

# Лабораторная работа номер 4

## Создание и процесс обработки программ на языке ассемблера NASM

Соловбев Богдан Михайлович

### Содержание

#### 1 Цель работы

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

#### 2 Задание

Здесь приводится описание задания в соответствии с рекомендациями методического пособия и выданным вариантом.

#### 3 Теоретическое введение

##### 3.1 Основные принципы работы компьютера

Основными функциональными элементами любой электронно-вычислительной машины

(ЭВМ) являются центральный процессор, память и периферийные устройства (рис. 4.1). 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 (системной) плате. Основной задачей процессора является обработка информации, а также организация координации всех узлов компьютера. В состав центрального процессора (ЦП) 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.

Структурная схема ЭВМ (рис. 1).



Figure 1: Архитектура ЭВМ

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.

Регистры процессора (рис. 2).

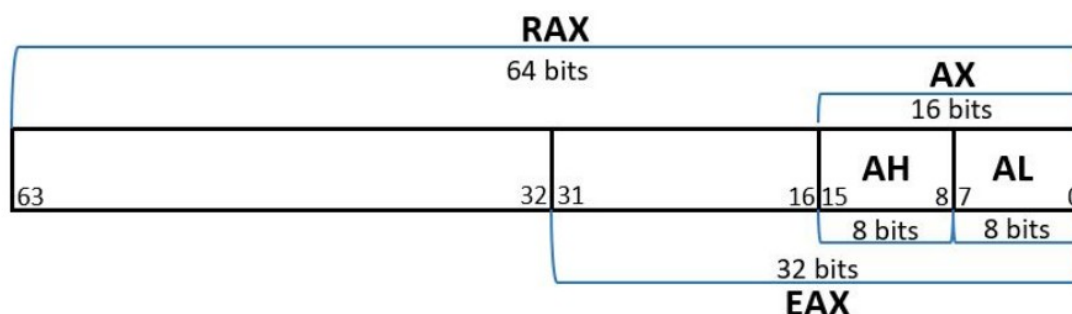


Figure 2: Регистры процессора

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, что 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. Более подробно введение о теоретических основах архитектуры ЭВМ см. в [9; 11].

### 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 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: • 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 идентификаторами, 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

Процесс создания ассемблерной программы можно изобразить в виде следующей схемы (рис. 3).



Figure 3: Процесс создания ассемблерной программы

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 `nasm`, 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 tar.

- 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).

Имя каталога	Описание каталога
/	Корневая директория, содержащая всю файловую
/bin	Основные системные утилиты, необходимые как в однопользовательском режиме, так и при обычной работе всем пользователям
/etc	Общесистемные конфигурационные файлы и файлы конфигурации установленных программ
/home	Содержит домашние директории пользователей, которые, в свою очередь, содержат персональные настройки и данные пользователя
/media	Точки монтирования для сменных носителей
/root	Домашняя директория пользователя root
/tmp	Временные файлы
/usr	Вторичная иерархия для данных пользователя



Более подробно об Unix см. в [1–6].

## 4 Выполнение лабораторной работы

### 4.1 Программа Hello World!

Создаю в папке “work” новую папку lab04 командой “mkdir” и перехожу в неё командой cd (рис. 4).

