

Отчёт по лабораторной работе №4

Создание и процесс обработки программ на языке ассемблера NASM

Прокопьева Марина

Содержание

1	Цель работы	5
2	Теоретическая часть:	6
3	Выполнение работы	12
4	Выводы	17

Список иллюстраций

3.1	Название рисунка	12
3.2	Название рисунка	12
3.3	Название рисунка	13
3.4	Название рисунка	13
3.5	Название рисунка	14
3.6	Название рисунка	14
3.7	Название рисунка	14
3.8	Название рисунка	14
3.9	Название рисунка	15
3.10	Название рисунка	15
3.11	Название рисунка	15
3.12	Название рисунка	16
3.13	Название рисунка	16
3.14	Название рисунка	16
3.15	Название рисунка	16
3.16	Название рисунка	16

Список таблиц

1 Цель работы

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

2 Теоретическая часть:

Основные принципы работы компьютера

(ЭВМ) являются центральный процессор, память и периферийные устройства (рис. 4.1). 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 (системной) плате. 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 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 (арифме-

тические 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 (половинки 16-битных регистров).

Например, AH (high AX) — старшие 8 бит регистра AX, AL (low AX) — младшие 8 бит регистра AX.

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
```

Обе команды поместят в регистр AX число 1. Разница будет заключаться только в том, что вторая команда обнулит старшие разряды регистра EAX, то есть после выполнения второй команды в регистре EAX будет число 1. А первая команда оставит в старших разрядах регистра EAX старые данные. И если там были данные, отличные от нуля, то после выполнения первой команды в регистре EAX будет какое-то число, но не 1. А вот в регистре AX будет число 1. 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. 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. Данный алгоритм позволяет выполнить хранящуюся в ОЗУ программу. Кроме того, в зависимости от команды при её выполнении могут проходить не все этапы. Более подробно введение о теоретических основах архитектуры ЭВМ см. в [9; 11].

Ассемблер и язык ассемблера

Язык ассемблера (assembly language, сокращённо asm) — машинно-ориентированный 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 операционной системы. Именно на этом уровне и работают программы, написанные на ассемблере. Но в отличие от языков высокого уровня ассемблерная программа содержит только тот код, который ввёл программист. Таким образом язык ассемблера — это язык, с помощью которого понятным для человека образом пишутся команды для процессора. Следует отметить, что процессор понимает не команды ассемблера, а последовательности из нулей и единиц — машинные коды. До появления языков ассемблера программистам приходилось писать программы, используя только лишь машинные коды, которые были крайне сложны для запоминания, так как представляли собой числа, записанные в двоичной или шестнадцатеричной системе счисления. Преобразование или трансляция команд языка ассемблера в исполняемый машинный код осуществляется специальной программой транслятором — Ассемблер. Программы, написанные на языке ассемблера, не уступают в качестве и скорости программам, написанным на машинном языке, так как транслятор просто переводит мнемонические обозначения команд в последовательности бит (нулей и единиц). Используемые мнемоники обычно одинаковы для всех процессоров одной архитектуры или семейства архитектур (среди широко известных — мнемоники процессоров и контроллеров x86, ARM, SPARC, PowerPC, M68k). Таким образом для каждой архитектуры существует свой ассемблер и, соответственно, свой язык ассемблера. Наиболее распространёнными ассемблерами для архитектуры x86 являются:

- для DOS/Windows: Borland Turbo Assembler (TASM), Microsoft Macro Assembler (MASM) и Watcom assembler (WASM);
- для GNU/Linux: gas (GNU Assembler), использующий AT&T-синтаксис, в отличие от большинства других популярных ассемблеров, которые используют Intel-синтаксис.

Более подробно о языке ассемблера см., например, в [10]. В нашем курсе будет использоваться ассемблер NASM (Netwide Assembler) [7; 12; 14]. 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] Здесь мнемокод — непосредственно мнемоника инструкции процессору, которая является обязательной частью команды. Операндами могут быть числа, данные, адреса регистров или адреса оперативной памяти. Метка — это идентификатор, с которым ассемблер ассоциирует некоторое число, чаще всего адрес в памяти. Т.о. метка перед командой связана с адресом данной команды. Допустимыми символами в метках являются буквы, цифры, а также символы. Начинаться метка или идентификатор могут с буквы, ., _ и ?. Перед идентификаторами, которые пишутся как зарезервированные слова, нужно писать \$, чтобы компилятор трактовал его верно (так называемое экранирование). Максимальная длина идентификатора 4095 символов. Программа на языке ассемблера также может содержать директивы — инструкции, не переводящиеся непосредственно в машинные команды, а управляющие работой транслятора. Например, директивы используются для определения данных (констант и переменных) и обычно пишутся большими буквами.

Процесс создания и обработки программы на языке ассемблера

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 `map`. •

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. Из-за специфики программирования, а также по традиции для создания программ на языке ассемблера обычно пользуются утилитами командной строки (хотя поддержка ассемблера есть в некоторых универсальных интегрированных средах).

3 Выполнение работы

Программа Hello world!

Создала каталог для работы с программами на языке ассемблера NASM и перешла в созданный каталог (рис. 3.1):

