

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

по лабораторной работе № 4

дисциплина: Архитектура компьютера

Студент: Сачковская София Александровна

Группа: НКАбд-06-25

МОСКВА

2025 г.

Содержание

1. Цель работы
2. Теоретическое введение
3. Выполнение лабораторной работы
4. Выполнение самостоятельных заданий
5. Выводы

Список иллюстраций:

Рис.1.1.....	5
Рис.1.2.....	6
Рис.1.3.....	9
Рис.2.1.....	11
Рис.2.2.....	11
Рис.2.3.....	11
Рис.2.4.....	11
Рис.2.5.....	11
Рис.2.6.....	12
Рис.2.7.....	12
Рис.2.8.....	12
Рис.2.9.....	12
Рис.3.1.....	13
Рис.3.2.....	13
Рис.3.3.....	13
Рис.3.4.....	14

1.Цель работы

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

2. Теоретическое введение

Основными функциональными элементами любой электронно-вычислительной машины (ЭВМ) являются центральный процессор, память и периферийные устройства (рис. 1.1). 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

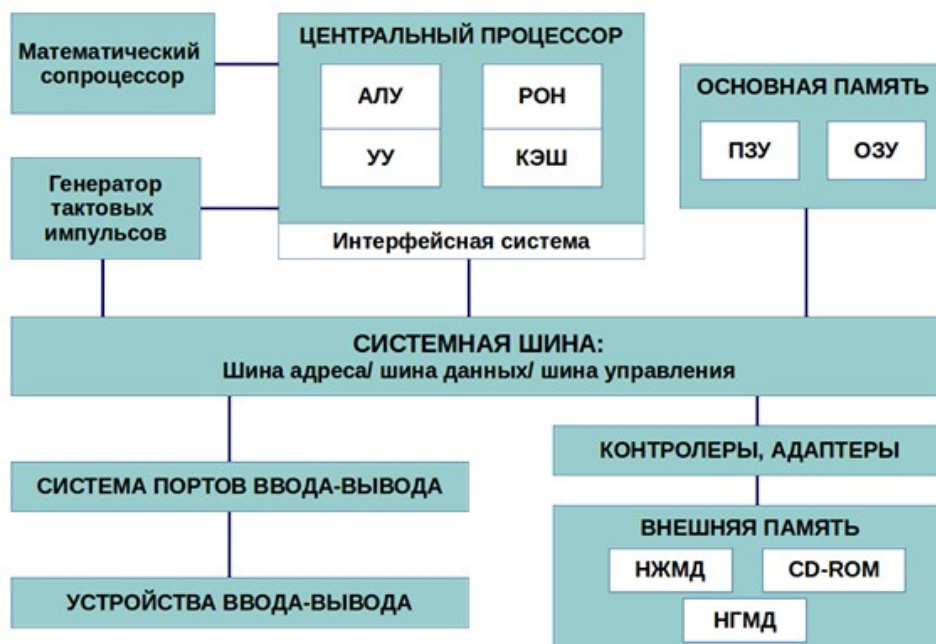


Рис 1.1. Структурная схема ЭВМ

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.

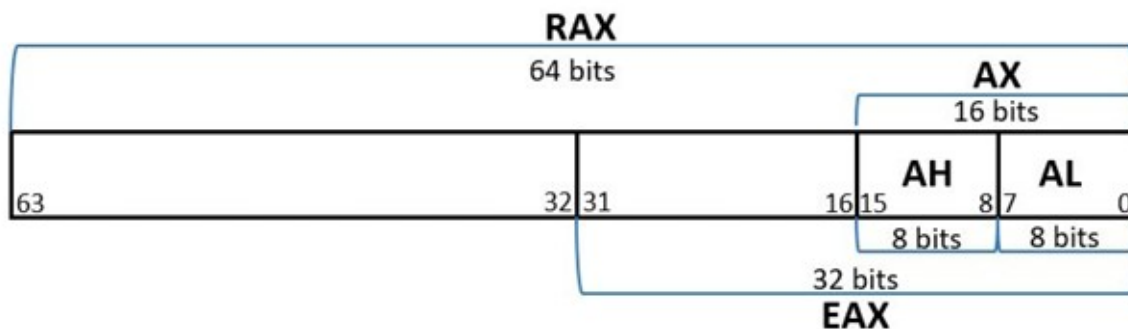


Рис. 1.2. 64-битный регистр процессора 'RAX'

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 длительного 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].

