

Лабораторная работа№4

Дисциплина:Архитектура компьютера

Тойчубекова Асель Нурлановна

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	8
4	Выполнение лабораторной работы	12
4.1	Программа Hello world!	12
4.2	Транслятор NASM	13
4.3	Расширенный синтаксис командной строки NASM	13
4.4	Компоновщик LD	14
4.5	Запуск исполняемого файла	14
4.6	Задание для самостоятельной работы	15
5	Выводы	18
	Список литературы	19

Список иллюстраций

3.1	РИС.1.1 Пример кода языка ассемблера	10
4.1	РИС.2.1 Создание каталога	12
4.2	РИС.2.2_1 Создание файла hello.asm	12
4.3	РИС.2.2_2 Написание программы в файле	13
4.4	РИС.3.1 Компиляция текста программы	13
4.5	РИС.4.1 Компиляция текста программы	14
4.6	РИС.5.1 Передача объективного файла на обработку компоовщику ld	14
4.7	РИС.5.2 Передача объективного файла на обработку компоовщику ld с заданным именем	14
4.8	РИС.6.1 Запуск исполняемого файла	15
4.9	РИС.7.1 Создание копии файла с другим именем	15
4.10	РИС.7.2 Изменение программы	15
4.11	РИС.7.3 Компиляция программы в объективный файл	16
4.12	РИС.7.4 Компоновка объективного файла	16
4.13	РИС.7.5 Запуск исполняемого файла	16
4.14	РИС.7.6 Копирование файлов	16
4.15	РИС.7.7 Загрузка файлов в github	17
4.16	РИС.7.8 Проверка загрузки файлов в github	17

Список таблиц

1 Цель работы

Целью работы является освоение процедуры компиляции и сборки программ, написанных на ассемблере NASM.

2 Задание

- Понять основные принципы работы компьютера:

- работа АЛУ;
- устройство управления;
- работа регистров;
- устройство внешней памяти;
- устройства ввода-вывода.

- Познакомиться с ассемблером и языком ассемблера:

- как создавать и обрабатывать программы на языке ассемблер; -общие понятия о языке ассемблера NASM;
- работа транслятора NASM;
- работа командной строки NASM;
- работа компоновщика;
- как запустить исполняемый файл.

- Написать программу для вывода на экран “Hello world”.

- Задание для самостоятельной работы:

1. В каталоге ~/work/arch-рс/lab04 с помощью команды `cp` создать копию файла `hello.asm` с именем `lab4.asm`
2. С помощью любого текстового редактора внести изменения в текст программы в файле `lab4.asm` так, чтобы вместо `Hello world!` на экран выводилась строка с фамилией и именем.

3. Оттранслировать полученный текст программы lab4.asm в объектный файл. Выполнить компоновку объектного файла и запустить получившийся исполняемый файл.
4. Скопировать файлы hello.asm и lab4.asm в наш локальный репозиторий в каталог ~/work/study/2023-2024/“Архитектура компьютера”/arch-rc/labs/lab04/. Загрузить файлы на Github.

3 Теоретическое введение

Основными функциональными элементами любой электронно-вычислительной машины (ЭВМ) являются центральный процессор, память и периферийные устройства, их взаимодействие осуществляется через общую шину, к которой они подключены. В состав центрального процессора входят:

- арифметико-логическое устройство(АЛУ);
- устройство управления(уу);
- регистры.

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.

В состав ЭВМ также входят периферийные устройства:

- устройство внешней памяти;
- устройства ввода-вывода;

В основе вычислительного процесса ЭВМ лежит принцип программного управления, то компьютер решает задачу последовательно, следуя программе, записанная в виде кодов. При выполнении каждой команды процессор выполняет определённую последовательность стандартных действий, которая называется

командным циклом процессора. В самом общем виде 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 (англ. «Assembler») — это низкоуровневый язык программирования, который представляет собой промежуточное звено между машинным кодом и высокоуровневыми языками программирования. Он используется для написания программ, которые управляют компьютером или другими устройствами на более низком уровне, непосредственно взаимодействуя с аппаратным обеспечением. Код, написанный на этом языке, обычно сохраняется с помощью расширения ASM.

