

Лабораторная работа №4

Создание и процесс обработки программ на языке ассемблера NASM

Четвергова Мария Викторовна

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	9
3.1	Компоновщик LD	11
3.2	Запуск исполняемого файла	12
3.3	Самостоятельная работа	13
4	Выводы	16

Список иллюстраций

3.1	Ввод данных на языке ассамблера NASM(1ч)	9
3.2	Ввод данных на языке ассамблера NASM(2ч)	10
3.3	компиляция приведённого текста программы	10
3.4	скомпилируем исходный файл hello.asm в obj.o	11
3.5	передаём программу на обработку компоновщику	11
3.6	исполняемый файл hello был создан	12
3.7	Запуск исполняемого файла	13
3.8	создадим копию файла hello.asm с именем lab4.asm	13
3.9	внесём изменения в текст программы в файле lab4.asm так, чтобы вместо Hello world! на экран выводилась другая строка	14
3.10	внесём изменения в текст программы в файле lab4.asm так, чтобы вместо Hello world! на экран выводилась другая строка	14
3.11	Запуск исполняемого файла	15
3.12	Загрузим файлы на Github	15

Список таблиц

1 Цель работы

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

2 Теоретическое введение

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 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.

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

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.

##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

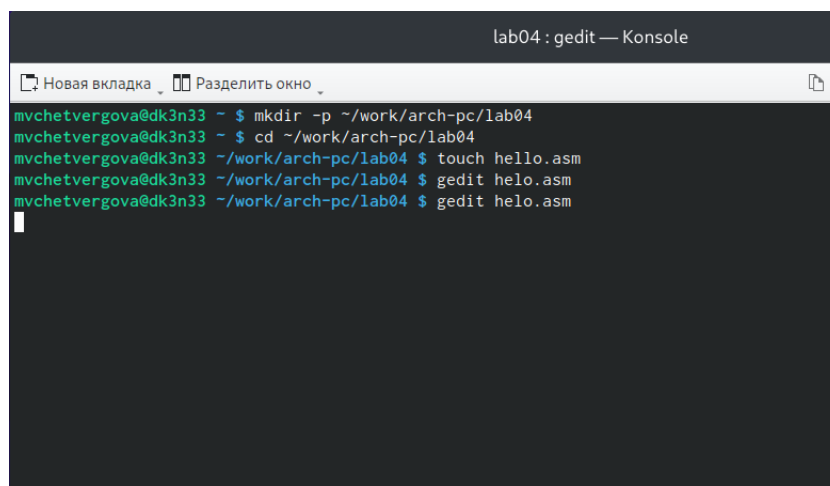
для человека образом пишутся команды для процессора. Следует отметить, что процессор понимает не команды ассемблера, а последовательности из нулей и единиц — машинные коды. До появления языков ассемблера программистам приходилось писать программы, используя только лишь машинные коды, которые были крайне 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.

3 Выполнение лабораторной работы

##Программа Hello world! Рассмотрим пример простой программы на языке ассемблера NASM. Традиционно первая программа выводит приветственное сообщение Hello world! на экран. Создайте каталог для работы с программами на языке ассемблера NASM. Перейдите в созданный каталог и создайте текстовый файл с именем hello.asm touch hello.asm откройте этот файл с помощью любого текстового редактора, например, gedit gedit hello.asm и введите в него следующий текст:

```
mvchetvergova@dk8n59 ~ $ cd ~/work/arch-pc/lab04
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ gedit hello.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ gedit hello.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ nasm -f elf hello.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ nasm -f elf hello.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ ls
hello.asm  hello.o  helo.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ gedit helo.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ nasm -o obj.o -f elf -g -l list.lst hello.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ ls
hello.asm  hello.o  helo.asm  list.lst  obj.o
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $
```

Рис. 3.1: Ввод данных на языке ассамблера NASM(1ч)



