

--- ## Front matter title: “Отчёт по лабораторной работе № 5” subtitle:
“Дисциплина: Архитектура Компьютера” author: “Извекова Мария Петровна,
НКАБд-01-22”

0.1 Generic options

lang: ru-RU toc-title: “Содержание”

0.2 Bibliography

bibliography: bib/cite.bib csl: pandoc/csl/gost-r-7-0-5-2008-numeric.csl

0.3 Pdf output format

toc: true # Table of contents toc-depth: 2 lof: true # List of figures lot: true # List of tables
fontsize: 12pt linestretch: 1.5 papersize: a4 documentclass: scrreprt ## l18n polyglossia
polyglossia-lang: name: russian options: - spelling=modern - babelshorthands=true
polyglossia-otherlangs: name: english ## l18n babel babel-lang: russian babel-otherlangs:
english ## Fonts mainfont: PT Serif romanfont: PT Serif sansfont: PT Sans monofont: PT Mono
mainfontoptions: Ligatures=TeX romanfontoptions: Ligatures=TeX sansfontoptions:
Ligatures=TeX,Scale=MatchLowercase monofontoptions: Scale=MatchLowercase,Scale=0.9 ##
Biblatex biblatex: true biblio-style: “gost-numeric” biblatexoptions: - parenttracker=true -
backend=biber - hyperref=auto - language=auto - autolang=other* - citestyle=gost-numeric ##
Pandoc-crossref LaTeX customization figureTitle: “” tableTitle: “Таблица” listingTitle:
“Листинг” lofTitle: “Список иллюстраций” lotTitle: “Список таблиц” lolTitle:
“Листинги” ## Misc options indent: true header-includes: -

- **keep figures where there are in the text**

- # keep figures where there are in the text ---

1 Цель работы

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

2 Задание

1. Рассмотреть самый простой пример программы на языке ассемблера NASM. Hello world!
2. Изучить способности транслятора NASM.
3. Скомпилировать файл формата .asm
4. Скомпоновать данный файл и запустить.
5. Провести самостоятельную работу согласно задачам из лаб. работы

3 Теоретическое введение

3.1 Основные принципы работы компьютера

Основными функциональными элементами любой электронно-вычислительной машины (ЭВМ) являются центральный процессор, память и периферийные устройства (рис. 5.1). 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. Структурная схема ЭВМ.

Основной задачей процессора является обработка информации, а также организация координации всех узлов компьютера. В состав центрального процессора (ЦП) входят следующие устройства:

1. арифметико-логическое устройство — выполняет логические и арифметические действия, необходимые для обработки информации, хранящейся в памяти;
2. устройство управления — обеспечивает управление и контроль всех устройств компьютера;
3. 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*. 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):
4. RAX, RCX, RDX, RBX, RSI, RDI — 64-битные
5. EAX, ECX, EDX, EBX, ESI, EDI — 32-битные

6. AX, CX, DX, BX, SI, DI — 16-битные
7. AH, AL, CH, CL, DH, DL, BH, BL — 8-битные

Таким образом можно отметить, что вы можете написать в своей программе, например, такие команды (mov – команда пересылки данных на языке ассемблера): `mov ax, 1` `mov eax, 1`

Другим важным узлом ЭВМ является оперативное запоминающее устройство. 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.

В состав ЭВМ также входят периферийные устройства, которые можно разделить на:

1. *устройства внешней памяти*, которые предназначены для долговременного хранения больших объёмов данных (жёсткие диски, твердотельные накопители, магнитные ленты);
2. *устройства ввода-вывода*, которые обеспечивают взаимодействие ЦП с внешней средой.

3.1.1 Ассемблер и язык ассемблера

Язык ассемблера (assembly language, сокращённо asm) — машинноориентированный 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. Заметим, что получить полный доступ к ресурсам компьютера в современных архитектурах нельзя, самым низким уровнем работы прикладной программы является обращение напрямую к ядру ОС.

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.

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.

3.1.1.1 Процесс создания и обработки программы на языке ассемблера

Процесс создания ассемблерной программы можно изобразить в виде следующей схемы

В процессе создания ассемблерной программы можно выделить четыре шага:

1. 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`.
2. 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`
3. 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 `map`.
4. 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.