```
meprokopjeva@dk4n60 ~ $ mkdir -p ~/work/arch-pc/lab04
meprokopjeva@dk4n60 ~ $ cd ~/work/arch-pc/lab04
meprokopjeva@dk4n60 ~/work/arch-pc/lab04 $
```

Рис. 3.1: Название рисунка

Создала текстовый файл с именем hello.asm, открыла этот файл с помощью любого текстового редактора (рис. 3.2).

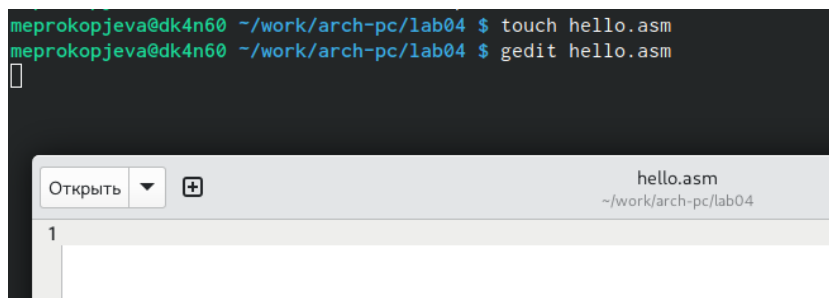
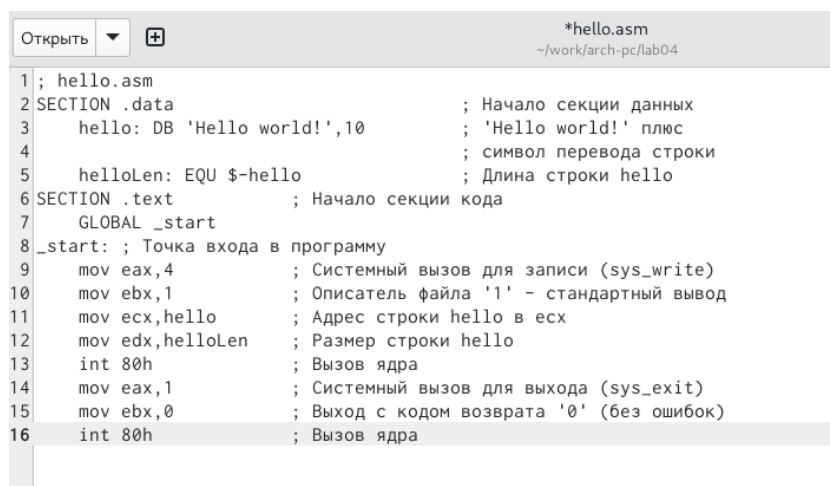


Рис. 3.2: Название рисунка

Ввела в него следующий текст (рис. 3.3).

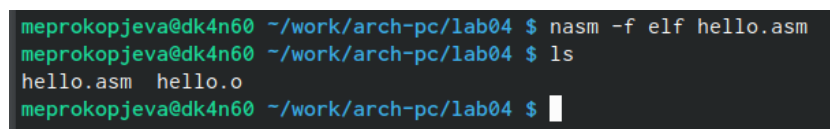


```
1 ; hello.asm
2 SECTION .data                                ; Начало секции данных
3     hello: DB 'Hello world!',10              ; 'Hello world!' плюс
4                                              ; символ перевода строки
5     helloLen: EQU $-hello                    ; Длина строки hello
6 SECTION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

Рис. 3.3: Название рисунка

Транслятор NASM

Компилирую приведенный выше текст и проверяю его (рис. 3.4).



```
meprokopjeva@dk4n60 ~/work/arch-pc/lab04 $ nasm -f elf hello.asm
meprokopjeva@dk4n60 ~/work/arch-pc/lab04 $ ls
hello.asm  hello.o
meprokopjeva@dk4n60 ~/work/arch-pc/lab04 $
```

Рис. 3.4: Название рисунка

Расширенный синтаксис командной строки NASM

Выполнила следующую команду:

```
nasm -o obj.o -f elf -g -l list.lst hello.asm
```

Данная команда 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).

