

Отчёт по лабораторной работе №4

Дисциплина: Архитектура компьютера

Луангсуваннавонг Сайпхачан

Содержание

1	Цель работы	4
2	Задание	5
3	Теоретическое введение	6
4	Выполнение лабораторной работы	8
4.1	Создание программы Hello world!	8
4.2	Работа с транслятором NASM	9
4.3	Работа с расширенным синтаксисом командной строки NASM . . .	10
4.4	Работа с компоновщиком LD	10
4.5	Запуск исполняемого файла	11
5	Выполнение заданий для самостоятельной работы	12
6	Выводы	15
7	Ответы на вопросы для самопроверки	16
8	Список литературы	18

Список иллюстраций

4.1	Создание каталога	8
4.2	Перемещение в созданный каталог	8
4.3	Создание файла	8
4.4	Открытие файла в текстовом редакторе	9
4.5	Файл в текстовом редакторе	9
4.6	Заполнение файла	9
4.7	Компиляция программы	10
4.8	Компиляция программы	10
4.9	Передача объектного файла в компоновщик	11
4.10	Передача объектного файла в компоновщик	11
4.11	Запуск программы hello	11
5.1	Копирование созданного файла	12
5.2	Изменение программы	12
5.3	Компиляция программы	13
5.4	Передача объектного файла в компоновщик	13
5.5	Выполнение программы lab4	13
5.6	Копирование программных файлов в новый каталог	14
5.7	Добавление файлов на GitHub	14
5.8	Загрузка файлов на сервер	14

1 Цель работы

Целью данной лабораторной работы является освоение 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
7. Ответы на вопросы для самопроверки

3 Теоретическое введение

Основными функциональными компонентами любой ЭВМ являются центральный процессор, память и периферийные устройства. Эти устройства взаимодействуют через общую шину, которая соединяет их друг с другом. Физически шина представлена множеством проводников на материнской плате. Основная задача процессора — обработка информации и координация работы всех компонентов компьютера. В состав центрального процессора входят: Арифметико-логическое устройство (АЛУ) - выполняющее логические и арифметические операции с данными из памяти. Устройство управления (УУ) - обеспечивающее контроль над работой всех устройств. Регистры - представляющие собой сверхбыструю оперативную память для временного хранения промежуточных результатов. Регистры делятся на два типа: общего назначения и специальные. Для написания программ на ассемблере необходимо знать существующие регистры и их применение. Большинство ассемблерных команд используют регистры в качестве операндов, а команды представляют собой операции над данными в регистрах, включая их пересылку и преобразование. Доступ к регистрам осуществляется по именам, а не по адресам, как в основной памяти. Архитектура x86 включает следующие основные регистры общего назначения:

64-битные: RAX, RCX, RDX, RBX, RSI, RDI 32-битные: EAX, ECX, EDX, EBX, ESI, EDI
16-битные: AX, CX, DX, BX, SI, DI 8-битные: AH, AL, CH, CL, DH, DL, BH, BL

Другим важным элементом ЭВМ является оперативное запоминающее устройство (ОЗУ). Это быстродействующее энергозависимое хранилище, взаимодействующее с процессором, и предназначенное для хранения программ и данных,

с которыми процессор работает в данный момент. ОЗУ состоит из пронумерованных ячеек памяти, адрес которых используется для доступа к данным.

Периферийные устройства делятся на устройства внешней памяти для длительного хранения данных и устройства ввода-вывода для взаимодействия процессора с внешней средой. В основе вычислительного процесса ЭВМ лежит принцип программного управления, при котором задачи решаются последовательно по записанным программам.

Команды представляют собой двоичные коды, состоящие из операционной и адресной частей. Операционная часть содержит код команды, а адресная — данные или адреса, используемые в операции. При выполнении команды процессор проходит через стандартный командный цикл: формирование адреса команды, считывание и декодирование кода, выполнение команды и переход к следующей.

Язык ассемблера (assembly language, asm) — это низкоуровневый машинно-ориентированный язык. NASM — это открытый проект ассемблера, доступный для различных операционных систем, поддерживающий Intel-синтаксис и инструкции x86-64.

4 Выполнение лабораторной работы

4.1 Создание программы Hello world!