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

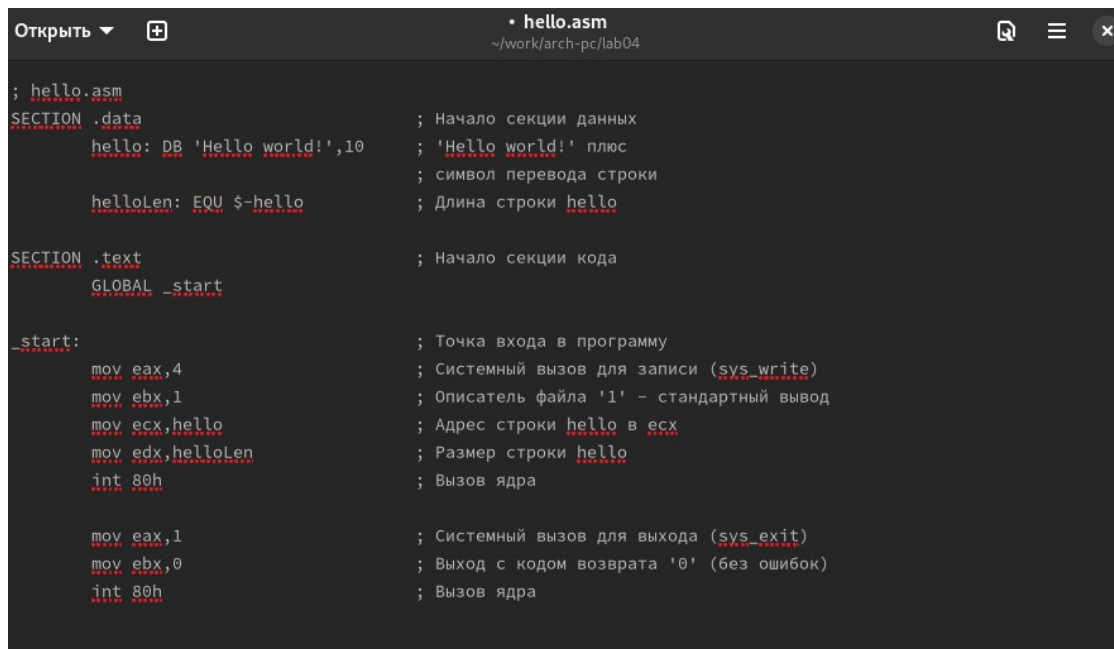
Figure 4: Создание папки

Создаю текстовый файл “hello.asm” и открываю его (рис. 5).

```
[bmsolovjev@fedora lab04]$ touch hello.asm
[bmsolovjev@fedora lab04]$ gedit hello.asm
```

Figure 5: Создание текстового файла

Ввожу в созданный файл скопированный текст (рис. 6).



```
Открыть ▾ + hello.asm
~/work/arch-pc/lab04

; hello.asm
SECTION .data
    hello: DB 'Hello world!',10
    helloLen: EQU $-hello
SECTION .text
    GLOBAL _start
_start:
    mov eax,4
    mov ebx,1
    mov ecx,hello
    mov edx,helloLen
    int 80h
    mov eax,1
    mov ebx,0
    int 80h
```

Comments in Russian:

- ; Начало секции данных
- ; 'Hello world!' плюс
- ; символ перевода строки
- ; Длина строки hello
- ; Начало секции кода
- ; Точка входа в программу
- ; Системный вызов для записи (sys\_write)
- ; Описатель файла '1' – стандартный вывод
- ; Адрес строки hello в ecx
- ; Размер строки hello
- ; Вызов ядра
- ; Системный вызов для выхода (sys\_exit)
- ; Выход с кодом возврата '0' (без ошибок)
- ; Вызов ядра

Figure 6: Команда Hello World



## 4.2 Транслятор NASM

Компилирую файл `hello.asm` (рис. 7).

```
[bmsolovjev@fedora lab04]$ nasm -f elf hello.asm
[bmsolovjev@fedora lab04]$ ls
hello.asm  hello.o
```

Figure 7: Компиляция файла `hello.asm`

## 4.3 Расширенный синтаксис командной строки NASM

Выполняю следующую команду (рис. 8).

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

Figure 8: Компиляция `.asm` в `.o`

## 4.4 Компоновщик LD

Объектный файл передаю на обработку компоновщику (рис. 9).

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

Figure 9: Компоновка файла

Выполняю следующую команду (рис. 10).

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

Figure 10: Создание исполняемого файла

## 4.5 Запуск исполняемого файла

Запускаю исполняемый файл (рис. 11).

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

Figure 11: Запуск программы `Hello World!`

# 5 Задания для самостоятельной работы

Создаю копию файла и переименовываю его (рис. 12).

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

Figure 12: Создание копии файла

Вношу изменения в текст программы, чтобы она выводила моё Имя и Фамилию (рис. 13).

```
; hello.asm
SECTION .data                                ; Начало секции данных
    hello: DB 'Bogdan Solovjev|',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
```

Figure 13: Изменение текста программы

Создаю объектный файл (рис. 14).

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

Figure 14: Создание объектного файла

Создаю исполняемый файл (рис. 15).

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

Figure 15: Создание исполняемого файла

Запускаю файл (рис. 16).

```
[bmsolovjev@fedora lab04]$ ./lab4
Bogdan Solovjev
```

Figure 16: Запуск исполняемого файла

Копирую файл и переношу его (рис. 17).