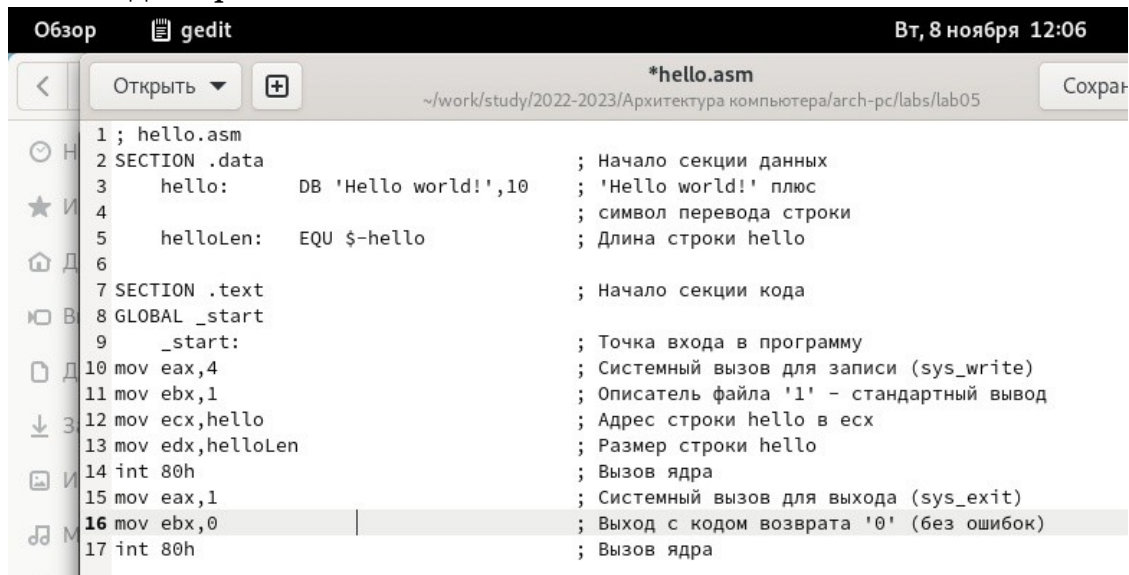
4 Выполнение лабораторной работы

1. Откроем каталог для работы с программами на языке ассемблера `NASM` и назовем его `asm`. Затем перейдя в данный каталог, создадим текстовый файл с названием `hello.asm` и откроем его с помощью текстового редактора `gedit`. (рис. 1)

```
[marie@fedora lab05]$ mkdir asm
[marie@fedora lab05]$ cd /asm
bash: cd: /asm: Нет такого файла или каталога
[marie@fedora lab05]$ touch hello.asm
[marie@fedora lab05]$ gedit hello.asm
```

Figure 1: Рис. 1

- Открыв hello.asm в gedit введем туда указанный в лабораторной работе текст, соблюдая синтаксис ассемблера NASM. Далее скомпилируем текст программы "Hello World", написав терминальную команду указанную на скриншоте. Таким образом, текст программы преобразован в объектный код, который записался в файл hello.o. С помощью команды ls проверим, что объектный файл действительно создан. (рис. 2)



```
1 ; hello.asm
2 SECTION .data
3     hello:    DB 'Hello world!',10
4               ; Начало секции данных
5               ; 'Hello world!' плюс
6               ; символ перевода строки
7     helloLen: EQU $-hello
8               ; Длина строки hello
9
10 SECTION .text
11 GLOBAL _start
12 _start:
13     mov eax,4
14     mov ebx,1
15     mov ecx,hello
16     mov edx,helloLen
17     int 80h
18     mov eax,1
19     int 80h
20     mov ebx,0
21     int 80h
22     ; Начало секции кода
23     ; Точка входа в программу
24     ; Системный вызов для записи (sys_write)
25     ; Описатель файла '1' - стандартный вывод
26     ; Адрес строки hello в ecx
27     ; Размер строки hello
28     ; Вызов ядра
29     ; Системный вызов для выхода (sys_exit)
30     ; Выход с кодом возврата '0' (без ошибок)
31     ; Вызов ядра
```

