

Отчёт по лабораторной работе №4

Дисциплина: архитектура компьютера

Ахмаров Роман

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	10
4.1	Работа с транслятором NASM	11
4.2	Работа с расширенным синтаксисом командной строки NASM . .	12
4.3	Работа с компоновщиком LD	12
4.4	Запуск исполняемого файла	13
4.5	Выполнение заданий для самостоятельной работы.	13
5	Выводы	17
5.1	Список литературы	17

Список иллюстраций

4.1	Перемещение между директориями	10
4.2	Создание пустого файла	10
4.3	Открытие файла в текстовом редакторе	10
4.4	Заполнение файла	11
4.5	Компиляция текста программы	11
4.6	создан файл листинга list.lst	12
4.7	Передача объектного файла на обработку компоновщику	12
4.8	Передача объектного файла на обработку компоновщику	13
4.9	Запуск исполняемого файла	13
4.10	Создание копии файла	13
4.11	Изменение программы	14
4.12	Компиляция текста программы	14
4.13	Передача объектного файла на обработку компоновщику	14
4.14	Запуск исполняемого файла	15
4.15	Добавление файлов на GitHub	16

Список таблиц

1 Цель работы

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

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

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 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. 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.

4 Выполнение лабораторной работы

С помощью утилиты `cd` перемещаюсь в каталог, в котором буду работать (рис. 4.1).

```
[rrakhmarov@fedora ~]$ cd ~/work/study/2023-2024/"Computer architecture"/arch-pc/labs/lab04/report  
[rrakhmarov@fedora report]$ cp report.md L04_Akhmarov_Roman.md
```

Рис. 4.1: Перемещение между директориями

Создаю в текущем каталоге пустой текстовый файл `hello.asm` с помощью утилиты `touch` (рис. 4.2).

```
[rrakhmarov@fedora report]$ touch hello.asm
```

Рис. 4.2: Создание пустого файла

Открываю созданный файл в текстовом редакторе `mousepad` (рис. 4.3).

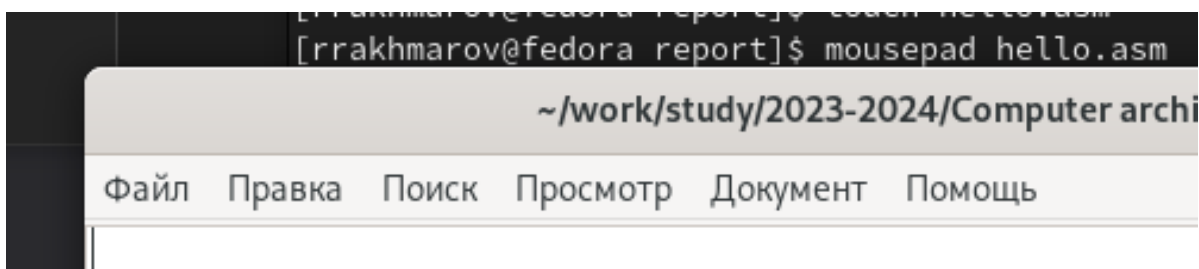
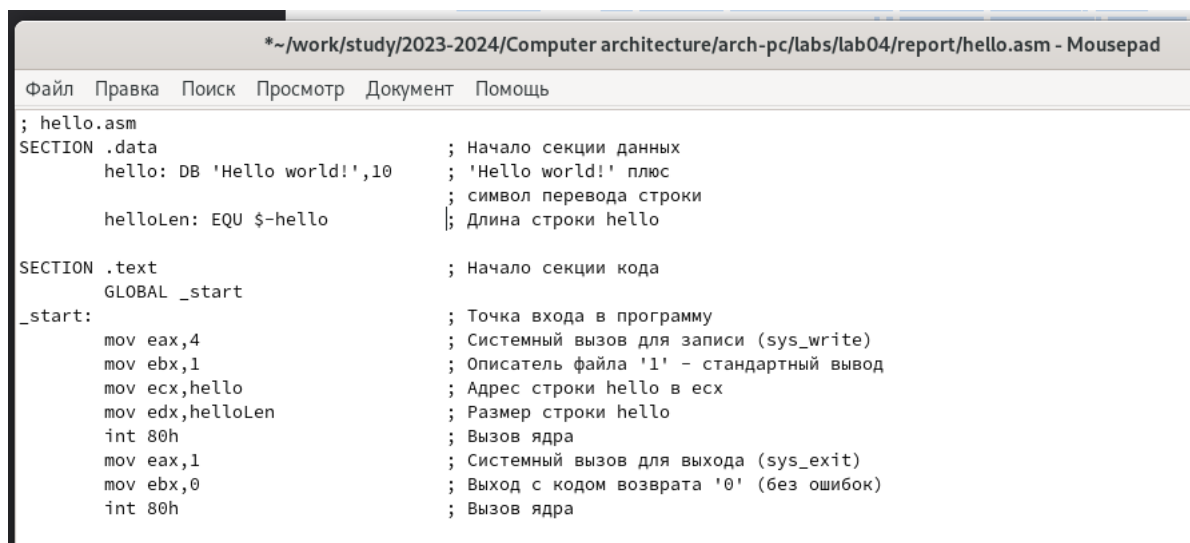


