

Отчет по лабораторной работе №4

Дисциплина: Архитектура компьютера

Первий Анастасия Андреевна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	9
4.1	Создание программы Hello World!	9
4.2	Работа с транслятором NASM	10
4.3	Работа с расширенным синтаксисом командной строки NASM . .	11
4.4	Работа с компоновщиком LD	11
4.5	Запуск исполняемого файла	12
4.6	Выполнение заданий для самостоятельной работы	12
5	Выводы	15
	Список литературы	16

Список иллюстраций

4.1	Создание каталога	9
4.2	Создание текстового файла hello.asm	9
4.3	Открытие файла	9
4.4	Ввод текста программы	10
4.5	Попытка компиляции текста	10
4.6	Компилирую текст	11
4.7	Компилирую файл	11
4.8	Передача объектного файла на обработку компоновщику LD . . .	11
4.9	Запуск исполняемого файла	12
4.10	Копирую файл hello.asm с именем lab04.asm	12
4.11	Название рисунка	13
4.12	Компилирую текст программы в объектный файл и провожу проверку	13
4.13	Копирую файлы	13
4.14	Загружаю файлы на Github	14
4.15	Загружаю файлы на Github	14

Список таблиц

1 Цель работы

Целью данной работы является приобретение практического опыта работы с программами, написанными на ассемблере NASM, а именно - освоение процедур компиляций и сборки.

2 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

3 Теоретическое введение

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. Но 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. Программы, написанные на языке ассемблера, не уступают в качестве и скорости программам, написанным на машинном языке, ибо 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: 1) Для DOS/Windows: Borland Turbo Assembler (TASM), Microsoft Macro Assembler (MASM) и Watcom assembler (WASM). 2) Для GNU/Linux: gas (GNU Assembler), использующий AT&T-синтаксис, в отличие от большинства других популярных ассемблеров, которые используют Intel-синтаксис. Для записи команд в NASM используются: 1) Мнемокод — непосредственно мнемоника инструкции процессору, которая является обязательной частью команды. 2) Операнды — числа, данные, адреса регистров или адреса оперативной памяти. 3) Метка — идентификатор, с которым ассемблер ассоциирует некоторое число, чаще всего адрес в памяти. (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 составляет 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.

4 Выполнение лабораторной работы

4.1 Создание программы Hello World!

Для создания простой программы, которая выводит сообщение **Hello world!**, необходимо создать каталог, в котором дальше будем работать с программами на языке ассемблере NASM. Для этого в терминале необходимо прописать команду **mkdir -p ~/work/arch-pc/lab04** (Рис. 4.1)

```
aaпerviyj@aaпerviyj--NPI-03-23:~$ mkdir -p ~/work/arch-pc/lab04
aaпerviyj@aaпerviyj--NPI-03-23:~$ cd ~/work/arch-pc/lab04
```

Рис. 4.1: Создание каталога

Далее создаю текстовый файл **hello.asm** (Рис. 4.2)

```
aaпerviyj@aaпerviyj--NPI-03-23:~/work/arch-pc/lab04$ touch hello.asm
aaпerviyj@aaпerviyj--NPI-03-23:~/work/arch-pc/lab04$ ls
hello.asm
```

Рис. 4.2: Создание текстового файла hello.asm

Открываем файл для дальнейшего ввода теста команды (Рис. 4.3)

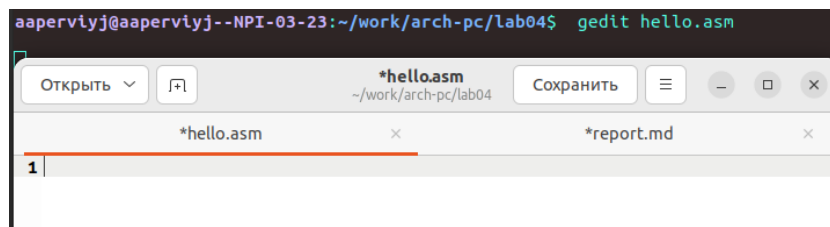


Рис. 4.3: Открытие файла

Ввожу текст программы (Рис. 4.4)

```
*hello.asm
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 mov eax,1 ; Системный вызов для выхода (sys_exit)
15 mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
16 int 80h ; Вызов ядра
```

Рис. 4.4: Ввод текста программы

4.2 Работа с транслятором NASM

Чтобы команда заработала, необходимо скомпилировать приведенный текст, для этого нужно прописать команду **nasm -f elf hello.asm** Выдало ошибку, так как у меня не было необходимой утилиты, поэтому я установила утилиту nasm (Рис. 4.5)

```
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$ nasm -f elf hello.asm
Команда «nasm» не найдена, но может быть установлена с помощью:
sudo apt install nasm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$ sudo apt install nasm
[sudo] пароль для aaperviyj:
Чтение списков пакетов... Готово
Построение дерева зависимостей... Готово
Чтение информации о состоянии... Готово
Следующие НОВЫЕ пакеты будут установлены:
  nasm
```

Рис. 4.5: Попытка компиляции текста

После установки необходимых утилит, пробую прописать команду **nasm -f elf hello.asm** еще раз (Рис. 4.6)

```

aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$ nasm -f elf hello.asm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$ ls
hello.asm  hello.o
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$