Figure 2: Рис. 2

- Далее применим команду, которая скомпилирует исходный файл hello.asm в obj.o (опция -o позволяет задать имя объектного файла, в данном случае obj.o), при этом формат выходного файла будет elf, и в него будут включены символы для отладки (опция -g), кроме того, будет создан файл листинга list.lst (опция -l) и с помощью команды ls проверим, что все файлы успешно созданы. Для получения списка форматов объектного файла пропишем nasm -hf. (рис.3 - 4)

```
marie@fedora:~/work/study/2022-2023/Архитектура компьют...
* Проверка изменений...
* Установка пакетов...

[marie@fedora lab05]$ nasm -f elf hello.asm
if: fatal: more than one input file specified: elf

[marie@fedora lab05]$ cd/asm
bash: cd/asm: Нет такого файла или каталога
[marie@fedora lab05]$ cd asm
[marie@fedora asm]$ tuoch asm
bash: tuoch: команда не найдена...
Аналогичная команда: 'touch'
[marie@fedora asm]$ touch asm
[marie@fedora asm]$ gedit asm
[marie@fedora asm]$ nasm -f elf hello.asm
nasm: fatal: unable to open input file `hello.asm' No such file or directory
[marie@fedora asm]$ ls
asm
[marie@fedora asm]$ touch hello.asm
[marie@fedora asm]$ gedit hello.asm
[marie@fedora asm]$ nasm -f elf hello.asm
[marie@fedora asm]$ ls
hello.asm  hello.o
[marie@fedora asm]$
```

Figure 3: Puc. 4


```
marie@fedora:~/work/study/2022-2023/Архитектура/asm$ ls
hello.asm  hello.o
[marie@fedora asm]$ nasm -o obj.o -f elf -g -l list.lst hello.asm
[marie@fedora asm]$ ls
hello.asm  hello.o  list.lst  obj.o
[marie@fedora asm]$ nasm -hf
Usage: nasm [-@ response_file] [options...] [--] filename
        nasm -v (or --v)

Options (values in brackets indicate defaults):

  -h          show this text and exit (also --help)
  -v (or --v) print the NASM version number and exit
  -@ file     response file; one command line option per line

  -o outfile  write output to outfile
  --keep-all output files will not be removed even if an error happens

  -Xformat    specify error reporting format (gnu or vc)
  -s          redirect error messages to stdout
  -Zfile      redirect error messages to file

  -M          generate Makefile dependencies on stdout
  -MG         d:o, missing files assumed generated
  -MF file    set Makefile dependency file
  -MD file    assemble and generate dependencies
  -MT file    dependency target name
  -MQ file    dependency target name (quoted)
  -MP         emit phony targets

  -f format   select output file format
```

Figure 4: Рис. 5

5. Затем, чтобы получить исполняемую программу необходимо объектный файл передать на обработку компоновщику с помощью необходимой команды. С помощью `ls` проверим, что исполняемый файл **hello** был создан. Затем преобразуем объектный файл `obj.o` в исполняемый, изменив его имя на `main`. Таким образом исполняемый файл имеет имя - `main`, а имя объектного файла, из которого он был создан - `obj.o`. (рис.5)

```
marie@fedora:~/work/study/2022-2023/Архитектура компьютера/arch-pc/labs/

phase          phase error during stabilization [off]
pragma         all warnings prefixed with "pragma-"
pragma-bad     malformed %pragma [off]
pragma-empty   empty %pragma directive [off]
pragma-na      %pragma not applicable to this compilation [off]
pragma-unknown unknown %pragma facility or directive [off]
ptr           non-NASM keyword used in other assemblers [on]
regsize       register size specification ignored [on]
unknown-warning unknown warning in -W/-w or warning directive [off]
user          %warning directives [on]
warn-stack-empty warning stack empty [on]
zeroing       RESx in initialized section becomes zero [on]
zext-reloc    relocation zero-extended to match output format [on]
other         any warning not specifically mentioned above [on]

--limit-X val  set execution limit X
passes        total number of passes [unlimited]
stalled-passes number of passes without forward progress [1000]
macro-levels  levels of macro expansion [10000]
macro-tokens  tokens processed during single-line macro expansion [10000000]
mmacros       multi-line macros before final return [100000]
rep           %rep count [1000000]
eval         expression evaluation descent [8192]
lines        total source lines processed [2000000000]

[marie@fedora asm]$ ld -m elf_i386 hello.o -o hello
[marie@fedora asm]$ ls
hello hello.asm hello.o list.lst obj.o
[marie@fedora asm]$ ld -m elf_i386 obj.o -o main
[marie@fedora asm]$ ls
hello hello.asm hello.o list.lst main obj.o
[marie@fedora asm]$
```