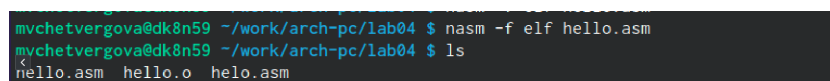
```
lab04: gedit — Konsole
mvchetvergova@dk3n33 ~ $ mkdir -p ~/work/arch-pc/lab04
mvchetvergova@dk3n33 ~ $ cd ~/work/arch-pc/lab04
mvchetvergova@dk3n33 ~/work/arch-pc/lab04 $ touch hello.asm
mvchetvergova@dk3n33 ~/work/arch-pc/lab04 $ gedit hello.asm
mvchetvergova@dk3n33 ~/work/arch-pc/lab04 $ gedit hello.asm
```

Рис. 3.2: Ввод данных на языке ассамблера NASM(2ч)

В отличие от многих современных высокоуровневых языков программирования, в ассемблерной программе каждая команда располагается на отдельной строке. Размещение нескольких команд на одной строке недопустимо. Синтаксис ассемблера NASM является чувствительным к регистру, т.е. есть разница между большими и малыми буквами.

##Транслятор NASM

NASM превращает текст программы в объектный код. Например, для компиляции приведённого выше текста программы «Hello World» необходимо написать:



```
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ nasm -f elf hello.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ ls
hello.asm  hello.o  hello.asm
```

Рис. 3.3: компиляция приведённого текста программы

Если текст программы набран без ошибок, то транслятор преобразует текст программы из файла hello.asm в объектный код, который запишется в файл hello.o. Таким образом, имена всех файлов получаются из имени входного файла и расширения по умолчанию. При наличии ошибок объектный файл не создаётся, а после запуска транслятора появятся сообщения об ошибках или предупреждения. С помощью команды ls проверьте, что объектный файл был создан. Какое имя

имеет объектный файл. NASM не запускают без параметров. Ключ -f указывает транслятору, что требуется создать бинарные файлы в формате ELF. Следует отметить, что формат elf64 позволяет создавать исполняемый код, работающий под 64-битными версиями Linux. Для 32-битных версий ОС указываем в качестве формата просто elf. NASM всегда создаёт выходные файлы в текущем каталоге.

##Расширенный синтаксис командной строки NASM

Выполним следующую команду:

```
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ nasm -o obj.o -f elf -g -l list.lst hello.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ ls
hello.asm  hello.o  helo.asm  list.lst  obj.o
```

Рис. 3.4: скомпилируем исходный файл hello.asm в obj.o

Данная команда скомпилирует исходный файл hello.asm в obj.o (опция -o позволяет задать имя объектного файла, в данном случае obj.o), при этом формат выходного файла будет elf, и в него будут включены символы для отладки (опция -g), кроме того, будет создан файл листинга list.lst (опция -l). С помощью команды ls проверьте, что файлы были созданы. Для более подробной информации см. man nasm. Для получения списка форматов объектного файла см. nasm -hf.

3.1 Компоновщик LD

чтобы получить исполняемую программу, объектный файл необходимо передать на обработку компоновщику:

```
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ nasm -o obj.o -f elf -g -l list.lst hello.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ ls
hello.asm  hello.o  helo.asm  list.lst  obj.o
```

Рис. 3.5: передаём программу на обработку компоновщику

С помощью команды ls проверьте, что исполняемый файл hello был создан. Компоновщик ld не предполагает по умолчанию расширений для файлов, но принято использовать следующие расширения: • o – для объектных файлов; • без

расширения – для исполняемых файлов; • **тар** – для файлов схемы программы;

