

Архитектура компьютера

Отчёт по лабораторной работе №4

Ибрахим Мохсейн Алькамал

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	10
5	Выполнение заданий для самостоятельной работы	13
6	Выводы	16
	Список литературы	17

Список иллюстраций

4.1	Создание папки	10
4.2	Переход в созданную папку	10
4.3	Создание файла	10
4.4	Заполнение файла	11
4.5	Скачивание	11
4.6	Преобразование файла в объектный код	11
4.7	Преобразование файла	12
4.8	Передача файла на обработку	12
4.9	1	12
4.10	2	12
5.1	Копирование файла	13
5.2	Файл в папо	13
5.3	Транслирую файл	14
5.4	Компановка и исполнение	14
5.5	Копирование файлов	14
5.6	Выгружаю изменения	15

Список таблиц

1 Цель работы

Освоение процедуры компиляции и сборки программ, написанных на ассемблере NASM.

2 Задание

- 1) Создать программу Hello world
- 2) Работа с транслятором NASM
- 3) Работа с расширенным синтаксисом командой строки NASM
- 4) Работа с компоновщиком LD
- 5) Запуск исполняемого файла
- 6) Выполнение заданий для самостоятельной работы

3 Теоретическое введение

Основной задачей процессора является обработка информации, а также организация координации всех узлов компьютера. В состав центрального процессора (ЦП) 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 объема, входящая в состав процессора, для временного хранения промежуточных результатов выполнения инструкций; регистры процессора делятся на два типа: регистры общего назначения и специальные регистры. В процессе создания ассемблерной программы можно выделить четыре шага: - Набор текста программы в текстовом редакторе и сохранение её в отдельном файле. Каждый файл имеет свой тип (или расширение), который определяет назначение файла. Файлы с исходным текстом программ на языке ассемблера имеют тип `asm`. - Трансляция — преобразование с помощью транслятора, например `nasm`, текста программы в машинный код, называемый объектным. На данном этапе также может быть получен листинг программы, содержащий кроме текста программы различную дополнительную информацию, созданную транслятором. Тип объектного файла — `o`, файла листинга — `lst`. - Компоновка или линковка — этап обработки объектного кода компоновщиком (`ld`), который принимает на вход объектные файлы и собирает по ним исполняемый файл. Исполняемый файл обычно не имеет расширения. Кроме того, можно получить файл карты загрузки программы в ОЗУ, имеющий расширение

мар. - Запуск программы. Конечной целью является работоспособный исполняемый файл. Ошибки на предыдущих этапах могут привести к некорректной работе программы, поэтому может присутствовать этап отладки программы при помощи специальной программы — отладчика. При нахождении ошибки необходимо провести коррекцию программы, начиная с первого шага. 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 (половинки 16-битных регистров).

Например,

AH (high AX) — старшие 8 бит регистра AX, AL (low AX) — младшие 8 бит регистра AX. В состав ЭВМ также входят периферийные устройства, которые можно разделить на: - устройства внешней памяти, которые предназначены для длительного хранения больших объёмов данных (жёсткие диски, твердотельные накопители, магнитные ленты); - 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. Программа состоит из машинных команд, которые указывают, какие операции и над какими данными (или операндами), в какой последовательности необходимо выполнить. Набор машинных команд определяется устройством конкретного процессора. 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. 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.

4 Выполнение лабораторной работы

- 1) Создаю рекурсивно вложенные в папку work папки arch-pc и lab04, проверяю их создание

```
(alkamal@Localhost)-[~/work/arch-pc]
$ mkdir -p ~/work/arch-pc/lab04

(alkamal@Localhost)-[~/work/arch-pc]
$ ls ~/work/arch-pc
lab04
```

Рис. 4.1: Создание папки

- 2) Перехожу в созданную папку

```
(alkamal@Localhost)-[~/work/arch-pc]
$ cd ~/work/arch-pc/lab04

(alkamal@Localhost)-[~/work/arch-pc/lab04]
$
```

Рис. 4.2: Переход в созданную папку

- 3) Создаю файл hello с разрешением asm и проверяю его создание

```
(alkamal@Localhost)-[~/work/arch-pc/lab04]
$ touch hello.asm

(alkamal@Localhost)-[~/work/arch-pc/lab04]
$ ls
hello.asm
```

