

Отчёта по лабораторной работе №4

Дисциплина: архитектура компьютера

Тищенко Диана Борисовна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	10
4.1	Создание программы Hello world!	10
4.2	Работа с транслятором NASM	11
4.3	Работа с расширенным синтаксисом командной строки NASM . . .	11
4.4	Работа с компоновщиком LD	12
4.5	Запуск исполняемого файла	12
4.6	Выполнение заданий для самостоятельной работы	13
5	Выводы	15

Список иллюстраций

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

Список таблиц

1 Цель работы

Цель данной лабораторной работы - освоить 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 6 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

7 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 утилиты `cd` перемещаюсь в каталог, в котором буду работать (рис. 4.1)

```
> cd /home/dina/work/study/2024-2025/"Архитектура компьютера"/study_2024-2025_arh-pc/labs/lab04/report
```

Рис. 4.1: Перемещение между директориями

Создаю в текущем каталоге пустой текстовый файл `hello.asm` с помощью утилиты `touch` (рис. 4.2)

```
> touch hello.asm
```

Рис. 4.2: Создание пустого файла

Открываю созданный файл в текстовом редакторе `gedit` (рис. 4.3).

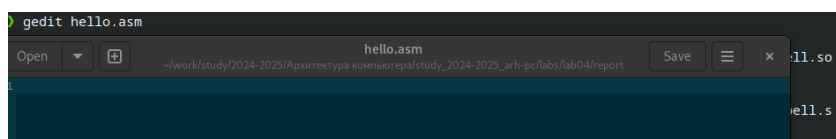
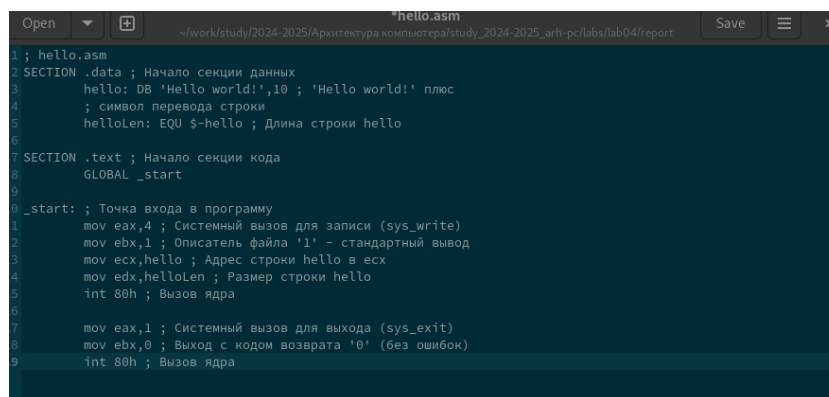


Рис. 4.3: Открытие файла в текстовом редакторе

Заполняю файл, вставляя в него программу для вывода “Hello word!” (рис. 4.4).

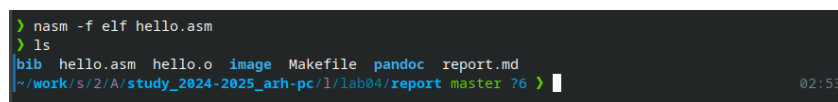


```
1; hello.asm
2SECTION .data ; Начало секции данных
3    hello: DB 'Hello world!',10 ; 'Hello world!' плюс
4            ; символ перевода строки
5            helloLen: EQU $-hello ; Длина строки hello
6
7SECTION .text ; Начало секции кода
8    GLOBAL _start
9
10_start: ; Точка входа в программу
11    mov eax,4 ; Системный вызов для записи (sys_write)
12    mov ebx,1 ; Описание файла '1' - стандартный вывод
13    mov ecx,hello ; Адрес строки hello в ecx
14    mov edx,helloLen ; 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. 4.4: Заполнение файла

4.2 Работа с транслятором NASM

Превращаю текст программы для вывода “Hello world!” в объектный код с помощью транслятора NASM, используя команду `nasm -f elf hello.asm`, ключ `-f` указывает транслятору `nasm`, что требуется создать бинарный файл в формате ELF (рис. 4.5). Далее проверяю правильность выполнения команды с помощью утилиты `ls`: действительно, создан файл “hello.o”



```
> nasm -f elf hello.asm
> ls
bib hello.asm hello.o image Makefile pandoc report.md
~/work/s/2/A/study_2024-2025_arh-pc/1/lab04/report master ?6 > |
```

Рис. 4.5: Компиляция текста программы

4.3 Работа с расширенным синтаксисом командной строки NASM

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 10` будет создан файл листинга `list.lst` (рис. 4.6). Далее проверяю с помощью утилиты `ls` правильность выполнения команды.