Сначала я открываю терминал и создаю новую директорию “lab04” для работы с программой на языке ассемблера NASM (Рис. 4.1) и затем я перехожу в созданный каталог (Рис. 4.2)

```
[sayprachanh@archlinux ~]$ mkdir -p ~/work/study/2024-2025/"Архитектура компьютера"/arch-pc/lab04  
[sayprachanh@archlinux ~]$
```

Рис. 4.1: Создание каталога

```
[sayprachanh@archlinux ~]$ cd ~/work/study/2024-2025/"Архитектура компьютера"/arch-pc/lab04  
[sayprachanh@archlinux lab04]$
```

Рис. 4.2: Перемещение в созданный каталог

Используя команду “touch”, я создаю пустой текстовый файл “hello.asm” в текущем каталоге (Рис. 4.3)

```
[sayprachanh@archlinux lab04]$ touch hello.asm  
[sayprachanh@archlinux lab04]$
```

Рис. 4.3: Создание файла

Я открываю созданный файл в текстовом редакторе helix(Рис. 4.4 и Рис. 4.5)

```
[sayprachanh@archlinux lab04]$ helix hello.asm  
[sayprachanh@archlinux lab04]$
```

Рис. 4.4: Открытие файла в текстовом редакторе

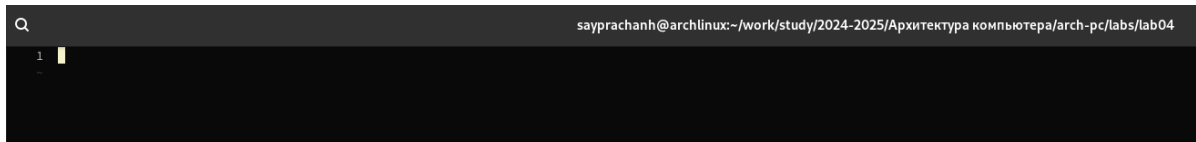


Рис. 4.5: Файл в текстовом редакторе

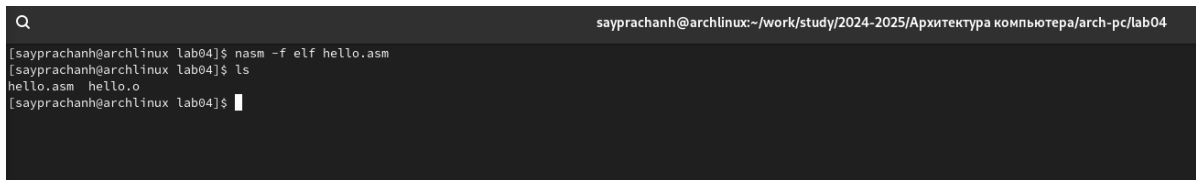
Затем я заполняю файл, вставляя в программу для вывода “Hello world!” (Рис. 4.6)

```
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  
7  SECTION .text ; Начало секции кода  
8  GLOBAL _start  
9  
10  
11 _start: ; Точка входа в программу  
12     mov eax,4 ; Системный вызов для записи (sys_write)  
13     mov ebx,1 ; Описатель файла '1' - стандартный вывод  
14     mov ecx,hello ; Адрес строки hello в ecx  
15     mov edx,helloLen ; Размер строки hello  
16     int 80h ; Вызов ядра  
17     mov eax,1 ; Системный вызов для выхода (sys_exit)  
18     mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)  
19     int 80h ; Вызов ядра  
20
```

Рис. 4.6: Заполнение файла

4.2 Работа с транслятором NASM

Выполняя команду `nasm -f elf hello.asm`, я преобразую текст программы “Hello world!” в объектный код с помощью транслятора TASM. (`-f` в команде указывает транслятору `nasm`, что требуется создать двоичный файл в формате ELF) (Рис. 4.7). Затем я проверяю правильность выполнения команды, мы видим, что файл “hello.o” создан.

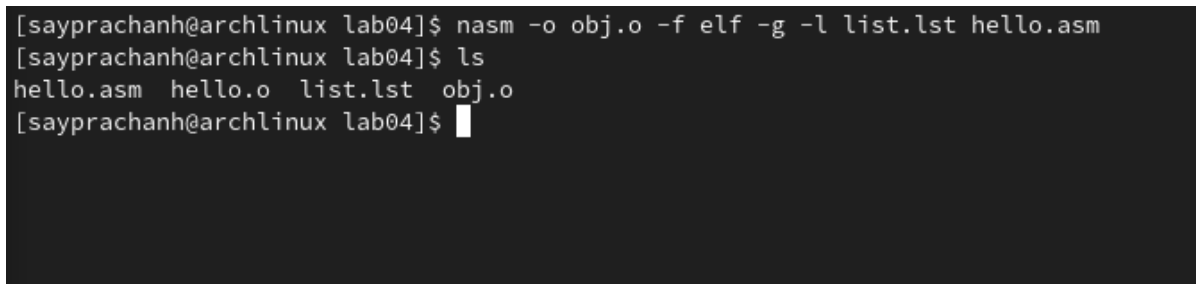


```
Q sayprachanh@archlinux:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab04
[sayprachanh@archlinux lab04]$ nasm -f elf hello.asm
[sayprachanh@archlinux lab04]$ ls
hello.asm  hello.o
[sayprachanh@archlinux lab04]$
```