- **lib** – для библиотек. Ключ **-o** с последующим значением задаёт в данном случае имя создаваемого исполняемого файла. Выполните следующую команду:

```
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ gedit hello.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ nasm -o obj.o -f elf -g -l list.lst hello.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ ls
hello.asm  hello.o  hello.asm  list.lst  obj.o
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ ld -m elf_i386 hello.o -o hello
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ ls
hello  hello.asm  hello.o  hello.asm  list.lst  obj.o
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ ld -m elf_i386 obj.o -o main
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ ld --help
Использование ld [параметры] файл...
Параметры:
-a КЛЮЧЕВОЕ СЛОВО                Управление общей библиотекой для совместимости с HP/UX
-A АРХИТЕКТУРА, --architecture АРХИТЕКТУРА    Задать архитектуру
-b ЦЕЛЬ, --format ЦЕЛЬ            Задать цель для следующих входных файлов
-c ФАЙЛ, --mri-script ФАЙЛ        Прочитать сценарий компоновщика в формате MRI
-d, -dc, -dp                      Принудительно делать общие символы определёнными
--dependency-file ФАЙЛ            Write dependency file
--force-group-allocation          Принудительно удалить членов группы из групп
-e АДРЕС, --entry АДРЕС          Задать начальный адрес
-E, --export-dynamic              Экспортировать все динамические символы
--no-export-dynamic              Отменить действие --export-dynamic
--enable-non-contiguous-regions  Enable support of non-contiguous memory regions
--enable-non-contiguous-regions-warnings Enable warnings when --enable-non-contiguous-regions may cause unex
-EV                               Компоновать объекты с прямым порядком байтов
-EL                               Компоновать объекты с обратным порядком байтов
-f SHLIB, --auxiliary SHLIB       Вспомогательный фильтр таблицы символов общих объектов
-F SHLIB, --filter SHLIB          Фильтр для таблицы символов общих объектов
-g                               Игнорируется
-G РАЗМЕР, --gpsize РАЗМЕР        Размер маленьких данных (если не указан, то берётся из --shared)
```

Рис. 3.6: исполняемый файл hello был создан

Какое имя будет иметь исполняемый файл? Какое имя имеет объектный файл из которого собран этот исполняемый файл? Формат командной строки LD можно увидеть, набрав **ld –help**. Для получения более подробной информации.

3.2 Запуск исполняемого файла

Запустить на выполнение созданный исполняемый файл, находящийся в текущем каталоге, можно, набрав в командной строке:

```
-z indirect-extern-access    Enable indirect external access
-z noindirect-extern-access  Disable indirect external access (default)
-z dynamic-undefined-weak    делать неопределённые слабые символы
                             динамическими
-z nodynamic-undefined-weak  не делать неопределённые слабые символы
                             динамическими
-z call-nop=ЗАПОЛНЕНИЕ       использовать ЗАПОЛНЕНИЕ в качестве
                             1-байтового NOP для ветвления

Сообщения об ошибках отправляйте в <https://bugs.gentoo.org/>
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ ./hello
hello world!
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $
```

Рис. 3.7: Запуск исполняемого файла

3.3 Самостоятельная работа

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

```
mvchetvergova@dk8n59 ~ $
mvchetvergova@dk8n59 ~ $
mvchetvergova@dk8n59 ~ $
mvchetvergova@dk8n59 ~ $ cd ~/work/arch-pc/lab04
bash: /afs/.dk.sci.pfu.edu.ru/home/m/v/mvchetvergova/work/arch-pc/lab04: Это каталог
mvchetvergova@dk8n59 ~ $ cd ~/work/arch-pc/lab04
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ cp hello.asm lab4.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ ls
hello  hello.asm  hello.o  helo.asm  lab4.asm  list.lst  main  obj.o
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $
```

Рис. 3.8: создадим копию файла `hello.asm` с именем `lab4.asm`

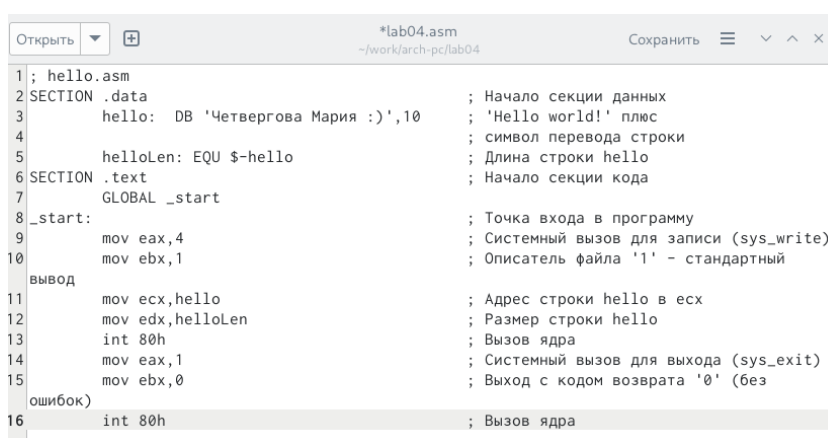
2. С помощью любого текстового редактора внесите изменения в текст программы в файле `lab4.asm` так, чтобы вместо `Hello world!` на экран выводилась строка с вашими фамилией и именем.

```

mvchetvergova@dk8n59 ~ $
mvchetvergova@dk8n59 ~ $
mvchetvergova@dk8n59 ~ $ ~/work/arch-pc/lab04
bash: /afs/.dk.sci.pfu.edu.ru/home/m/v/mvchetvergova/work/arch-pc/lab04: Это каталог
mvchetvergova@dk8n59 ~ $ cd ~/work/arch-pc/lab04
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ cp hello.asm lab4.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ ls
hello hello.asm hello.o helo.asm lab4.asm list.lst main obj.o
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ gedit lab04.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ gedit helo.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ gedit hello.asm
mvchetvergova@dk8n59 ~/work/arch-pc/lab04 $ gedit lab04.asm

