Отчёта по лабораторной работе №4

Дисциплина: архитектура компьютера

Хасанов Тимур

Содержание

Список иллюстраций

Цель работы

Цель данной лабораторной работы - освоить процедуры компиляции и сборки программ, написанных на ассемблере NASM.

Задание

1. Создание программы Hello world!
2. Работа с транслятором NASM
3. Работа с расширенным синтаксисом командной строки NASM
4. Работа с компоновщиком LD
5. Запуск исполняемого файла
6. Выполнение заданий для самостоятельной работы.

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 каче- стве 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.

Выполнение лабораторной работы

Создание программы Hello world!

С помощью утилиты cd перемещаюсь в каталог, в котором буду работать (рис. 1).

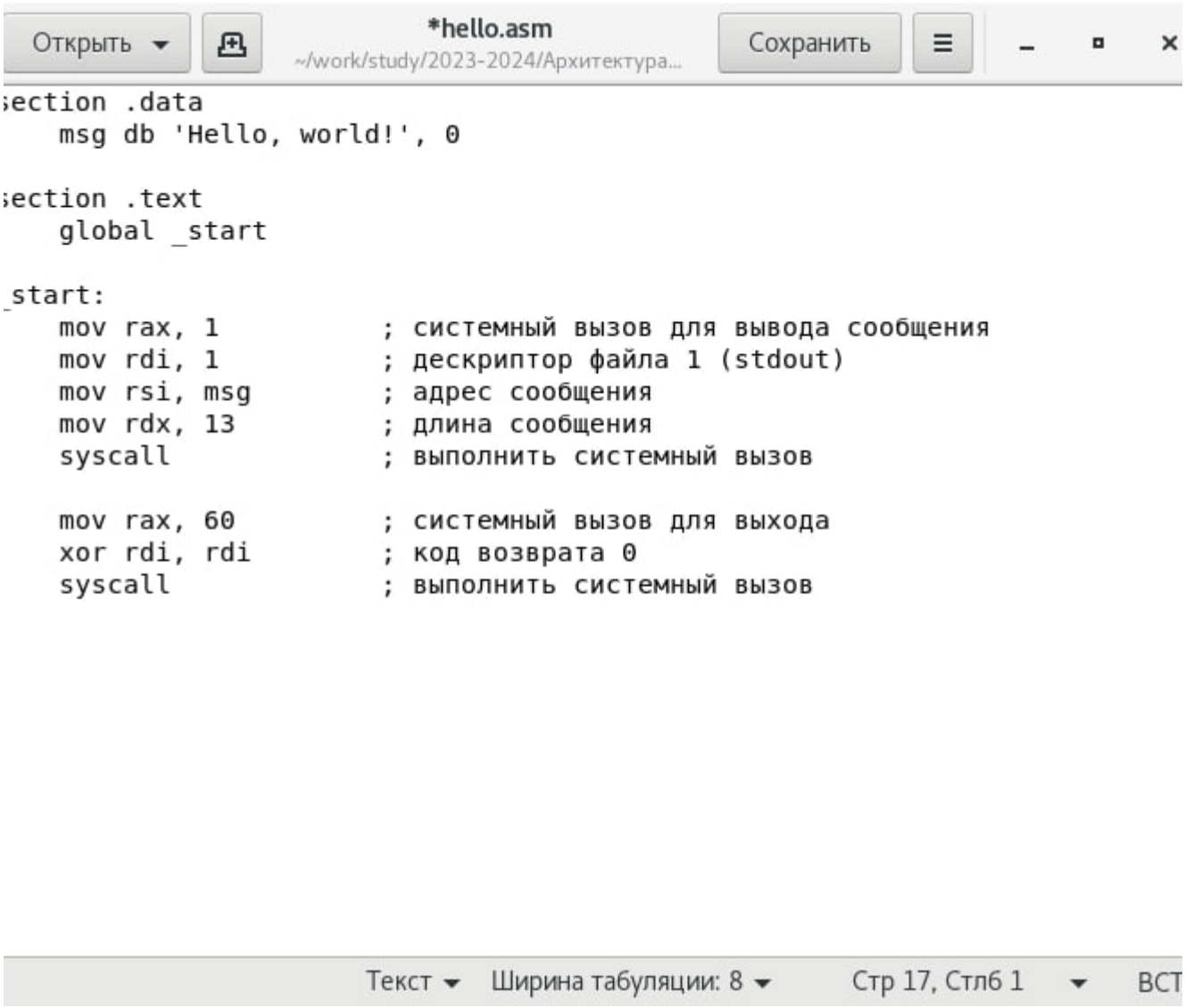
```
tihasanov@tihasanov ~]$ cd work/study/2023-2024/"Архитектура компьютера"/
rch-pc/labs/lab04
tihasanov@tihasanov lab04]$
```