Рис. 4.7: Компиляция программы

4.3 Работа с расширенным синтаксисом командной строки NASM

После этого я ввожу команду, которая скомпилирует исходный файл `hello.asm` в `obj.o` (Рис. 4.8), а также установит формат выходного файла в `elf`, при этом в него будут включены символы отладки (опция `-g`), и с помощью ключа `-l` также будет создан файл `lst.lst`. Далее я проверяю правильность выполнения команды



```
[sayprachanh@archlinux lab04]$ nasm -o obj.o -f elf -g -l list.lst hello.asm
[sayprachanh@archlinux lab04]$ ls
hello.asm  hello.o  list.lst  obj.o
[sayprachanh@archlinux lab04]$
```

Рис. 4.8: Компиляция программы

4.4 Работа с компоновщиком LD

Я передаю объектный файл `hello.o` компоновщику `ld`, чтобы получить исполняемую программу `hello`. (ключ `-o` задаёт имя создаваемого исполняемого файла)(Рис. 4.9). Затем, используя команду `ls`, я проверяю правильность выполненной команды

```
[sayprachanh@archlinux lab04]$ ld -m elf_i386 hello.o -o hello
[sayprachanh@archlinux lab04]$ ls
hello hello.asm hello.o list.lst obj.o
[sayprachanh@archlinux lab04]$
```

Рис. 4.9: Передача объектного файла в компоновщик

Затем я ввожу следующую команду. Исполняемый файл будет иметь имя `main`, потому что после ключа `-o` было задано значение, в данном случае это было значение `main`, а объектным файлом, который скомпилировал этот исполняемый файл, является `obj.o` (Рис. 4.10)

```
[sayprachanh@archlinux lab04]$ ld -m elf_i386 hello.o -o main
[sayprachanh@archlinux lab04]$ ls
hello hello.asm hello.o list.lst main obj.o
[sayprachanh@archlinux lab04]$
```

Рис. 4.10: Передача объектного файла в компоновщик

4.5 Запуск исполняемого файла

Наконец, я запускаю созданный исполняемый файл `hello` (Рис. 4.11)

```
[sayprachanh@archlinux lab04]$ ./hello
Hello world!
[sayprachanh@archlinux lab04]$
```

Рис. 4.11: Запуск программы `hello`

5 Выполнение заданий для самостоятельной работы

В текущем каталоге я копирую файл `hello.asm` и называю его `lab4.asm` с помощью утилиты `cp` (Рис. 5.1)

```
[sayprachanh@archlinux lab04]$ cp hello.asm lab4.asm
[sayprachanh@archlinux lab04]$ ls
hello  hello.asm  hello.o  lab4.asm  list.lst  main  obj.o
[sayprachanh@archlinux lab04]$
```

Рис. 5.1: Копирование созданного файла

Используя текстовый редактор `helix`, я редактирую содержимое в `lab4.asm` и заполняю программу, которая будет выводить мое имя и фамилию при запуске программы (Рис. 5.2)

```
Q sayprachanh@archlinux:~/work/study/2024-2025/Архитектура компьютера/arch-pc/lab04
1 ; lab4.asm
2 SECTION .data ; Начало секции данных
3     lab4: DB 'Луангсуваннавонг Сайпхачан',10
4
5     lab4Len: EQU $-lab4 ; Длина строки lab4
6
7
8 SECTION .text ; Начало секции кода
9     GLOBAL _start
10
11
12 _start: ; Точка входа в программу
13     mov eax,4 ; Системный вызов для записи (sys_write)
14     mov ebx,1 ; Описатель файла '1' - стандартный вывод
15     mov ecx,lab4 ; Адрес строки hello в ecx
16     mov edx,lab4Len ; Размер строки hello
17     int 80h ; Вызов ядра
18     mov eax,1 ; Системный вызов для выхода (sys_exit)
19     mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
20     int 80h ; Вызов ядра
```

Рис. 5.2: Изменение программы

Я компилирую текст программы в объектный файл, затем, используя ls, проверяю, создан ли объектный файл lab4.o (Рис. 5.3)

```
[sayprachanh@archlinux lab04]$ nasm -f elf lab4.asm
[sayprachanh@archlinux lab04]$ ls
hello hello.asm hello.o lab4.asm lab4.o list.lst main obj.o
[sayprachanh@archlinux lab04]$
```

Рис. 5.3: Компиляция программы

Затем я передаю объектный файл lab4.o компоновщику ld, чтобы получить исполняемый файл lab4 (Рис. 5.4)

```
[sayprachanh@archlinux lab04]$ ld -m elf_i386 lab4.o -o lab4
[sayprachanh@archlinux lab04]$ ls
hello hello.asm hello.o lab4 lab4.asm lab4.o list.lst main obj.o
[sayprachanh@archlinux lab04]$
```