Рис. 4.3: Открытие файла в текстовом редакторе

Заполняю файл, вставляя в него программу для вывода “Hello word!”. Так, как ассемблер не является высокоуровневым языком, каждая команда размещается

на отдельной строке, так же обращаю внимание на регистр, так как Assembly чувствителен к нему (рис. 4.4).



```
*~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04/report/hello.asm - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь

; hello.asm
SECTION .data                                ; Начало секции данных
    hello: DB 'Hello world!',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
    int 80h                                 ; Вызов ядра
    mov eax,1                               ; Системный вызов для выхода (sys_exit)
    mov ebx,0                               ; Выход с кодом возврата '0' (без ошибок)
    int 80h                                 ; Вызов ядра
```

Рис. 4.4: Заполнение файла

4.1 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. 4.5). Далее проверяю правильность выполнения команды с помощью утилиты `ls`: действительно, создан файл “hello.o”.



```
[rrakhmarov@fedora report]$ nasm -f elf hello.asm
[rrakhmarov@fedora report]$ ls
b1b hello.asm  hello.o  image  L04_Akhmarov_Roman.docx  L04_Akhmarov_Roman.md  L04_Akhmarov_Roman.pdf  Makefile  pandoc
```

Рис. 4.5: Компиляция текста программы

4.2 Работа с расширенным синтаксисом командной строки NASM

Ввожу команду, которая скомпилирует файл `hello.asm` в файл `obj.o`, используя ключ `-o` который задает имя объектному файлу, так же в файл будут включены символы для отладки (ключ `-g`), с помощью ключа `-l` будет создан файл листинга `list.lst` (рис. 4.6). Далее проверяю с помощью утилиты `ls` правильность выполнения команды.

```
[rrakhmarov@fedora report]$ nasm -o obj.o -f elf -g -l list.lst hello.asm
[rrakhmarov@fedora report]$ ls
bib      hello.o    L04_Akhmarov_Roman.docx  L04_Akhmarov_Roman.pdf  Makefile  pandoc
hello.asm image      L04_Akhmarov_Roman.md    list.lst                 obj.o
```

Рис. 4.6: создан файл листинга `list.lst`

4.3 Работа с компоновщиком LD

Передаю объектный файл `hello.o` на обработку компоновщику `LD`, чтобы получить исполняемый файл `hello` (рис. 4.7). Ключ `-o` задает имя создаваемого исполняемого файла. Далее проверяю с помощью утилиты `ls` правильность выполнения команды.

```
[rrakhmarov@fedora report]$ ld -m elf_i386 hello.o -o hello
[rrakhmarov@fedora report]$ ls
bib      hello.asm  image      L04_Akhmarov_Roman.md  list.lst  obj.o
hello    hello.o    L04_Akhmarov_Roman.docx  L04_Akhmarov_Roman.pdf  Makefile  pandoc
```

Рис. 4.7: Передача объектного файла на обработку компоновщику

Выполняю следующую команду (рис. 4.8). 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`

```
[rrakhmarov@fedora report]$ ld -m elf_i386 obj.o -o main
[rrakhmarov@fedora report]$ ls
bib    hello.asm  image      L04_Akhmarov_Roman.md  list.lst  Makefile  pandoc
hello  hello.o    L04_Akhmarov_Roman.docx L04_Akhmarov_Roman.pdf main      obj.o
```

Рис. 4.8: Передача объектного файла на обработку компоновщику

4.4 Запуск исполняемого файла

Запускаю на выполнение созданный исполняемый файл hello (рис. 4.9).

```
[rrakhmarov@fedora report]$ ./hello
Hello world!
```

Рис. 4.9: Запуск исполняемого файла

4.5 Выполнение заданий для самостоятельной работы.

С помощью утилиты `cp` создаю в текущем каталоге копию файла `hello.asm` с именем `lab4.asm` (рис. 4.10).

```
[rrakhmarov@fedora report]$ cp hello.asm lab4.asm
[rrakhmarov@fedora report]$ ls
bib    hello.asm  image      L04_Akhmarov_Roman.md  lab4.asm  main      obj.o
hello  hello.o    L04_Akhmarov_Roman.docx L04_Akhmarov_Roman.pdf list.lst  Makefile  pandoc
```