Рис. 4.3: Создание файла

- 4) Открываю этот файл в папо и копирую туда код из задания лабораторной работы

```

GNU nano 7.2                                hello.asm
; 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

5) Скачиваю nasm

```

└─$ sudo apt install nasm
[sudo] password for alkamal:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
nasm is already the newest version (2.16.01-1).
nasm set to manually installed.
The following packages were automatically installed and are no longer required:
 fonts-junicode insserv libc-bin libc6 libc6-i386 libbsd0 libbz2-1.0 libcap2
 linux-headers-6.5.0-kali1-amd64 linux-headers-6.5.0-kali1-common linux-image-6.5.0-kali1-amd64 linux-kbuild-6.5.0-kali1 python3-icmplib
 python3-pyminifier startpar sysv-rc
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 10 not upgraded.

```

Рис. 4.5: Скачивание

6) Преобразовываю файл hello.asm в объектный код, записанный в файл hello.o. Проверяю, был ли создан файл

```

(alkamal@Localhost)-[~/work/arch-pc/lab04]
└─$ nasm -f elf hello.asm

(alkamal@Localhost)-[~/work/arch-pc/lab04]
└─$ ls
hello.asm  hello.o

```

Рис. 4.6: Преобразование файла в объектный код

7) Преобразую файл hello.asm в obj.o с помощью опции -o, которая позволяет задать имя объекта. Из-за elf -g формат выходного файла будет elf, и в него будут включены символы для отладки, а так же будет создан файл листинга list.lst, благодаря -l. Проверяю созданные файлы

```

(alkamal@Localhost)-[~/work/arch-pc/lab04]
$ nasm -o obj.o -f elf -g -l list.lst hello.asm

(alkamal@Localhost)-[~/work/arch-pc/lab04]
$ ls
hello.asm  hello.o  list.lst  obj.o

```

Рис. 4.7: Преобразование файла

- 8) Передаю файл компоновщику с помощью ld. Проверяю, создан ли исполняемый файл

```

(alkamal@Localhost)-[~/work/arch-pc/lab04]
$ ld -m elf_i386 hello.o -o hello

(alkamal@Localhost)-[~/work/arch-pc/lab04]
$ ls
hello  hello.asm  hello.o  list.lst  obj.o

```

Рис. 4.8: Передача файла на обработку

- 9) Передаю компоновщику файл obj.o и называю скомпонованный файл main. запуская сначала код для предыдущего файла(1), а затем для созданного сейчас(2)

```

(alkamal@Localhost)-[~/work/arch-pc/lab04]
$ ld -m elf_i386 obj.o -o main

(alkamal@Localhost)-[~/work/arch-pc/lab04]
$ ./hello
Hello world!

```

Рис. 4.9: 1

```

(alkamal@Localhost)-[~/work/arch-pc/lab04]
$ ./main
Hello world!

```

Рис. 4.10: 2

5 Выполнение заданий для самостоятельной работы

1) Копирую hello.asm с названием lab4.asm

```
(alkamal@Localhost)-[~/work/arch-pc/lab04]
$ cp hello.asm lab4.asm

(alkamal@Localhost)-[~/work/arch-pc/lab04]
$ ls
hello  hello.asm  hello.o  lab4.asm  list.lst  main  obj.o
```

Рис. 5.1: Копирование файла

2) С помощью nano изменяю текст кода так, чтобы он выводил моё имя и фамилию

```
GNU nano 7.2 lab4.asm
hello.asm
SECTION .data ; Начало секции данных
hello: DB 'Alkamal Ebrahim!',10 ; 'my name!' 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 ехх
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. 5.2: Файл в nano

3) Транслирую файл lab4.asm в объектный

```
(alkamal@Localhost)-[~/work/arch-pc/lab04]  
$ nasm -f elf lab4.asm
```

Рис. 5.3: Транслирую файл

- 4) Выполняю компоновку и запускаю исполняемый файл

```
(alkamal@Localhost)-[~/work/arch-pc/lab04]  
$ ld -m elf_i386 lab4.o -o lab4  
  
(alkamal@Localhost)-[~/work/arch-pc/lab04]  
$ ./lab4  
ALKamal Ebrahim!
```