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.

Процесс создания и обработки программы на языке ассемблера

Процесс создания ассемблерной программы можно изобразить в виде следующей схемы (рис. 1.3)



Рис. 1.3. 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 `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).

3.Выполнение лабораторной работы

Создадим каталог для работы с программами на языке ассемблера NASM:

```
sasachkovskaya@dk3n55 ~ $ mkdir -p ~/work/arch-pc/lab04
```

Рис.2.1.

Перейдём в созданный каталог

```
sasachkovskaya@dk3n55 ~ $ cd ~/work/arch-pc/lab04
sasachkovskaya@dk3n55 ~/work/arch-pc/lab04 $
```

Рис.2.2.

Создадим текстовый файл с именем hello.asm

```
sasachkovskaya@dk3n55 ~/work/arch-pc/lab04 $ touch hello.asm
```

Рис.2.3.

Откроем этот файл с помощью текстового редактора gedit и введем в него текст нашей программы

```
sasachkovskaya@dk3n55 ~/work/arch-pc/lab04 $ gedit hello.asm
```



Рис.2.4.

Для компиляции приведённого выше текста программы «Hello World» в объектный код напомним:

```
sasachkovskaya@dk3n55 ~/work/arch-pc/lab04 $ nasm -f elf hello.asm
```

Рис.2.5.

Скомпилируем исходный файл hello.asm в obj.o и с помощью команды ls проверим, что файлы были созданы.

```
sasachkovskaya@dk3n55 ~/work/arch-pc/lab04 $ nasm -o obj.o -f elf -g -l list.lst hello.asm
sasachkovskaya@dk3n55 ~/work/arch-pc/lab04 $
```

Рис.2.6.

чтобы получить исполняемую программу, передадим объектный файл на обработку компоновщику.

```
...
sasachkovskaya@dk3n55 ~/work/arch-pc/lab04 $ ld -m elf_i386 hello.o -o hello
sasachkovskaya@dk3n55 ~/work/arch-pc/lab04 $
```

Рис.2.7.

Выполним следующую команду:

```
sasachkovskaya@dk3n55 ~/work/arch-pc/lab04 $ ld -m elf_i386 obj.o -o main
sasachkovskaya@dk3n55 ~/work/arch-pc/lab04 $
```

Рис.2.8.

Исполняемый файл также, как и объектный будет иметь имя main

Запустим на выполнение созданный исполняемый файл, находящийся в текущем каталоге:

```
sasachkovskaya@dk3n55 ~/work/arch-pc/lab04 $ ./hello
Hello, world!
sasachkovskaya@dk3n55 ~/work/arch-pc/lab04 $
```

Рис.2.9.

4.Выполнение самостоятельной работы

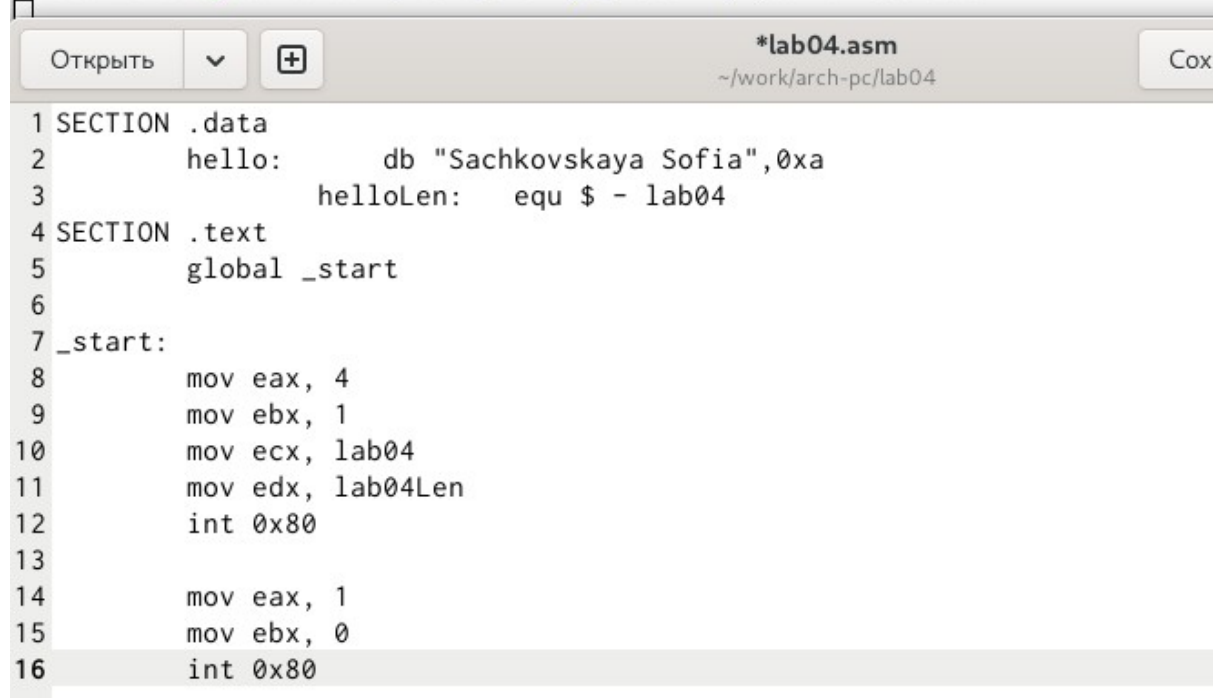
1. В каталоге ~/work/arch-pc/lab04 с помощью команды cp создайте копию файла hello.asm с именем lab4.asm

```
sasachkovskaya@dk3n55 ~/work/arch-pc/lab04 $ cp hello.asm lab04.asm
sasachkovskaya@dk3n55 ~/work/arch-pc/lab04 $
```