Программы на ассемблере пишутся в виде набора мнемонических инструкций, каждая из которых соответствует определенной команде процессора. Эти инструкции затем транслируются (ассемблируются) в машинный код — набор двоичных чисел, которые понимает центральный процессор и выполняет соответствующие операции. Для каждой архитектуры существует свой ассемблер и свой язык ассемблера.

Ниже приведена РИС.1.1 с примерами кода, соответствующими инструкциями на ассемблере и их описание.

Машинный код	Инструкции на ассемблере	Описание
10110000	MOV AL, 0	Загрузить значение 0 в регистр AL
10110001	MOV BL, 1	Загрузить значение 1 в регистр BL
10001011	ADD AL, BL	Сложить значения в регистрах AL и BL
00100000	JMP label	Безусловный переход к метке (адресу) label
11001001	CMP AX, BX	Сравнить значения в регистрах AX и BX
01100010	JZ label	Переход к метке label, если флаг ZF=1
10010110	SUB AL, 6	Вычесть 6 из значения в регистре AL
11001000	AND AX, BX	Побитовое И между значениями AX и BX
11101010	OR AL, 10	Побитовое ИЛИ значения в регистре AL и 10
10111000	XOR AL, 8	Побитовое исключающее ИЛИ с 8 в регистре AL
00000000	NOP	Пустая операция (ничего не делать)
01100110	PUSH AX	Положить значение из регистра AX в стек
10010111	POP CX	Извлечь значение из стека в регистр CX
11011000	CALL subroutine	Вызвать подпрограмму (переход с сохранением адреса возврата)

Рис. 3.1: РИС.1.1 Пример кода языка ассемблера

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.

Типичный формат записи команд NASM имеет вид: [метка: мнемокод [операнд {, операнд}]] [; комментарий]

В процессе создания ассемблерной программы можно выделить четыре шага:

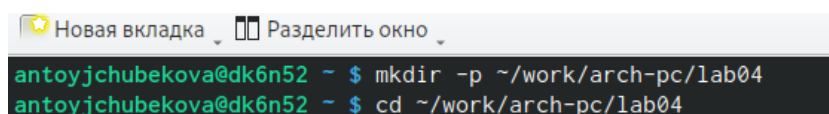
- набор текста в соответствующий файл;
- трансляция текста а=программы с помощью транслятора,наприме nasm;
- компоновка или линковка;

- запуск программы.

4 Выполнение лабораторной работы

4.1 Программа Hello world!

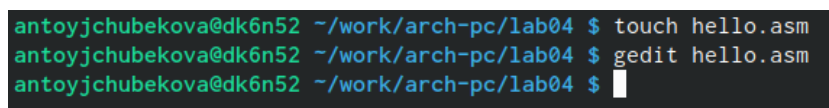
Для начала создадим каталог для работы с программами на языке ассемблера NASM, затем перейдем в этот каталог.(РИС.2.1)

A screenshot of a terminal window with a light gray title bar containing "Новая вкладка" and "Разделить окно". The terminal has a dark background with green text. It shows two commands being executed: "mkdir -p ~/work/arch-pc/lab04" and "cd ~/work/arch-pc/lab04".

```
antoyjchubekova@dk6n52 ~ $ mkdir -p ~/work/arch-pc/lab04
antoyjchubekova@dk6n52 ~ $ cd ~/work/arch-pc/lab04
```