Рис. 5.4: Компоновка и исполнение

- 5) Копирую файлы в мой локальный репозиторий

```
(alkamal@Localhost)-[~/work/arch-pc/lab04]  
$ cp hello.asm ~/work/study/2023-2024/Архитектура\ компьютера/study_2023_2024_arch-pc/labs/Lab04  
  
(alkamal@Localhost)-[~/work/arch-pc/lab04]  
$ cp lab4.asm ~/work/study/2023-2024/Архитектура\ компьютера/study_2023_2024_arch-pc/labs/Lab04
```

Рис. 5.5: Копирование файлов

- 6) Выгружаю изменения на GitHub

```
alkamal@Localhost: ~/work/study/2023-2024/Архитектура компьютера/study_2023_2024_arch-...
(alkamal@Localhost)-[~/study_2023_2024_arch-pc/labs/lab04/report]
$ git add .
(alkamal@Localhost)-[~/study_2023_2024_arch-pc/labs/lab04/report]
$ git commit -am 'add lab04'
[master 1bf982a] add lab04
31 files changed, 4405 insertions(+)
delete mode 100644 labs/lab02/report/.л03_Ибрахим_Отчёт.swp
create mode 100644 labs/lab04/report/Makefile
create mode 100644 labs/lab04/report/bib/cite.bib
create mode 100644 labs/lab04/report/image/1.jpg
create mode 100644 labs/lab04/report/image/10.jpg
create mode 100644 labs/lab04/report/image/11.jpg
create mode 100644 labs/lab04/report/image/12.jpg
create mode 100644 labs/lab04/report/image/13.jpg
create mode 100644 labs/lab04/report/image/14.jpg
create mode 100644 labs/lab04/report/image/2.jpg
create mode 100644 labs/lab04/report/image/3.jpg
create mode 100644 labs/lab04/report/image/4.jpg
create mode 100644 labs/lab04/report/image/5.jpg
create mode 100644 labs/lab04/report/image/6.jpg
create mode 100644 labs/lab04/report/image/7.jpg
create mode 100644 labs/lab04/report/image/8.jpg
create mode 100644 labs/lab04/report/image/9.jpg
create mode 100644 labs/lab04/report/image/91.jpg
create mode 100644 labs/lab04/report/image/placeimg_800_600_tech.jpg
create mode 100644 labs/lab04/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
create mode 100755 labs/lab04/report/pandoc/filters/pandoc_eqnos.py
create mode 100755 labs/lab04/report/pandoc/filters/pandoc_fignos.py
create mode 100755 labs/lab04/report/pandoc/filters/pandoc_secnos.py
create mode 100755 labs/lab04/report/pandoc/filters/pandoc_tablenos.py
create mode 100644 labs/lab04/report/pandoc/filters/pandocxnos/__init__.py
create mode 100644 labs/lab04/report/pandoc/filters/pandocxnos/core.py
create mode 100644 labs/lab04/report/pandoc/filters/pandocxnos/main.py
create mode 100644 labs/lab04/report/pandoc/filters/pandocxnos/pandocattributes.py
create mode 100644 labs/lab04/report/л04_Ибрахим_отчёт.docx
create mode 100644 labs/lab04/report/л04_Ибрахим_отчёт.md
create mode 100644 labs/lab04/report/л04_Ибрахим_отчёт.pdf
(alkamal@Localhost)-[~/study_2023_2024_arch-pc/labs/lab04/report]
$ git push
Enumerating objects: 72, done.
Counting objects: 100% (72/72), done.
Delta compression using up to 6 threads
Compressing objects: 100% (65/65), done.
Writing objects: 100% (70/70), 5.16 MiB | 3.09 MiB/s, done.
Total 70 (delta 7), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (7/7), completed with 1 local object
```

Рис. 5.6: Выгружаю изменения

6 Выводы

Я освоила процедуры компиляции и сборки программ, написанных на ассемблере NASM

Список литературы

#refs <https://esystem.rudn.ru/mod/resource/view.php?id=1030552>