Создаю в текущем каталоге пустой текстовый файл hello.asm с помощью утилиты touch (рис. 2).

```
tihasanov@tihasanov lab04]$ touch hello.asm
tihasanov@tihasanov lab04]$
```

Открываю созданный файл в текстовом редакторе (рис. 3).

Заполняю файл, вставляя в него программу для вывода "Hello word!" (рис. 4).



Работа с транслятором NASM

Превращаю текст программы для вывода "Hello world!" в объектный код с помощью транслятора NASM, используя команду nasm -f elf hello.asm, ключ -f указывает транслятору nasm, что требуется создать бинарный файл в формате ELF (рис. 5). Далее проверяю правильность выполнения команды с помощью утилиты ls: действительно, создан файл "hello.o".

```
[tihasanov@tihasanov lab04]$ nasm -f elf64 hello.asm
[tihasanov@tihasanov lab04]$ ls
hello.asm  hello.o  presentation  report
[tihasanov@tihasanov lab04]$
```

Работа с расширенным синтаксисом командной строки NASM

Ввожу команду, которая скомпилирует файл hello.asm в файл obj.o, при этом в файл будут включены символы для отладки (ключ -g), также с помощью ключа -l будет создан файл листинга list.lst (рис. 6). Далее проверяю с помощью утилиты ls правильность выполнения команды.

```
[tihasanov@tihasanov lab04]$ nasm -f elf64 -g -l list.lst hello.asm
[tihasanov@tihasanov lab04]$ ls
hello.asm  hello.o  list.lst  presentation  report
```

Работа с компоновщиком LD

Передаю объектный файл hello.o на обработку компоновщику LD, чтобы получить исполняемый файл hello (рис. 7). Ключ -o задает имя создаваемого исполняемого файла. Далее проверяю с помощью утилиты ls правильность выполнения команды.

```
[tihasanov@tihasanov lab04]$ ld -o hello hello.o
[tihasanov@tihasanov lab04]$ ls
hello  hello.asm  hello.o  list.lst  presentation  report
```

Выполняю следующую команду (рис. 8). Исполняемый файл будет иметь имя main, т.к. после ключа -o было задано значение main. Объектный файл, из которого собран этот исполняемый файл, имеет имя obj.o

```
[tihasanov@tihasanov lab04]$ ld -o main obj.o
```

Запуск исполняемого файла

Запускаю на выполнение созданный исполняемый файл hello (рис. 9).

```
[tihasanov@tihasanov lab04]$ ./hello
Hello, world![tihasanov@tihasanov lab04]$
```

Выполнение заданий для самостоятельной работы.

