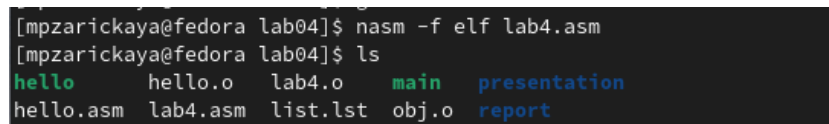
С помощью текстового редактора gedit открываю файл lab4.asm и вношу изменения в программу так, чтобы вместо Hello world! на экран выводилась строка с моими фамилией и именем. (рис. -4.11).



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

Рис. 4.11: Изменение программы

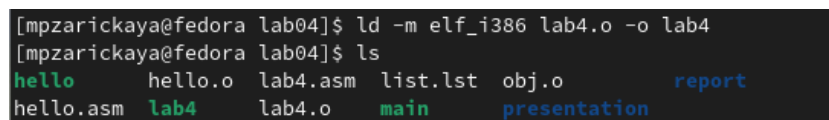
Транслирую текст программы в объектный файл и проверяю с помощью утилиты ls, что файл lab4.o создан. (рис. -4.12).



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

Рис. 4.12: Компиляция текста программы

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



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

Рис. 4.13: Передача объектного файла на обработку компоновщику LD

Запускаю исполняемый файл lab4, на экран действительно выводятся мои фамилия и имя (рис. -4.14).

```
[mpzarickaya@fedora lab04]$ ./lab4
Зарицкая Марина
```

Рис. 4.14: Запуск исполняемого файла

Копирую файлы hello.asm и lab4.asm в свой локальный репозиторий в каталог ~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04 (рис. -4.15).

```
[mpzarickaya@fedora lab04]$ cp hello.asm lab4.asm ~/work/study/2023-2024/"Архитектура компьютера"/arch-pc/labs/lab04
```

Рис. 4.15: Копия файлов в локальный репозиторий

С помощью команд git add . и git commit добавляю файлы на GitHub (рис. -4.16).

```
[mpzarickaya@fedora lab04]$ git add .
[mpzarickaya@fedora lab04]$ git commit -m "Add existing files"
[master 872444c] Add existing files
3 files changed, 232 insertions(+)
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/lab4.asm
create mode 100644 labs/lab04/report/Л04_Зарицкая_отчет.md
```

Рис. 4.16: Добавление файлов на GitHub

Отправляю файлы на сервер с помощью команды git push (рис. -4.17).

```
[mpzarickaya@fedora lab04]$ git push
Перечисление объектов: 12, готово.
Подсчет объектов: 100% (12/12), готово.
При сжатии изменений используется до 2 потоков
Сжатие объектов: 100% (8/8), готово.
Запись объектов: 100% (8/8), 5.71 КиБ | 1.43 МиБ/с, готово.
Всего 8 (изменений 4), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (4/4), completed with 3 local objects.
To github.com:mpzarickaya/study_2023-2024_arhpc.git
 c61d0e5..872444c master -> master
```

Рис. 4.17: Отправка файлов

5 Выводы

При выполнении данной лабораторной работы я освоила процедуры компиляции и сборки программ, написанных на ассемблере NASM.