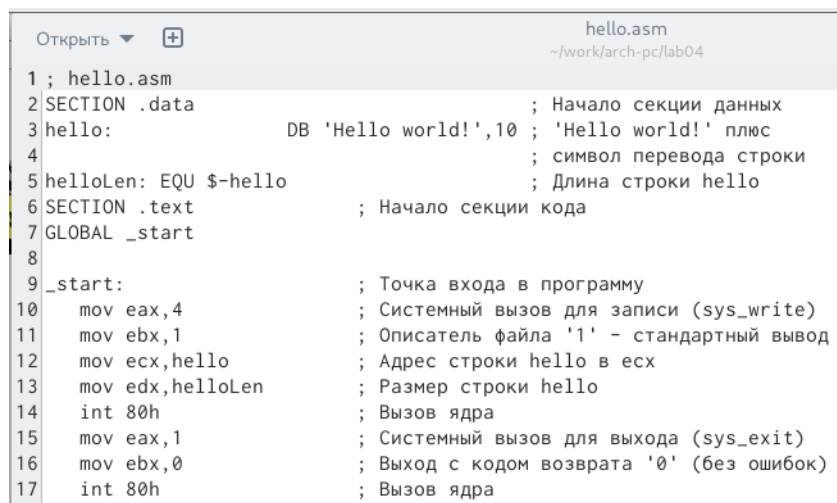
Рис. 4.1: РИС.2.1 Создание каталога

Создадим текстовый файл с именем hello.asm с помощью touch, затем откроем этот файл с помощью редактора report и напишем в него программу для вывода на экран Hello world!(РИС2.2_1) И (РИС.2.2_2)

A screenshot of a terminal window with a dark background and green text. It shows three commands being executed: "touch hello.asm", "gedit hello.asm", and a prompt for the next command.

```
antoyjchubekova@dk6n52 ~/work/arch-pc/lab04 $ touch hello.asm
antoyjchubekova@dk6n52 ~/work/arch-pc/lab04 $ gedit hello.asm
antoyjchubekova@dk6n52 ~/work/arch-pc/lab04 $
```

Рис. 4.2: РИС.2.2_1 Создание файла hello.asm



```

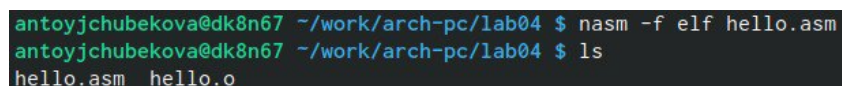
Открыть ▾ + hello.asm
~/work/arch-pc/lab04
1 ; hello.asm
2 SECTION .data ; Начало секции данных
3 hello: DB 'Hello world!',10 ; 'Hello world!' плюс
4 ; символ перевода строки
5 helloLen: EQU $-hello ; Длина строки hello
6 SECTION .text ; Начало секции кода
7 GLOBAL _start
8
9 _start: ; Точка входа в программу
10 mov eax,4 ; Системный вызов для записи (sys_write)
11 mov ebx,1 ; Описатель файла '1' - стандартный вывод
12 mov ecx,hello ; Адрес строки hello в ecx
13 mov edx,helloLen ; Размер строки hello
14 int 80h ; Вызов ядра
15 mov eax,1 ; Системный вызов для выхода (sys_exit)
16 mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
17 int 80h ; Вызов ядра

```

Рис. 4.3: РИС.2.2_2 Написание программы в файле

4.2 Транслятор NASM

Дальше с помощью команды `nasm -f elf hello.asm` компилируем текст программы для вывода Hello world! в объективный код в новом файле `hello.o`, в формате ELF. Используя команду `ls` мы увидим, что файл был создан.(РИС.3.1)



```

antojychubekova@dk8n67 ~/work/arch-pc/lab04 $ nasm -f elf hello.asm
antojychubekova@dk8n67 ~/work/arch-pc/lab04 $ ls
hello.asm hello.o

```

Рис. 4.4: РИС.3.1 Компиляция текста программы

4.3 Расширенный синтаксис командной строки NASM

Введем команду для компиляции файла `hello.asm` в файл `obj.o`, при этом формат выходного файла будет `elf` и в него будут включены символы для отладки (опции-`g`), также будет создан файл листинга `list.lst`(опции-`l`). С помощью команды `ls` мы удостоверились, что все файлы были правильно созданы.(РИС.4.1)

```
antoychubekova@dk8n67 ~/work/arch-pc/lab04 $ nasm -o obj.o -f elf -g -l list.lst hello.asm
antoychubekova@dk8n67 ~/work/arch-pc/lab04 $ ls
hello.asm  hello.o  list.lst  obj.o
```

Рис. 4.5: РИС.4.1 Компиляция текста программы

4.4 Компоновщик LD

Дальше передадим объективный файл hello.o на обработку компоновщику LD используя команду “`d -m elf_i386 hello.o -o hello`”. Дальше проверяем корректность выполнения команды, введя команду `ls`. (РИС.5.1)

```
antoychubekova@dk8n67 ~/work/arch-pc/lab04 $ ld -m elf_i386 hello.o -o hello
antoychubekova@dk8n67 ~/work/arch-pc/lab04 $ ls
hello  hello.asm  hello.o  list.lst  obj.o
```