С помощью команды ls проверила, что файлы были созданы (рис. 3.5).

```

meprokojjeva@dk4n60 ~/work/arch-pc/lab04 $ nasm -o obj.o -f elf -g -l list.lst hello.asm
meprokojjeva@dk4n60 ~/work/arch-pc/lab04 $ ls
hello.asm  hello.o  list.lst  obj.o
meprokojjeva@dk4n60 ~/work/arch-pc/lab04 $

```

Рис. 3.5: Название рисунка

Компоновщик LD

Выполните следующую команду: `ld -m elf_i386 hello.o -o hello` С помощью команды `ls` проверила что команда работает так как надо (рис. 3.6).

```

meprokojjeva@dk4n60 ~/work/arch-pc/lab04 $ ld -m elf_i386 hello.o -o hello
meprokojjeva@dk4n60 ~/work/arch-pc/lab04 $ ls
hello  hello.asm  hello.o  list.lst  obj.o
meprokojjeva@dk4n60 ~/work/arch-pc/lab04 $

```

Рис. 3.6: Название рисунка

Выполнила следующую команду: `ld -m elf_i386 obj.o -o main` Формат командной строки LD можно увидеть, набрав `ld --help`

```

meprokojjeva@dk4n60 ~/work/arch-pc/lab04 $ ld -m elf_i386 obj.o -o main
meprokojjeva@dk4n60 ~/work/arch-pc/lab04 $ ld --help
Использование ld [параметры] файл...
Параметры:
  -a КЛЮЧЕВОЕ СЛОВО                Управление общей библиотекой для совместимости с HP/UX
  -A АРХИТЕКТУРА, --architecture АРХИТЕКТУРА  Задать архитектуру
  -b ЦЕЛЬ, --format ЦЕЛЬ            Задать цель для следующих входных файлов

```

Рис. 3.7: Название рисунка

Запуск исполняемого файла

Запустила на выполнение созданный исполняемый файл, находящийся в текущем каталоге, можно, набрав в командной строке: `./hello` (рис. 3.8)

```

meprokojjeva@dk4n60 ~/work/arch-pc/lab04 $ ./hello
Hello world!
meprokojjeva@dk4n60 ~/work/arch-pc/lab04 $

```

Рис. 3.8: Название рисунка

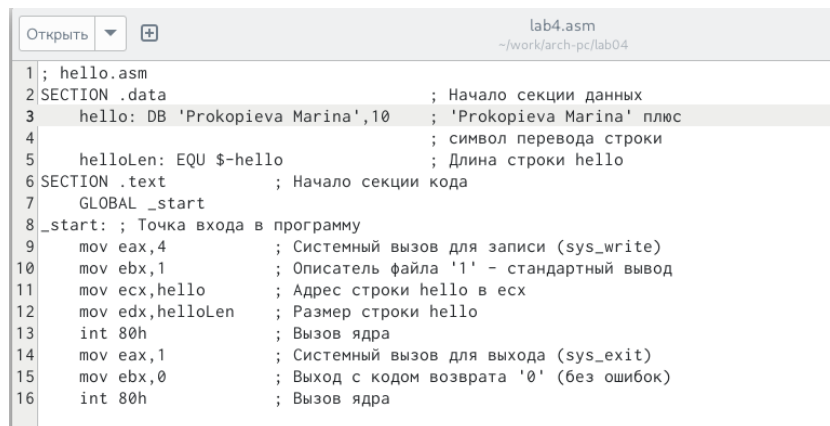
Самостоятельная работа

1 В каталоге ~/work/arch-pc/lab04 с помощью команды `cp` создала копию файла `hello.asm` с именем `lab4.asm` и проверила (рис. 3.9).

```
meprokojjeva@dk4n60 ~/work/arch-pc/lab04 $ cp hello.asm lab4.asm
meprokojjeva@dk4n60 ~/work/arch-pc/lab04 $ ls
hello  hello.asm  hello.o  lab4.asm  list.lst  main  obj.o
meprokojjeva@dk4n60 ~/work/arch-pc/lab04 $
```