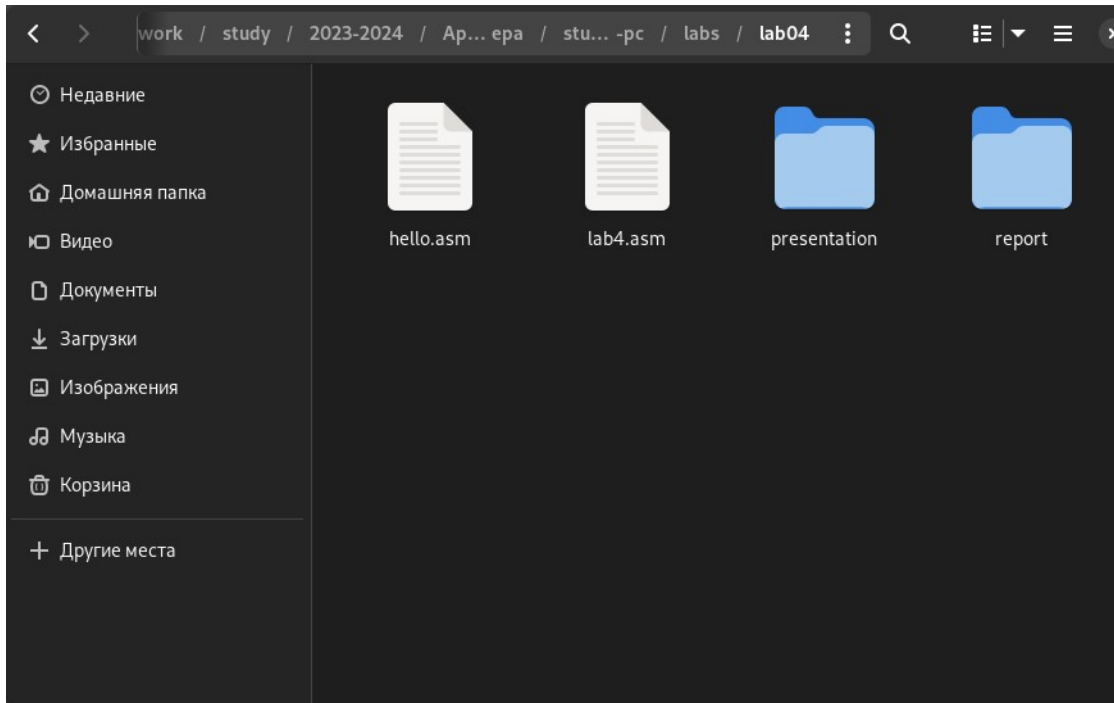


Figure 17: Копирование и перенос файла

Отправляю изменения на GitHub(рис. ??).

```
[bmsolovjev@fedora study_2023-2024_arch-pc]$ git add .
[bmsolovjev@fedora study_2023-2024_arch-pc]$ git commit -m "Add files for lab4"
[master e17e08c] Add files for lab4
18 files changed, 67 insertions(+), 1 deletion(-)
create mode 100644 123
delete mode 100644 labs/lab03/report/./~lock.л03_Соловьев_отчёт.docx#
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/lab4.asm
create mode 100644 labs/lab04/report/image/1_1.png
create mode 100644 labs/lab04/report/image/1_2.png
create mode 100644 labs/lab04/report/image/1_3.png
create mode 100644 labs/lab04/report/image/2_1.png
create mode 100644 labs/lab04/report/image/3_1.png
create mode 100644 labs/lab04/report/image/4_1.png
create mode 100644 labs/lab04/report/image/4_2.png
create mode 100644 labs/lab04/report/image/5_1.png
create mode 100644 labs/lab04/report/image/6.png
create mode 100644 labs/lab04/report/image/6_2.png
create mode 100644 labs/lab04/report/image/6_3.png
create mode 100644 labs/lab04/report/image/6_4.png
create mode 100644 labs/lab04/report/image/6_5.png
create mode 100644 labs/lab04/report/image/6_6.png
[bmsolovjev@fedora study_2023-2024_arch-pc]$ git push
Перечисление объектов: 32, готово.
Подсчет объектов: 100% (32/32), готово.
При сжатии изменений используется до 4 потоков
Сжатие объектов: 100% (25/25), готово.
Запись объектов: 100% (25/25), 191.40 КиБ | 1.43 МиБ/с, готово.
Всего 25 (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:bmsolovjev/study_2023-2024_arch-pc.git
 a570da1..e17e08c master -> master
```

#

## Выводы

Я освоил процедуры компиляции и сборки программ, написанных на ассемблере NASM.

## Список литературы

1. GNU Bash Manual [Электронный ресурс]. Free Software Foundation, 2016. URL: <https://www.gnu.org/software/bash/manual/>.
2. Newham C. [Learning the bash Shell: Unix Shell Programming](#). O'Reilly Media, 2005. 354 с.
3. Zarrelli G. Mastering Bash. Packt Publishing, 2017. 502 с.
4. Robbins A. [Bash Pocket Reference](#). O'Reilly Media, 2016. 156 с.
5. Таненбаум Э. Архитектура компьютера. 6-е изд. СПб.: Питер, 2013. 874 с.
6. Таненбаум Э., Бос Х. Современные операционные системы. 4-е изд. СПб.: Питер, 2015. 1120 с.