С помощью утилиты cp создаю в текущем каталоге копию файла hello.asm с именем lab4.asm (рис. 10).

```
Hello, world![tihasanov@tihasanov lab04]$ cp hello.asm lab4.asm
[tihasanov@tihasanov lab04]$ ls
hello      hello.o   list.lst  obj.o         report
hello.asm  lab4.asm  obj.asm   presentation
```

С помощью текстового редактора открываю файл lab4.asm и вношу изменения в программу так, чтобы она выводила мои имя и фамилию. (рис. 11).

```
section .data
    msg db 'Hasanov Timur', 0

section .text
    global _start

_start:
    mov rax, 1          ; системный вызов для вывода сообщения
    mov rdi, 1          ; дескриптор файла 1 (stdout)
    mov rsi, msg        ; адрес сообщения
    mov rdx, 13         ; длина сообщения
    syscall             ; выполнить системный вызов

    mov rax, 60         ; системный вызов для выхода
    xor rdi, rdi        ; код возврата 0
    syscall             ; выполнить системный вызов
```

Компилирую текст программы в объектный файл (рис. 12). Проверяю с помощью утилиты ls, что файл lab4.o создан.

```
[tihasanov@tihasanov lab04]$ nasm -f elf64 -g -l lab4.lst lab4.asm
[tihasanov@tihasanov lab04]$ ls
hello      hello.o   lab4.lst  list.lst  obj.o         report
hello.asm  lab4.asm  lab4.o    obj.asm   presentation
```

Передаю объектный файл lab4.o на обработку компоновщику LD, чтобы получить исполняемый файл lab4 (рис. 13).

```
[tihasanov@tihasanov lab04]$ ld -o lab4 lab4.o
[tihasanov@tihasanov lab04]$ ls
hello      hello.o  lab4.asm  lab4.o    obj.asm  presentation
hello.asm  lab4     lab4.lst  list.lst  obj.o    report
```

Запускаю исполняемый файл lab4, на экран действительно выводятся мои имя и фамилия (рис. 14).

```
[tihasanov@tihasanov lab04]$ ./lab4
Hasanov Timur[tihasanov@tihasanov lab04]$
```

К сожалению, я начал работу не в том каталоге, поэтому создаю другую директорию lab04 с помощью mkdir, прописывая полный путь к каталогу, в котором хочу создать эту директорию. Далее копирую из текущего каталога файлы, созданные в процессе выполнения лабораторной работы, с помощью утилиты cp, указывая вместо имени файла символ *, чтобы скопировать все файлы. Команда проигнорирует директории в этом каталоге, т. к. не указан ключ -r, это мне и нужно (рис. 15). Проверяю с помощью утилиты ls правильность выполнения команды.

```
[tihasanov@tihasanov lab04]$ cp * lab04/
cp: пропускается каталог «lab04»
cp: пропускается каталог «presentation»
cp: пропускается каталог «report»
[tihasanov@tihasanov lab04]$ ls lab04/
hello      hello.o  lab4.asm  lab4.o    obj.asm
hello.asm  lab4     lab4.lst  list.lst  obj.o
[tihasanov@tihasanov lab04]$
```

Удаляю лишние файлы в текущем каталоге с помощью утилиты rm, ведь копии файлов остались в другой директории (рис. 16).

```
[tihasanov@tihasanov lab04]$ rm hello hello.o lab4.o obj.asm lab4 lab4.lst
 list.lst obj.o
[tihasanov@tihasanov lab04]$ ls
hello.asm  lab4.asm
```

С помощью команд git add . и git commit добавляю файлы на GitHub, комментируя действие как добавление файлов для лабораторной работы №5 (рис. 17).

```
[tihasanov@tihasanov lab04]$ git add .
[tihasanov@tihasanov lab04]$ git commit -m "lab04"
[master 68ac3d0] lab04
 2 files changed, 38 insertions(+)
 create mode 100644 labs/lab04/lab04/hello.asm
 create mode 100644 labs/lab04/lab04/lab4.asm
[tihasanov@tihasanov lab04]$
```

Отправляю файлы на сервер с помощью команды git push (рис. 18).

```
[tihasanov@tihasanov lab04]$ git push
```

Выводы

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

Список литературы

1. https://esystem.rudn.ru/pluginfile.php/1584628/mod_resource/content/1/%D0%9B%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%20%E2%84%965.pdf