

# **ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4**

**дисциплина: Архитектура компьютера**

Газизянов Владислав Альбертович

# Содержание

<b>1</b>	<b>Цель работы</b>	<b>4</b>
<b>2</b>	<b>Задание</b>	<b>5</b>
<b>3</b>	<b>Теоретическое введение</b>	<b>6</b>
3.1	Основные принципы работы компьютера . . . . .	6
3.2	Ассемблер и язык ассемблера . . . . .	9
3.3	Процесс создания и обработки программы на языке ассемблера .	11
<b>4</b>	<b>Выполнение лабораторной работы</b>	<b>13</b>
<b>5</b>	<b>Выполнение самостоятельной работы</b>	<b>16</b>
<b>6</b>	<b>Выводы</b>	<b>18</b>

# Список иллюстраций

4.1	каталог . . . . .	13
4.2	текстовый файл . . . . .	13
4.3	вводим текст . . . . .	14
4.4	компиляция . . . . .	14
4.5	компиляция в obj.o . . . . .	14
4.6	обработка . . . . .	14
4.7	выполнение команды . . . . .	15
5.1	копируем файлы . . . . .	16
5.2	ФИО . . . . .	16
5.3	транслируем текст . . . . .	16
5.4	копируем в репозиторий . . . . .	17
5.5	Github . . . . .	17

# 1 Цель работы

Целью данной работы является изучение процесса создания и обработки программ на языке ассемблера NASM. Это включает в себя следующие задачи:

1. Изучение основ языка ассемблера NASM.
2. Разработка программы на языке ассемблера NASM.
3. Отладка и тестирование разработанной программы.
4. Анализ результатов работы программы.

## 2 Задание

Овладение процессом компиляции и сборки программ, созданных на языке ассемблера NASM.

## 3 Теоретическое введение

### 3.1 Основные принципы работы компьютера

Основными функциональными элементами любой электронно-вычислительной машины

(ЭВМ) являются центральный процессор, память и периферийные устройства. Взаимодействие этих устройств осуществляется через общую шину, к которой они подклю- чены. Физически шина представляет собой большое количество проводников, соединяющих устройства друг с другом. В современных ком- пьютерах проводники выполнены в виде элек- тропроводящих дорожек на материнской (системной) плате. Основной задачей процессора является обра- ботка информации, а также организация координации всех узлов компьютера. В состав центрального процессора (ЦП) входят следующие устройства:

арифметико-логическое устройство (АЛУ) — выполняет логические и арифметиче- ские 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 со- став 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 (половинки 16-битных регистров).

Например, AH (high AX) — старшие 8 бит регистра AX, AL (low AX) — младшие 8 бит регистра AX.

Таким образом можно отметить, что вы можете написать в своей программе, например,

такие команды (mov – команда пересылки данных на языке ассемблера):

```
mov ax, 1
```

```
mov eax, 1
```

Обе команды поместят в регистр AX число 1. Разница будет заключаться только в том

вторая команда обнулит старшие разряды регистра EAX, то есть после выполнения второй команды в регистре EAX будет число 1. А первая команда оставит в старших разрядах регистра EAX старые данные. И если там были данные, отличные от нуля, то после выполнения первой команды в регистре EAX будет какое-то число, но не 1. А вот в регистре AX будет число 1.

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.

В состав ЭВМ также входят периферийные устройства, которые можно разделить на:

- устройства внешней памяти, которые предназначены для долговременного хранения больших объёмов данных (жёсткие диски, твердотельные накопители, магнитные ленты);
- 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. Программа состоит из машинных команд, которые указывают, какие операции и над какими данными (или операндами), в какой последовательности необходимо выполнить.

Набор машинных команд определяется устройством конкретного процессора. 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. В самом общем виде 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.

Данный алгоритм позволяет выполнить хранящуюся в ОЗУ программу. Кроме того, в зависимости от команды при её выполнении могут проходить не все этапы.

## 3.2 Ассемблер и язык ассемблера

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. Заметим, что 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 — Ассемблер.

Программы, написанные на языке ассемблера, не уступают в качестве и скорости программам, написанным на машинном языке, так как транслятор просто пе-

реводит 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:

- для DOS/Windows: Borland Turbo Assembler (TASM), Microsoft Macro Assembler (MASM) и Watcom assembler (WASM);
- для GNU/Linux: gas (GNU Assembler), использующий AT&T-синтаксис, в отличие от большинства других популярных ассемблеров, которые используют Intel-синтаксис.

Более подробно о языке ассемблера см., например, в [10].

В нашем курсе будет использоваться ассемблер NASM (Netwide Assembler) [7; 12; 14].

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.