```

> nasm -o obj.o -f elf -g -l list.lst hello.asm
> ls
bib hello.asm hello.o image list.lst Makefile obj.o pandoc report.md
~/work/s/2/A/study_2024-2025_arh-pc/1/1ab04/report master ?9 >
02:55

```

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 (рис. 4.7). Ключ -o задает имя создаваемого исполняемого файла. Далее проверяю с помощью утилиты ls правильность выполнения команды.

```

> ld -m elf_i386 hello.o -o hello
> ls
bib hello hello.asm hello.o image list.lst Makefile obj.o pandoc report.md
~/work/s/2/A/study_2024-2025_arh-pc/1/1ab04/report master ?9 >
02:55

```

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

```

> ld -m elf_i386 obj.o -o main
> ls
bib hello hello.asm hello.o image list.lst main Makefile obj.o pandoc report.md
~/work/s/2/A/study_2024-2025_arh-pc/1/1ab04/report master ?9 >
02:55

```

Рис. 4.8: Передача объектного файла на обработку компоновщику

4.5 Запуск исполняемого файла

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

```

> ./hello
Hello world!
~/work/s/2/A/study_2024-2025_arh-pc/1/1ab04/report master ?9 >
02:55

```

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).

```
> cp hello.asm lab4.asm
> ls
bib hello hello.o image lab4.asm list.lst main Makefile obj.o pandoc report.md
```

Рис. 4.10: Создание копии файла

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

```
|, hello.asm
SECTION .data ; Начало секции данных
    lab4: DB 'Tischenko Diana',10

    lab4Len: EQU $-lab4 ; Длина строки lab4

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,lab4 ; Адрес строки lab4 в ecx
    mov edx,lab4Len ; Размер строки lab
    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. 4.11: Изменение программы

Компилирую текст программы в объектный файл (рис. 4.12). Проверяю с помощью утилиты `ls`, что файл `lab4.o` создан.

```
> nasm -f elf lab4.asm
lab4.asm:3: error: comma expected after operand, got 'Diana'
> nasm -f elf lab4.asm
> ls
bib hello.asm image lab4.o main obj.o report.md
hello hello.o lab4.asm list.lst Makefile pandoc
```

Рис. 4.12: Компиляция текста программы

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

```
> ld -m elf_i386 lab4.o -o lab4
> ls
bib hello.asm image lab4.asm list.lst Makefile pandoc
hello hello.o lab4 lab4.o main obj.o report.md
```

Рис. 4.13: Передача объектного файла на обработку компоновщику

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

```
> ./lab4
Tischenko Diana
```

Рис. 4.14: Запуск исполняемого файла

С помощью команд `git add .` и `git commit` добавляю файлы на GitHub, комментируя действие как добавление файлов для лабораторной работы №4 (рис. 4.15).

```
> git add .
> git commit -m "Add files for lab4"
[master d3fd669] Add files for lab4
16 files changed, 58 insertions(+)
create mode 100755 labs/lab04/report/hello
create mode 100644 labs/lab04/report/hello.asm
create mode 100644 labs/lab04/report/hello.o
create mode 100644 labs/lab04/report/image/1.png
create mode 100644 labs/lab04/report/image/2.png
create mode 100644 labs/lab04/report/image/3.png
create mode 100644 labs/lab04/report/image/4.png
create mode 100644 labs/lab04/report/image/5.png
create mode 100644 labs/lab04/report/image/6.png
```

Рис. 4.15: Добавление файлов на GitHub

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

```
> git push
Enumerating objects: 27, done.
Counting objects: 100% (27/27), done.
Delta compression using up to 12 threads
Compressing objects: 100% (22/22), done.
Writing objects: 100% (22/22), 137.57 KiB | 1.30 MiB/s, done.
Total 22 (delta 6), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (6/6), completed with 2 local objects.
To github.com:Lynx1425/study_2024-2025_arh-pc.git
 c589a32..d3fd669  master -> master
~/work/s/2/A/study_2024-2025_arh-pc/l/lab04/report master > [] 04:09:42
```

Рис. 4.16: Отправка файлов

5 Выводы

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