Рис. 3.9: Название рисунка

2 С помощью любого текстового редактора внесла изменения в текст программы в файле `lab4.asm` так, чтобы вместо `Hello world!` на экран выводилась строка с моей фамилией и именем (рис. 3.10).



```
lab4.asm
~/work/arch-pc/lab04

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

Рис. 3.10: Название рисунка

3 Оттранслировала полученный текст программы `lab4.asm` в объектный файл. Выполнила компоновку объектного файла и запустила получившийся исполняемый файл (рис. 3.11, 3.12, 3.13).

```
meprokojjeva@dk4n60 ~/work/arch-pc/lab04 $ nasm -f elf lab4.asm
meprokojjeva@dk4n60 ~/work/arch-pc/lab04 $ ls
hello  hello.asm  hello.o  lab4.asm  lab4.o  list.lst  main  obj.o
meprokojjeva@dk4n60 ~/work/arch-pc/lab04 $
```

Рис. 3.11: Название рисунка

```
meprokopjeva@dk4n60 ~/work/arch-pc/lab04 $ ld -m elf_i386 lab4.o -o lab4
meprokopjeva@dk4n60 ~/work/arch-pc/lab04 $ ls
hello hello.asm hello.o lab4 lab4.asm lab4.o list.lst main obj.o
meprokopjeva@dk4n60 ~/work/arch-pc/lab04 $
```

Рис. 3.12: Название рисунка

```
meprokopjeva@dk4n60 ~/work/arch-pc/lab04 $ ./lab4
Prokopieva Marina
meprokopjeva@dk4n60 ~/work/arch-pc/lab04 $
```

Рис. 3.13: Название рисунка

4 Скопировала файлы hello.asm и lab4.asm в Ваш локальный репозиторий в каталог ~/work/study/2023-2024/“Архитектура компьютера”/arch-pc/labs/lab04/. Загрузила файлы на Github (рис. 3.14, 3.15, 3.16).

```
meprokopjeva@dk8n52 ~ $ cp ~/work/arch-pc/lab04/* ~/work/study/2023-2024/“Computer architecture”/arch-pc/labs/lab04
meprokopjeva@dk8n52 ~ $ cd ~/work/study/2023-2024/“Computer architecture”/arch-pc/labs/lab04
meprokopjeva@dk8n52 ~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04 $ ls
hello hello.asm hello.o lab4 lab4.asm lab4.o list.lst main obj.o presentation report
meprokopjeva@dk8n52 ~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab04 $
```

Рис. 3.14: Название рисунка

```
meprokopjeva@dk8n52 ~ $ cd ~/work/study/2023-2024/“Computer architecture”/arch-pc/labs
meprokopjeva@dk8n52 ~/work/study/2023-2024/Computer architecture/arch-pc/labs $ git add .
meprokopjeva@dk8n52 ~/work/study/2023-2024/Computer architecture/arch-pc/labs $ git commit -m 'Add files'
[master eb2ea0f] Add files
27 files changed, 400 insertions(+), 98 deletions(-)
create mode 100644 labs/lab03/report/L03_Prokopieva_Отчет_НБИ-02-23.pdf
create mode 100755 labs/lab04/hello
create mode 100644 labs/lab04/hello.asm
```

Рис. 3.15: Название рисунка

```
meprokopjeva@dk8n52 ~/work/study/2023-2024/Computer architecture/arch-pc/labs $ git push
Перечисление объектов: 41, готово.
Подсчет объектов: 100% (41/41), готово.
При сжатии изменений используется до 6 потоков
Сжатие объектов: 100% (33/33), готово.
Запись объектов: 100% (33/33), 1.23 МиБ | 21.67 МиБ/с, готово.
Всего 33 (изменений 7), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
remote: Resolving deltas: 100% (7/7), completed with 3 local objects.
remote: This repository moved. Please use the new location:
remote: git@github.com:MarinaPE02-23/study_2023-2024_arch-pc.git
To github.com:MarinaPE02-23/study_2023-2024_arch-pc.git
cb9a77d..eb2ea0f master -> master
meprokopjeva@dk8n52 ~/work/study/2023-2024/Computer architecture/arch-pc/labs $
```

Рис. 3.16: Название рисунка

4 Выводы

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