Рис. 4.6: РИС.5.1 Передача объективного файла на обработку компоновщику ld

Дальше введем команду “`d -m elf_i386 obj.o -o main`”. Исполняемый файл будет иметь имя `main`, так как после ключа `-o` было задано значение `main`. Проверим наличие файлов командой `ls`. Объективный файл, из которого собран этот исполняемый файл, имеет имя `obj.o`. (РИС.5.2)

```
antoychubekova@dk8n67 ~/work/arch-pc/lab04 $ ld -m elf_i386 obj.o -o main
antoychubekova@dk8n67 ~/work/arch-pc/lab04 $ ls
hello  hello.asm  hello.o  list.lst  main  obj.o
```

Рис. 4.7: РИС.5.2 Передача объективного файла на обработку компоновщику ld с заданным именем

4.5 Запуск исполняемого файла

Запускаем на выполнение созданный исполняемый файл `hello` и види, что она успешно запустилась (РИС.6.1)

```
antoychubekova@dk8n67 ~/work/arch-pc/lab04 $ ./hello
Hello world!
antoychubekova@dk8n67 ~/work/arch-pc/lab04 $
```

Рис. 4.8: РИС.6.1 Запуск исполняемого файла

4.6 Задание для самостоятельной работы

Для начала в каталоге ~/work/arch-pc/lab04 с помощью команды cp создадим копию файла hello.asm с именем lab4.asm. С помощью ls проверим их наличие(РИС.7.1)

```
antoychubekova@dk8n67 ~/work/arch-pc/lab04 $ cp hello.asm lab4.asm
antoychubekova@dk8n67 ~/work/arch-pc/lab04 $
antoychubekova@dk8n67 ~/work/arch-pc/lab04 $ ls
hello hello.asm hello.o lab4.asm list.lst main obj.o
```

Рис. 4.9: РИС.7.1 Создание копии файла с другим именем

С помощью текстового редактора gedit вношу изменения в текст программы в файле lab4.asm так, чтобы на экран выводилась мое фамилие и имя.(РИС.7.2)

```
Открыть ▾ + lab4.asm
~/work/arch-pc/lab04
1 ; lab4.asm
2 SECTION .data ; Начало секции данных
3 lab4: DB 'Toichubekova Asel',10 ; 'Toichubekova Asel' плюс
4 ; символ перевода строки
5 lab4Len: EQU $-lab4 ; Длина строки lab4
6 SECTION .text ; Начало секции кода
7 GLOBAL _start
8
9 _start: ; Точка входа в программу
10 mov eax,4 ; Системный вызов для записи (sys_write)
11 mov ebx,1 ; Описатель файла '1' - стандартный вывод
12 mov ecx,lab4 ; Адрес строки lab4 в ecx
13 mov edx,lab4Len ; Размер строки lab4
14 int 80h ; Вызов ядра
15 mov eax,1 ; Системный вызов для выхода (sys_exit)
16 mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
17 int 80h ; Вызов ядра
```

Рис. 4.10: РИС.7.2 Изменение программы

Оттранслируем полученный текст программы lab4.asm в объектный файл. С помощью команды ls мы видим, что файл lab4.o был создан.(РИС.7.3)

```
antoyjchubekova@dk8n67 ~/work/arch-pc/lab04 $ nasm -f elf lab4.asm
antoyjchubekova@dk8n67 ~/work/arch-pc/lab04 $ ls
hello hello.asm hello.o lab4.asm lab4.o list.lst main obj.o
```

Рис. 4.11: РИС.7.3 Компиляция программы в объективный файл

Далее выполним компоновку объектного файла, с компоновщиком ld. Введя команду ls можно заметить, что исполняемый файл lab4 существует.(РИС.7.4)

```
antoyjchubekova@dk8n67 ~/work/arch-pc/lab04 $ ld -m elf_i386 lab4.o -o lab4
antoyjchubekova@dk8n67 ~/work/arch-pc/lab04 $ ls
hello hello.asm hello.o lab4 lab4.asm lab4.o list.lst main obj.o
```