Рис.3.1.

2. 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.

```
sasachkovskaya@dk3n55 ~/work/arch-pc/lab04 $ gedit lab04.asm
```



```
1 SECTION .data
2     hello:      db "Sachkovskaya Sofia",0xa
3     helloLen:   equ $ - lab04
4 SECTION .text
5     global _start
6
7 _start:
8     mov eax, 4
9     mov ebx, 1
10    mov ecx, lab04
11    mov edx, lab04Len
12    int 0x80
13
14    mov eax, 1
15    mov ebx, 0
16    int 0x80
```

Рис.3.2.

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

```
sasachkovskaya@dk3n55 ~/work/arch-pc/lab04 $ nasm -f elf lab04.asm
sasachkovskaya@dk3n55 ~/work/arch-pc/lab04 $ ld -m elf_i386 lab04.o -o lab04
sasachkovskaya@dk3n55 ~/work/arch-pc/lab04 $ ./lab04
Sachkovskaya Sofia
sasachkovskaya@dk3n55 ~/work/arch-pc/lab04 $
```

Рис3.3

4. Скопируйте файлы hello.asm и lab4.asm в Ваш локальный репозиторий в каталог ~/work/study/2025-2026/"Архитектура компьютера"/arch-pc/labs/lab04/. Загрузите файлы на Github.

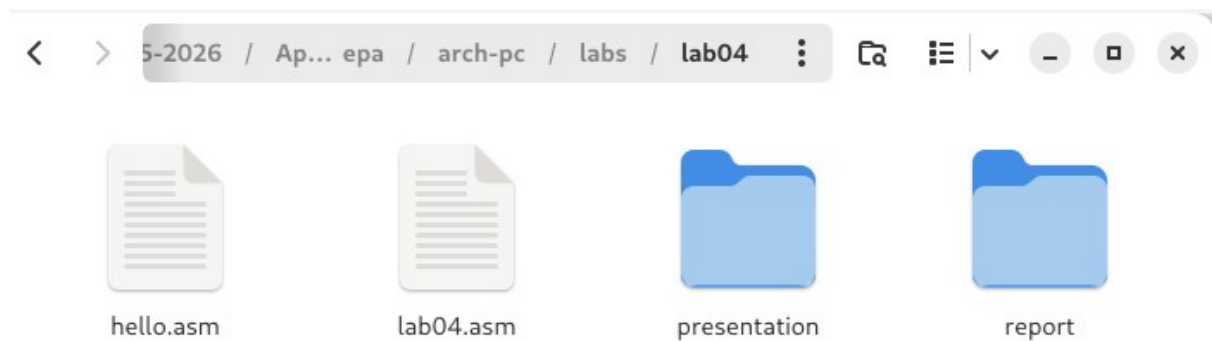


Рис.3.4

Файлы на Github загрузил

Вывод: Я выполнила задания по самостоятельной работе. Работа была выполнена и отправлена на Github.

5.Вывод

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

.