```

Рис. 4.6: Компилирую текст

4.3 Работа с расширенным синтаксисом командной строки NASM

Теперь необходимо скомпилировать файл **hello.asm** в файл **obj.o**. При компиляции будут включены символы для отладки: ключ **-g**, также с помощью ключа **-l** будет создан файл листинга **list.lst**. Проверяю правильность выполнения команды (рис. 4.7)

```

aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$ nasm -o obj.o -f elf -g -l list.lst
t hello.asm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$ ls
hello.asm  hello.o  list.lst  obj.o
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$

```

Рис. 4.7: Компилирую файл

4.4 Работа с компоновщиком LD

Чтобы получить исполняемую команду, необходимо передать объектный файл на обработку компоновщику LD. Передаю файлы **hello.o** и **obj.o** (рис. 4.8)

```

aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$ ld -m elf_i386 hello.o -o hello
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  list.lst  obj.o
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$ ld -m elf_i386 obj.o -o main
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  list.lst  main  obj.o

```

Рис. 4.8: Передача объектного файла на обработку компоновщику LD

4.5 Запуск исполняемого файла

Теперь можно запустить созданный исполняемый файл **hello**. Для этого ввожу команду в текущем каталоге **./hello** (рис. 4.9)

```
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$ ./hello
Hello world!
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$
```

Рис. 4.9: Запуск исполняемого файла

4.6 Выполнение заданий для самостоятельной работы

1. В каталоге **~/work/arch-pc/lab04** с помощью команды **cp** создаю копию файла **hello.asm** с именем **lab04.asm** (рис. 4.10)

```
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$ cp hello.asm lab04.asm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  lab04.asm  list.lst  main  obj.o
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$
```

Рис. 4.10: Копирую файл **hello.asm** с именем **lab04.asm**

2. С помощью текстового редактора вношу изменения в текст программы в файле **lab4.asm** так, чтобы вместо **Hello world!** на экран выводилась строка с моими фамилией и именем (рис. 4.11)

```

aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$ gedit lab04.asm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$

```

```

1; hello.asm
2 SECTION .data ; Начало секции данных
3 hello: DB 'Perviy Anastasia',10 ; 'Perviy Anastasia' 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 mov eax,1 ; Системный вызов для выхода (sys_exit)
15 mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
16 int 80h ; Вызов ядра

```

Рис. 4.11: Название рисунка

- Оттранслирую полученный текст программы **lab4.asm** в объектный файл. Выполняю компоновку объектного файла и запускаю получившийся исполняемый файл (рис. 4.12)

```

aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$ nasm -f elf lab04.asm
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$ ld -m elf_i386 lab04.o -o lab04
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$ ./lab04
Perviy Anastasia
aaperviyj@aaperviyj--NPI-03-23:~/work/arch-pc/lab04$

```

Рис. 4.12: Компилирую текст программы в объектный файл и провожу проверку

- Копирую файлы **hello.asm** и **lab4.asm** в свой локальный репозиторий в каталог `~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/` (рис. 4.13)

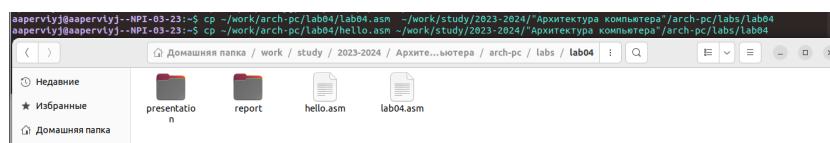


Рис. 4.13: Копирую файлы

Загружаю файлы на Github (рис. 4.14) (рис. 4.15)

```

aaperviyj@aaperviyj--NPI-03-23:~/work/study/2023-2024/Архитектура компьютера/arch-pc$ git add .
aaperviyj@aaperviyj--NPI-03-23:~/work/study/2023-2024/Архитектура компьютера/arch-pc$ git commit -a
viyj@aaperviyj--NPI-03-23:~/work/study/2023-2024/Архитектура компьютера/aaperviyj@aaperviyj--NPI-03-
PI-03-23:~/work/study/2023-2024/Архитектура компьютера/aaperviyj@aaperviyj--NPI-03-23:~/work/study
24/Архитектура компьютера/arch-pc$ git commit -am 'feat(main):make course structure'
[master 8f44f2e] feat(main):make course structure
44 files changed, 212 insertions(+), 145 deletions(-)
create mode 100644 labs/lab02/report/image/001.jpeg
create mode 100644 labs/lab02/report/image/002.jpeg
create mode 100644 labs/lab02/report/image/003.jpeg
create mode 100644 labs/lab02/report/image/004.jpeg
create mode 100644 labs/lab02/report/image/005.jpeg

```

Рис. 4.14: Загружаю файлы на Github

```

aaperviyj@aaperviyj--NPI-03-23:~/work/study/2023-2024/Архитектура компьютера/arch-pc$ git push
Перечисление объектов: 65, готово.
Подсчет объектов: 100% (62/62), готово.
При сжатии изменений используется до 7 потоков
Сжатие объектов: 100% (54/54), готово.
Запись объектов: 100% (54/54), 1.75 Миб | 2.61 Миб/с, готово.
Всего 54 (изменений 8), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (8/8), completed with 4 local objects.
To github.com:aaperviyj-NPI-03-23/study_2023-2024_arh-pc.git
 d29a0d8..8f44f2e master -> master
aaperviyj@aaperviyj--NPI-03-23:~/work/study/2023-2024/Архитектура компьютера/arch-pc$

```

Рис. 4.15: Загружаю файлы на Github

5 Выводы

Во время выполнения лабораторной работы я обрела практический опыт работы с программами, написанными на языке ассемблере NASM. Освоила процедуры компиляций и сборки.

Список литературы

Архитектура ЭВМ :::