Рис. 4.12: РИС.7.4 Компоновка объективного файла

Запустим получившийся исполняемый файл.(РИС.7.5)

```
antoyjchubekova@dk8n67 ~/work/arch-pc/lab04 $ ./lab4
Toichubekova Asel
```

Рис. 4.13: РИС.7.5 Запуск исполняемого файла

Скопируем файлы hello.asm и lab4.asm в наш локальный репозиторий в каталог-~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/. Перейдя в наш репозиторий и с командой ls видим, что все скопировалось.(РИС.7.6)

```
antoyjchubekova@dk8n67 ~/work/arch-pc/lab04 $ cp hello.asm ~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04
antoyjchubekova@dk8n67 ~/work/arch-pc/lab04 $ cp lab4.asm ~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04
antoyjchubekova@dk8n67 ~/work/arch-pc/lab04 $ cd ~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04
antoyjchubekova@dk8n67 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ ls
hello.asm lab4.asm presentation report
```

Рис. 4.14: РИС.7.6 Копирование файлов

Загрузим файлы на Github и проверим их наличие.(РИС.7.7) И (РИС.7.8).


```

antoyjchubekova@dk8n67 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ git add .
antoyjchubekova@dk8n67 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ git commit -am 'feat(main):add files lab04'
[master 00e600d] feat(main):add files lab04
2 files changed, 34 insertions(+)
create mode 100644 labs/lab04/hello.asm
create mode 100644 labs/lab04/lab4.asm
antoyjchubekova@dk8n67 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $ git push
Пересчет объектов: 100% (9/9), готово.
При сжатии изменений используется до 6 потоков
Сжатие объектов: 100% (6/6), готово.
Запись объектов: 100% (6/6), 1.05 КиБ | 1.05 МБ/с, готово.
Всего 6 (изменений 3), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (3/3), completed with 2 local objects.
To github.com:aselttoichubekova/study_2023-2024_arh-pc.git
  910a937..00e600d master -> master
antoyjchubekova@dk8n67 ~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/lab04 $

```

Рис. 4.15: РИС.7.7 Загрузка файлов в github

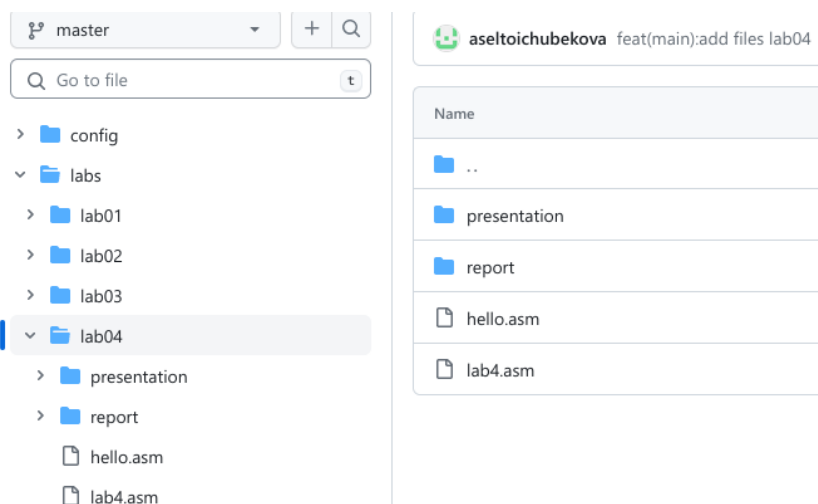


Рис. 4.16: РИС.7.8 Проверка загрузки файлов в github

5 Выводы

В ходе выполнения лабораторной работы №4 я поняла основные принципы компьютера, познакомилась с понятием ассемблер и с языком ассемблера NASM, освоила процедуры компиляции и сборки программ, написанных на ассемблере NASM. Пользуясь полученными знаниями написала программу на языке ассемблера NASM, выводящая на экран фразу “Hello world!” и мое имя и фамилию.

Список литературы

- <https://esystem.rudn.ru/course/view.php?id=4975>
- <https://skillbox.ru/media/code/что-такое-ассемблер/>
- <https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source>