Типичный формат записи команд NASM имеет вид:

[метка:] мнемокод [операнд {, операнд}] [; комментарий]

Здесь мнемокод — непосредственно мнемоника инструкции процессору, которая является

обязательной частью команды. Операндами могут быть числа, данные, адреса регистров или адреса оперативной памяти. Метка — это идентификатор, с которым ассемблер ассоциирует некоторое число, чаще всего адрес в памяти. Т.о. 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 идентификаторам

которые пишутся как зарезервированные слова, нужно писать \$, чтобы компилятор трактовал его верно (так называемое экранирование). Максимальная длина идентификатора 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.

### **3.3 Процесс создания и обработки программы на языке ассемблера**

В процессе создания ассемблерной программы можно выделить четыре шага:

- Набор текста программы в текстовом редакторе и сохранение её в отдельном файле.

Каждый файл имеет свой тип (или расширение), который определяет назначение файла. Файлы с исходным текстом программ на языке ассемблера имеют тип `asm`. • Трансляция — преобразование с помощью транслятора, например `nas`, текста программы в машинный код, называемый объектным. На данном этапе также может быть получен листинг программы, содержащий кроме текста программы различную дополнительную информацию, созданную транслятором. Тип объектного файла — `o`, файла листинга — `lst`.

- Компоновка или линковка — этап обработки объектного кода компоновщиком (`ld`), который принимает на вход объектные файлы и собирает по ним исполняемый файл. Исполняемый файл обычно не имеет расширения. Кроме того, можно получить файл карты загрузки программы в ОЗУ, имеющий расширение `map`.
- Запуск программы. Конечной целью является работоспособный исполняемый

файл. Ошибки на предыдущих этапах могут привести к некорректной работе программы, поэтому может присутствовать этап отладки программы при помощи специальной программы — отладчика. При нахождении ошибки необходимо провести коррекцию программы, начиная с первого шага.

Из-за специфики программирования, а также по традиции для создания программ на языке ассемблера обычно пользуются утилитами командной строки (хотя поддержка ассемблера есть в некоторых универсальных интегрированных средах).

## 4 Выполнение лабораторной работы

Создайте каталог для работы с программами на языке ассемблера NASM и  
Перейдите в созданный каталог

```
vagazizyanov@vagazizyanov:~$ mkdir -p ~/work/arch-pc/lab04  
vagazizyanov@vagazizyanov:~$ cd ~/work/arch-pc/lab04~
```

Рис. 4.1: каталог

Создайте текстовый файл с именем hello.asm

```
vagazizyanov@vagazizyanov:~$ cd ~/work/arch-pc/lab04  
vagazizyanov@vagazizyanov:~/work/arch-pc/lab04$ touch hello.asm
```

Рис. 4.2: текстовый файл

откройте этот файл с помощью любого текстового редактора, например, gedit  
и введите в него текст

```

1 ; hello.asm
2 SECTION .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 SECTION .text ; Начало секции кода
7     GLOBAL _start
8 _start: ; Точка входа в программу
9         mov eax,4 ; Системный вызов для записи (sys_write)
10        mov ebx,1 ; Описатель файла '1' - стандартный вывод
11        mov ecx,hello ; Адрес строки hello в ecx
12        mov edx,helloLen ; Размер строки hello
13        int 80h ; Вызов ядра
14
15        mov eax,1 ; Системный вызов для выхода (sys_exit)
16        mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
17        int 80h ; Вызов ядра

```

Рис. 4.3: вводим текст

скомпилируем текст

```
vagazizyanov@vagazizyanov:~/work/arch-pc/lab04$ nasm -f elf hello.asm
```

Рис. 4.4: компиляция

скомпилируем исходный файл hello.asm в obj.o

```
vagazizyanov@vagazizyanov:~/work/arch-pc/lab04$ nasm -o obj.o -f elf -g -l list.lst hello.asm
```

Рис. 4.5: компиляция в obj.o

объектный файл необходимо передать на обработку компоновщику

```
vagazizyanov@vagazizyanov:~/work/arch-pc/lab04$ ld -m elf_i386 hello.o -o hello
```

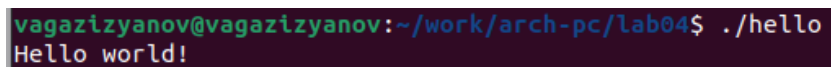
Рис. 4.6: обработка

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

```
vagazizyanov@vagazizyanov:~/work/arch-pc/lab04$ ld -m elf_i386 obj.o -o main
```

Снимок экрана от 2023-10-28 15-45-44.png{#fig:001 width=70%}

Запустить на выполнение созданный исполняемый файл,

A terminal window with a dark background. The prompt is 'vagazizyanov@vagazizyanov:~/work/arch-pc/lab04\$' in green. The command './hello' is entered in blue. The output 'Hello world!' is shown in white on the next line.

```
vagazizyanov@vagazizyanov:~/work/arch-pc/lab04$ ./hello
Hello world!
```