Рис. 4.10: Создание копии файла

С помощью текстового редактора `mousepad` открываю файл `lab4.asm` и вношу изменения в программу так, чтобы она выводила мои имя и фамилию. (рис. 4.11)

```

*~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04/report/lab4.asm - Mousepad
Файл  Правка  Поиск  Просмотр  Документ  Помощь
; hello.asm
SECTION .data                                ; Начало секции данных
    hello: DB 'Akhmarov Roman!',10          ; Мой текст 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
    int 80h                                 ; Вызов ядра
    mov eax,1                               ; Системный вызов для выхода (sys_exit)
    mov ebx,0                               ; Выход с кодом возврата '0' (без ошибок)
    int 80h                                 ; Вызов ядра

```

Рис. 4.11: Изменение программы

Компилирую текст программы в объектный файл (рис. 4.12). Проверяю с помощью утилиты `ls`, что файл `lab4.o` создан.

```

[rrakhmarov@fedora report]$ nasm -f elf lab4.asm
[rrakhmarov@fedora report]$ ls
bib  hello.asm  image  L04_Akhmarov_Roman.md  lab4.asm  list.lst  Makefile  pandoc
hello hello.o    L04_Akhmarov_Roman.docx  L04_Akhmarov_Roman.pdf  lab4.o    main      obj.o

```

Рис. 4.12: Компиляция текста программы

Передаю объектный файл `lab4.o` на обработку компоновщику `LD`, чтобы получить исполняемый файл `lab4` (рис. 4.13).

```

[rrakhmarov@fedora report]$ ld -m elf_i386 lab4.o -o lab4
[rrakhmarov@fedora report]$ ls
bib  hello.asm  image  L04_Akhmarov_Roman.md  lab4  lab4.o  main  obj.o
hello hello.o    L04_Akhmarov_Roman.docx  L04_Akhmarov_Roman.pdf  lab4.asm  list.lst  Makefile  pandoc

```

Рис. 4.13: Передача объектного файла на обработку компоновщику

Запускаю исполняемый файл `lab4`, на экран действительно выводятся мои имя и фамилия (рис. 4.14)

```
[rrakhmarov@fedora report]$ ./lab4  
Akhmarov Roman!
```

Рис. 4.14: Запуск исполняемого файла

С помощью команд `git add .` и `git commit` добавляю файлы на GitHub, комментируя действие как добавление файлов для лабораторной работы №4 (рис. 4.15)

```

[rrakhmarov@fedora arch-pc]$ git add .
[rrakhmarov@fedora arch-pc]$ git commit -m "Add files for lab04"
[master 3a2fadc] Add files for lab04
28 files changed, 179 insertions(+), 119 deletions(-)
create mode 100644 labs/lab04/report/L04_Akhmarov_Roman.docx
create mode 100644 labs/lab04/report/L04_Akhmarov_Roman.md
create mode 100644 labs/lab04/report/L04_Akhmarov_Roman.pdf
create mode 100755 labs/lab04/report/hello
create mode 100644 labs/lab04/report/hello.asm
create mode 100644 labs/lab04/report/hello.o
create mode 100644 labs/lab04/report/image/1.png
create mode 100644 labs/lab04/report/image/10.png
create mode 100644 labs/lab04/report/image/11.png
create mode 100644 labs/lab04/report/image/12.png
create mode 100644 labs/lab04/report/image/13.png
create mode 100644 labs/lab04/report/image/14.png
create mode 100644 labs/lab04/report/image/2.png
create mode 100644 labs/lab04/report/image/3.png
create mode 100644 labs/lab04/report/image/4.png
create mode 100644 labs/lab04/report/image/5.png
create mode 100644 labs/lab04/report/image/6.png
create mode 100644 labs/lab04/report/image/7.png
create mode 100644 labs/lab04/report/image/8.png
create mode 100644 labs/lab04/report/image/9.png
delete mode 100644 labs/lab04/report/image/placeimg_800_600_tech.jpg
create mode 100755 labs/lab04/report/lab4
create mode 100644 labs/lab04/report/lab4.asm
create mode 100644 labs/lab04/report/lab4.o
create mode 100644 labs/lab04/report/list.lst
create mode 100755 labs/lab04/report/main
create mode 100644 labs/lab04/report/obj.o
delete mode 100644 labs/lab04/report/report.md
[rrakhmarov@fedora arch-pc]$ git push
Перечисление объектов: 37, готово.
Подсчет объектов: 100% (37/37), готово.
При сжатии изменений используется до 12 потоков
Сжатие объектов: 100% (32/32), готово.
Запись объектов: 100% (32/32), 313.22 КиБ | 2.49 МиБ/с, готово.
Всего 32 (изменений 7), повторно использовано 0 (изменений 0), повторно использо
remote: Resolving deltas: 100% (7/7), completed with 2 local objects.
To github.com:romashalun/study_2023-2024_arch-pc.git
   b2845b9..3a2fadc  master -> master

```

Рис. 4.15: Добавление файлов на GitHub

5 Выводы

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

5.1 Список литературы

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