

Отчёт по лабораторной работе №4

Архитектура компьютеров и операционные системы.

Брыляков Никита Евгеньевич

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	9
4.1	Программа Hello world!	9
4.2	Транслятор NASM	10
4.3	Расширенный синтаксис командной строки NASM	11
4.4	Компоновщик LD	11
4.5	Запуск исполняемого файла	12
4.6	Задание для самостоятельной работы	12
5	Вывод	15
6	Список литературы	16

Список иллюстраций

4.1	Создание каталога	9
4.2	Переход в каталог	9
4.3	Создание текстового файла	9
4.4	Открытие файла в текстовом редакторе	10
4.5	Заполнение файла	10
4.6	Компиляция текста программы	10
4.7	Компиляция текста программы	11
4.8	Передача объектного файла на обработку компоновщику	11
4.9	Передача объектного файла на обработку компоновщику	12
4.10	Запуск исполняемого файла	12
4.11	Создание копии файла	12
4.12	Изменение программы	13
4.13	Компиляция текста программы	13
4.14	Передача объектного файла на обработку компоновщику	13
4.15	Запуск исполняемого файла	13
4.16	Копирование в репозиторий	14
4.17	Загрузка на GitHub	14

1 Цель работы

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

2 Задание

1. Программа Hello world!
2. Транслятор NASM
3. Расширенный синтаксис командной строки NASM
4. Компоновщик LD
5. Запуск исполняемого файла
6. Задание 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

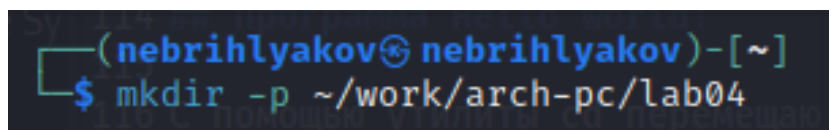
следующей команде.

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 Программа Hello world!

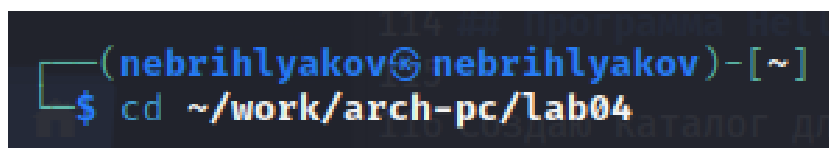
Создаю каталог для работы с программами на языке ассемблера NASM: (рис. [4.1]).



```
(nebrihlyakov@nebrihlyakov)-[~]  
$ mkdir -p ~/work/arch-pc/lab04
```

Рис. 4.1: Создание каталога

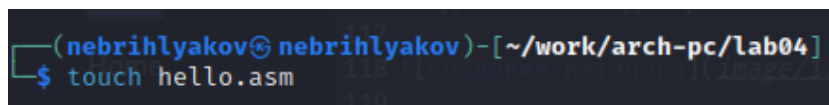
Перехожу в созданный каталог (рис. [4.2]).



```
(nebrihlyakov@nebrihlyakov)-[~]  
$ cd ~/work/arch-pc/lab04
```

Рис. 4.2: Переход в каталог

Создаю текстовый файл с именем hello.asm (рис. [4.3]).



```
(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab04]  
$ touch hello.asm
```

Рис. 4.3: Создание текстового файла

Открываю этот файл с помощью mousepad (рис. [4.4]).

```
(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab04]
$ mousepad hello.asm
```

Рис. 4.4: Открытие файла в текстовом редакторе

Ввожу в него следующий текст: (рис. [4.5]).

```
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 ; Описатель 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. 4.5: Заполнение файла

4.2 Транслятор NASM

Для компиляции приведённого выше текста программы «Hello World» пишу необходимую команду и сразу проверяю правильность выполнения с помощью команды `ls`: (рис. [4.6]).

```
(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab04]
$ nasm -f elf hello.asm
(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab04]
$ ls
hello.asm  hello.o
```

Рис. 4.6: Компиляция текста программы

4.3 Расширенный синтаксис командной строки NASM

Скомпилирую исходный файл `hello.asm` в `obj.o` и сразу проверяю правильность выполнения с помощью команды `ls`: (рис. [4.7]).

```
(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab04]
$ nasm -o obj.o -f elf -g -l list.lst hello.asm

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab04]
$ ls
hello.asm  hello.o  list.lst  obj.o
```

Рис. 4.7: Компиляция текста программы

4.4 Компоновщик LD

Передаю объектный файл `hello.o` на обработку компоновщику LD, чтобы получить исполняемый файл `hello` и проверяю с помощью `ls` правильность выполнения команды. (рис. [4.8]).