Figure 5: Puc. 6

- И в завершении основной части лабораторной работы запустим на выполнение созданный исполняемый файл, находящийся в текущем каталоге, набрав в командной строке : `./hello`. (рис.6)

```
[marie@fedora asm]$ ld -m elf_i386 hello.o -o hello
[marie@fedora asm]$ ls
hello hello.asm hello.o list.lst obj.o
[marie@fedora asm]$ ld -m elf_i386 obj.o -o main
[marie@fedora asm]$ ls
hello hello.asm hello.o list.lst main obj.o
[marie@fedora asm]$ ./hello
Hello world!
[marie@fedora asm]$
```

Figure 6: Puc. 7

- Приступим к заданиям для самостоятельной работы. И начнем с того, что в каталоге `~/work/arch-pc/lab05` создадим копию файла `hello.asm` с именем `lab05.asm`. (рис.7)


```

[marie@fedora asm]$ touch lab05.asm
[marie@fedora asm]$ ls
hello hello.asm hello.o lab05.asm list.lst main obj.o
[marie@fedora asm]$ gedit labo5.asm
[marie@fedora asm]$ gedit lab05.asm
[marie@fedora asm]$ nasm -f elf lab05.asm
lab05.asm:5: error: symbol `hello' not defined
[marie@fedora asm]$ nasm -f elf lab05.asm
[marie@fedora asm]$ ls
hello hello.asm hello.o lab05.asm lab05.o list.lst main obj.o
[marie@fedora asm]$

```

Figure 7: Puc. 8

8. Затем с помощью текстового редактора gedit внесем необходимые изменения в текст программы в файле lab5.asm так, чтобы вместо Hello world! на экран выводилась строка с моим ФИО. Сразу проведем необходимые преобразования объектных файлов в исполняемые, проверив все командой ls и запустим на выполнение созданный исполняемый файл. Задача успешно выполнена - выводится мое ФИО. Оттранслируем полученный текст программы lab5.asm в объектный файл. Выполним компоновку объектного файла и запустим получившийся исполняемый файл. (рис.8)

```

[marie@fedora asm]$ ld -m elf_i386 hello.o -o hello
[marie@fedora asm]$ ls
hello hello.asm hello.o list.lst obj.o
[marie@fedora asm]$ ld -m elf_i386 obj.o -o main
[marie@fedora asm]$ ls
hello hello.asm hello.o list.lst main obj.o
[marie@fedora asm]$ ./hello
Hello world!
[marie@fedora asm]$ touch lab05.asm
[marie@fedora asm]$ ls
hello hello.asm hello.o lab05.asm list.lst main obj.o
[marie@fedora asm]$ gedit labo5.asm
[marie@fedora asm]$ gedit lab05.asm
[marie@fedora asm]$ nasm -f elf lab05.asm
lab05.asm:5: error: symbol `hello' not defined
[marie@fedora asm]$ nasm -f elf lab05.asm
[marie@fedora asm]$ ls
hello hello.asm hello.o lab05.asm lab05.o list.lst main obj.o
[marie@fedora asm]$ nasm -o obj.o -f elf -g -l list.lst lab05.asm
[marie@fedora asm]$ ld -m elf_i386 lab05.o -o lab05
[marie@fedora asm]$ ls
hello hello.asm hello.o lab05 lab05.asm lab05.o list.lst main obj.o
[marie@fedora asm]$ ./lab05
Izvekova Marie Petrovna
[marie@fedora asm]$

```

Figure 8: Puc. 9

10. Скопируем файлы hello.asm и lab5.asm в локальный репозиторий в каталог ~ /work/study/2022-2023/“Архитектура компьютера”/archpc/labs/lab05/. Загрузим файлы на [Github](#) и создадим отчет о проделанной работе. (рис. **fig:012?**)

5 Выводы

В ходе данной работы я освоил процедуры компиляции и сборки программ, написанных на ассемблере NASM.

Список литературы