Рис. 4.7: выполнение команды

## 5 Выполнение самостоятельной работы

В каталоге `~/work/arch-pc/lab04` с помощью команды `cp` создайте копию файла `hello.asm` с именем `lab4.asm`

```
vagazizyanov@vagazizyanov:~/work/arch-pc/lab04$ cp hello.asm lab4.asm
```

Рис. 5.1: копируем файлы

С помощью любого текстового редактора внесите изменения в текст программы в файле `lab4.asm` так, чтобы вместо `Hello world!` на экран выводилась строка с вашими фамилией и именем

```
1 ; hello.asm
2 SECTION .data ; Начало секции данных
3     hello: DB 'Газизьянов Владислав',10 ; 'Hello world!' плюс
4 ; символ перевода строки
```

Рис. 5.2: ФИО

Оттранслируйте полученный текст программы `lab4.asm` в объектный файл. Выполните компоновку объектного файла и запустите получившийся исполняемый файл.

```
vagazizyanov@vagazizyanov:~/work/arch-pc/lab04$ cp hello.asm lab4.asm
vagazizyanov@vagazizyanov:~/work/arch-pc/lab04$ gedit lab4.asm
vagazizyanov@vagazizyanov:~/work/arch-pc/lab04$ nasm -f elf lab4.asm
vagazizyanov@vagazizyanov:~/work/arch-pc/lab04$ nasm -o obj.o -f elf -g -l list.lst lab4.asm
vagazizyanov@vagazizyanov:~/work/arch-pc/lab04$ ld -m elf_i386 lab4.o -o hello
vagazizyanov@vagazizyanov:~/work/arch-pc/lab04$ ld -m elf_i386 obj.o -o main
vagazizyanov@vagazizyanov:~/work/arch-pc/lab04$ ./lab4
bash: ./lab4: Нет такого файла или каталога
vagazizyanov@vagazizyanov:~/work/arch-pc/lab04$ ./hello
Газизьянов Владислав
```

Рис. 5.3: транслируем текст



Скопируйте файлы hello.asm и lab4.asm в Ваш локальный репозиторий в каталог ~/work/study2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/.

```
vagazizyanov@vagazizyanov:~/work/arch-pc/lab04$ cp hello.asm ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/labs/lab04
vagazizyanov@vagazizyanov:~/work/arch-pc/lab04$ cp lab4.asm ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/labs/lab04
```

Рис. 5.4: копируем в репозиторий

Загрузите файлы на Github.

```
vagazizyanov@vagazizyanov:~/work/study/2023-2024/Архитектура\ компьютера/arch-pc$
git add .
vagazizyanov@vagazizyanov:~/work/study/2023-2024/Архитектура\ компьютера/arch-pc$
git commit -am 'feat(main):add file lab04'
[master 3948f2f] feat(main):add file lab04
15 files changed, 292 insertions(+), 120 deletions(-)
create mode 100644 labs/lab03/report.zip
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/lab4.asm
create mode 100644 labs/lab04/report/image/Снимок экрана от 2023-10-28 15-34-45.png
create mode 100644 labs/lab04/report/image/Снимок экрана от 2023-10-28 15-35-29.png
create mode 100644 labs/lab04/report/image/Снимок экрана от 2023-10-28 15-40-19.png
create mode 100644 labs/lab04/report/image/Снимок экрана от 2023-10-28 15-42-09.png
create mode 100644 labs/lab04/report/image/Снимок экрана от 2023-10-28 15-43-32.png
create mode 100644 labs/lab04/report/image/Снимок экрана от 2023-10-28 15-44-53.png
create mode 100644 labs/lab04/report/image/Снимок экрана от 2023-10-28 15-45-44.png
create mode 100644 labs/lab04/report/image/Снимок экрана от 2023-10-28 15-46-38.png
create mode 100644 labs/lab04/report/report.docx
rewrite labs/lab04/report/report.md (73%)
rename labs/lab04/report/{image/placeholder_800_600_tech.jpg => report.pdf} (73%)
vagazizyanov@vagazizyanov:~/work/study/2023-2024/Архитектура\ компьютера/arch-pc$ git push
Перечисление объектов: 31, готово.
Подсчет объектов: 100% (31/31), готово.
Сжатие объектов: 100% (23/23), готово.
Запись объектов: 100% (23/23), 1.98 МиБ | 1.36 МиБ/с, готово.
Всего 23 (изменений 5), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (5/5), completed with 4 local objects.
To github.com:ONYX8880/study_2023-2024_arh-pc.git
7599ae0..3948f2f master -> master
```

Рис. 5.5: Github

## 6 Выводы

Были изучены основы языка ассемблера NASM, включая его синтаксис, структуру программы и основные команды. Была разработана программа на языке ассемблера NASM, проведена ее отладка и тестирование.