Рис. 5.4: Передача объектного файла в компоновщик

Запускаю созданный исполняемый файл lab4, и мы видим, что в нем отображаются мои имя и фамилия (Рис. 5.5)

```
[sayprachanh@archlinux lab04]$ ./lab4
Луангсуваннавонг Сайпхачан
[sayprachanh@archlinux lab04]$
```

Рис. 5.5: Выполнение программы lab4

Копирую файлы hello.asm и lab4.asm в каталог курса lab04, созданный на основе прошлых лабораторных работ (Рис. 5.6). Затем проверяю правильность выполненной команды

```
[sayprachanh@archlinux lab04]$ cp hello.asm lab4.asm ~/work/study/2024-2025/Архитектура\ компьютера/arch-pc/labs/lab04
[sayprachanh@archlinux lab04]$ ls ~/work/study/2024-2025/Архитектура\ компьютера/arch-pc/labs/lab04
hello.asm lab4.asm presentation report
[sayprachanh@archlinux lab04]$
```

Рис. 5.6: Копирование программных файлов в новый каталог

Используя команду `git add .` и `git commit`, я добавляю файлы на GitHub с комментарием “добавляю файлы для лабораторной работы № 5”. (Рис. 5.7)

```
[sayprachanh@archlinux arch-pc]$ git add .
[sayprachanh@archlinux arch-pc]$ git commit -m "Adding files for lab04"
[master 607b26b] Adding files for lab04
11 files changed, 101 insertions(+)
```

Рис. 5.7: Добавление файлов на GitHub

Наконец, я загружаю файлы на сервер с помощью команды `git push` (Рис. 5.8)

```
[sayprachanh@archlinux arch-pc]$ git push
Enumerating objects: 16, done.
Counting objects: 100% (16/16), done.
Delta compression using up to 4 threads
Compressing objects: 100% (13/13), done.
Writing objects: 100% (13/13), 3.17 KiB | 1.59 MiB/s, done.
Total 13 (delta 6), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (6/6), completed with 2 local objects.
To github.com:sayprachanh-lsvnv/study_2024-2025_arh-pc.git
73d1479..607b26b master -> master
[sayprachanh@archlinux arch-pc]$
```

Рис. 5.8: Загрузка файлов на сервер

6 Выводы

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

7 Ответы на вопросы для самопроверки

1. Какие основные отличия ассемблерных программ от программ на языках высокого уровня?

Отличия ассемблера и языков высокого уровня: Ассемблер ближе к машинам, сложнее, требует знаний о компьютере. Языки высокого уровня проще и удобнее.

2. В чём состоит отличие инструкции от директивы на языке ассемблера?

Инструкция и директива: Инструкция выполняет действия (сложение чисел или перемещение данных), а Директива — это указание для компилятора, например, как организовать память или какие данные использовать. Директивы не выполняются, они просто помогают программе работать правильно.

3. Перечислите основные правила оформления программ на языке ассемблера?

При написании программы на ассемблере нужно соблюдать несколько правил. Каждая строка должна содержать либо инструкцию, либо директиву. Хорошо использовать метки для удобства. Важно добавлять комментарии, чтобы объяснить код. Также следите за правильным синтаксисом и отступами для лучшей читаемости.

4. Каковы этапы получения исполняемого файла?

Чтобы получить исполняемый файл, нужно пройти несколько этапов. Сначала пишется исходный код на ассемблере, затем он преобразуется в объектный файл. После этого компоновщик объединяет объектные файлы в один исполняемый файл, который можно запускать на компьютере.

5. Каково назначение этапа трансляции?

Этап трансляции преобразует исходный код из ассемблера в объектный файл с машинным кодом, который понимает процессор. Без него программа не будет работать.

6. Каково назначение этапа компоновки?

Компоновка объединяет разные объектные файлы в один исполняемый файл и связывает части программы для корректной работы, особенно если они зависят друг от друга.

7. Какие файлы могут создаваться при трансляции программы, какие из них создаются по умолчанию?

При трансляции создаётся несколько файлов. Основной — объектный файл, который создаётся по умолчанию. Также могут быть листинг-файл с ассемблерным кодом и файл с сообщениями об ошибках, если есть ошибки в коде.

8. Каковы форматы файлов для `nasm` и `ld`?

NASM поддерживает различные форматы файлов, такие как ELF (обычно для Linux) и COFF (для Windows). LD, компоновщик, также использует формат ELF для Linux и COFF для Windows.

8 Список литературы

Архитектура ЭВМ