```
(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab04]
$ ld -m elf_i386 hello.o -o hello

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab04]
$ ls
hello  hello.asm  hello.o  list.lst  obj.o
```

Рис. 4.8: Передача объектного файла на обработку компоновщику

Выполняю следующую команду. Исполняемый файл будет иметь имя `main`. Проверяю с помощью команды `ls` (рис. [4.9]).

```

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab04]
$ ld -m elf_i386 obj.o -o main

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab04]
$ ls
hello  hello.asm  hello.o  list.lst  main  obj.o

```

Рис. 4.9: Передача объектного файла на обработку компоновщику

4.5 Запуск исполняемого файла

Запускаю на выполнение созданный исполняемый файл hello (рис. [4.10]).

```

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab04]
$ ./hello
Hello world!

```

Рис. 4.10: Запуск исполняемого файла

4.6 Задание для самостоятельной работы

С помощью команды `cp` создайте копию файла `hello.asm` с именем `lab4.asm` (рис. [4.11]).

```

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab04]
$ cp hello.asm lab4.asm

```

Рис. 4.11: Создание копии файла

С помощью текстового редактора `mousepad` открываю файл `lab5.asm` и вношу изменения в текст программы так, чтобы она выводила мои имя и фамилию. (рис. [4.12]).

```

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

```

Рис. 4.12: Изменение программы

Компилирую текст программы в объектный файл и проверяю с помощью ls, что файл lab4.o создан. (рис. [4.13]).

```

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab04]
$ nasm -f elf lab4.asm
(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab04]
$ ls
hello  hello.asm  hello.o  lab4.asm  lab4.o  list.lst  main  obj.o

```

Рис. 4.13: Компиляция текста программы

Передаю объектный файл lab4.o на обработку компоновщику LD, чтобы получить исполняемый файл lab4 (рис. [4.14]).

```

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab04]
$ ld -m elf_i386 lab4.o -o lab4

```

Рис. 4.14: Передача объектного файла на обработку компоновщику

Запускаю исполняемый файл lab4 (рис. [4.15]).

```

(nebrihlyakov@nebrihlyakov)-[~/work/arch-pc/lab04]
$ ./lab4
Brihlyakov Nikita

```

Рис. 4.15: Запуск исполняемого файла

Копирую файлы hello.asm и lab4.asm в свой локальный репозиторий (рис. [4.16]).

```
(nebrihlyakov@nebrihlyakov)~[/work/arch-pc/lab04]
$ cp hello.asm ~/work/study/2023-2024/"Архитектура компьютера"/arch-pc/labs/lab04/

(nebrihlyakov@nebrihlyakov)~[/work/arch-pc/lab04]
$

(nebrihlyakov@nebrihlyakov)~[/work/arch-pc/lab04]
$ cp lab4.asm ~/work/study/2023-2024/"Архитектура компьютера"/arch-pc/labs/lab04/
```

Рис. 4.16: Копирование в репозиторий

Загружаю файлы на Github. (рис. [4.17]).

```
(nebrihlyakov@nebrihlyakov)~[/work/study/2023-2024/Архитектура компьютера/arch-pc]
$ git add .

(nebrihlyakov@nebrihlyakov)~[/work/study/2023-2024/Архитектура компьютера/arch-pc]
$ git commit -m "Add files for lab4"
[master 6e841d1] Add files for lab4
19 files changed, 37 insertions(+), 1 deletion(-)
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/lab4.asm
create mode 100644 labs/lab04/report/image/1.png
create mode 100644 labs/lab04/report/image/10.png
create mode 100644 labs/lab04/report/image/11.png
create mode 100644 labs/lab04/report/image/12.png
create mode 100644 labs/lab04/report/image/13.png
create mode 100644 labs/lab04/report/image/14.png
create mode 100644 labs/lab04/report/image/15.png
create mode 100644 labs/lab04/report/image/16.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
create mode 100644 labs/lab04/report/image/7.png
create mode 100644 labs/lab04/report/image/8.png
create mode 100644 labs/lab04/report/image/9.png

(nebrihlyakov@nebrihlyakov)~[/work/study/2023-2024/Архитектура компьютера/arch-pc]
$ git push
Enumerating objects: 31, done.
Counting objects: 100% (31/31), done.
Delta compression using up to 6 threads
Compressing objects: 100% (25/25), done.
Writing objects: 100% (25/25), 337.08 KiB | 2.98 MiB/s, done.
Total 25 (delta 5), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (5/5), completed with 4 local objects.
To github.com:nebrihlyakov/study_2023-2024_arh-pc.git
e9dfb52..6e841d1 master -> master
```

Рис. 4.17: Загрузка на GitHub

5 Вывод

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

6 Список литературы

- [illegible]