```

Рис. 3.9: внесём изменения в текст программы в файле lab4.asm так, чтобы вместо Hello world! на экран выводилась другая строка



```

1; hello.asm
2SECTION .data
3    hello: DB 'Четвергова Мария :)',10
4
5    helloLen: EQU $-hello
6SECTION .text
7    GLOBAL _start
8_start:
9    mov eax,4
10   mov ebx,1
11   вывод
12   mov ecx,hello
13   mov edx,helloLen
14   int 80h
15   mov eax,1
16   mov ebx,0
17   int 80h

```

Рис. 3.10: внесём изменения в текст программы в файле lab4.asm так, чтобы вместо Hello world! на экран выводилась другая строка

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

```

mvchettergova@dk8n59 ~/work/arch-pc/lab04 $
mvchettergova@dk8n59 ~/work/arch-pc/lab04 $ nasm -f elf lab04.asm
mvchettergova@dk8n59 ~/work/arch-pc/lab04 $ ls
hello hello.asm hello.o hello.asm lab04.asm lab04.o lab4.asm list.lst main obj.o
mvchettergova@dk8n59 ~/work/arch-pc/lab04 $ nasm -o obj.o -f elf -g -l list.lst lab04.asm
mvchettergova@dk8n59 ~/work/arch-pc/lab04 $ ls
hello hello.asm hello.o hello.asm lab04.asm lab04.o lab4.asm list.lst main obj.o
mvchettergova@dk8n59 ~/work/arch-pc/lab04 $ ld -m elf_i386 lab04.o -o lab04
mvchettergova@dk8n59 ~/work/arch-pc/lab04 $ ls
hello hello.asm hello.o hello.asm lab04.asm lab04.o lab4.asm list.lst main obj.o
mvchettergova@dk8n59 ~/work/arch-pc/lab04 $ ld -m elf_i386 obj.o main
ld: main: в функции «_start»:
/afs/.dk.sci.pfu.edu.ru/home/m/v/mvchettergova/work/arch-pc/lab04//hello.asm:9: повторное определение «_start
»; obj.o:/afs/.dk.sci.pfu.edu.ru/home/m/v/mvchettergova/work/arch-pc/lab04//lab04.asm:9: здесь первое определ
ение
ld: cannot use executable file 'main' as input to a link
mvchettergova@dk8n59 ~/work/arch-pc/lab04 $
mvchettergova@dk8n59 ~/work/arch-pc/lab04 $ ./lab04
Четвергова Мария :)
mvchettergova@dk8n59 ~/work/arch-pc/lab04 $

```

Рис. 3.11: Запуск исполняемого файла

4. Скопируйте файлы hello.asm и lab4.asm в Ваш локальный репозиторий в каталог `~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/`. Загрузите файлы на Github

```

mvchettergova@dk8n59 ~/work/arch-pc/lab04 $
mvchettergova@dk8n59 ~/work/arch-pc/lab04 $ cd ~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04
mvchettergova@dk8n59 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ git add .
mvchettergova@dk8n59 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ git commit -am
error: switch 'm' requires a value
mvchettergova@dk8n59 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ git push
Everything up-to-date
mvchettergova@dk8n59 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $

```

Рис. 3.12: Загрузим файлы на Github

4 Выводы

В ходе выполнения лабораторной работы №4 мы освоили процедуры компиляции и сборки программ, написанных на ассемблере NASM.