Лабораторная работа №4

НКАбд-02-23

Выборнов Дмитрий Валерьевич

1 Цель работы

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

2 Задание

1. Создание программы Hello world!
2. Использование транслятора NASM.
3. Работа с расширенным синтаксисом командной строки NASM.
4. Использование компоновщика LD.
5. Запуск исполняемого файла.
6. Задания для самостоятельной работы.

3 Теоретическое введение

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 с 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. В нашем курсе будет использоваться ассемблер NASM (Netwide Assembler). 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.

4 Выполнение лабораторной работы

4.1 Создание программы Hello world!

Создаю каталог для работы с программами на языке ассемблера NASM и перехожу в него:

```
dvvybornov@vvv-zenbook:~$ mkdir -p ~/work/arch-pc/lab04
dvvybornov@vvv-zenbook:~$ cd ~/work/arch-pc/lab04
```

Рис. 4.1: Первый шаг.

Создаю текстовый файл с именем hello.asm и открываю его при помощи nano:

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab04$ touch hello.asm
dvvybornov@vvv-zenbook:~/work/arch-pc/lab04$ nano hello.asm
```

Рис. 4.2: Второй шаг.

Ввожу в файл текст необходимой программы:

```
  GNU nano 6.2
; 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. 4.3: Третий шаг.

4.2 Использование транслятора NASM.

Ввожу команду для использования транслятора NASM и проверяю, что все нужные мне вайлы были созданы:

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab04$ nasm -f elf hello.asm
dvvybornov@vvv-zenbook:~/work/arch-pc/lab04$ ls
hello.asm  hello.o
```

Рис. 4.4: Четвёртый шаг.

4.3 Работа с расширенным синтаксисом командной строки NASM.

Выполняю команду, которая скомпилирует файл Hello.asm в obj.o и создаст файл листинга, и проверяю, что все нужные файлы созданы:

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab04$ nasm -o obj.o -f elf -g -l list.lst hello.asm
dvvybornov@vvv-zenbook:~/work/arch-pc/lab04$ ls
hello.asm  hello.o  list.lst  obj.o
```

Рис. 4.5: Пятый шаг.

4.4 Использование компоновщика LD.

Использую команду ld для создания исполняемого файла Hello и проверяю, что он был создан:

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab04$ ld -m elf_i386 hello.o -o hello
dvvybornov@vvv-zenbook:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  list.lst  obj.o
```

Рис. 4.6: Шестой шаг.

Выполняю команду, которая создаст файл main из объектного файла obj.o:

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab04$ ld -m elf_i386 obj.o -o main
dvvybornov@vvv-zenbook:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  list.lst  main  obj.o
```

Рис. 4.7: Седьмой шаг.

4.5 Запуск исполняемого файла.

Запускаю созданный исполняемый файл:

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab04$ ./hello
Hello world!
```

Рис. 4.8: Восьмой шаг.

4.6 Задания для самостоятельной работы

4.6.1 №1

С помощью команды cp создаю копию файла hello.asm с именем lab4.asm:

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab04$ cp hello.asm lab4.asm
```

Рис. 4.9: Первое задание.

4.6.2 №2

Вношу необходимые изменения в файл lab4.asm:

```
 GNU nano 6.2
; lab4.asm
SECTION data                                ; Начало секции данных
    lab4:    DB 'Dmitry Vybornov.', 10
                                            ; Символ перевода строки
    lab4Len: EQU $-lab4                     ; Длина строки

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,lab4    ; Адрес строки в ecx
    mov edx,lab4Len ; Размер строки
    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. 4.10: Второе задание.

4.6.3 №3

Транслирую полученный текст программы в объектный файл:

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab04$ nano lab4.asm
dvvybornov@vvv-zenbook:~/work/arch-pc/lab04$ nasm -f elf lab4.asm
```

Рис. 4.11: Первый шаг третьего задания.

Выполняю компоновку объектного файла и запускаю получившийся исполняемый файл:

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab04$ ld -m elf_i386 lab4.o -o lab4
dvvybornov@vvv-zenbook:~/work/arch-pc/lab04$ ./lab4
Dmitry Vybornov.
```

Рис. 4.12: Второй шаг третьего задания.

4.6.4 №4

Копирую файлы Hello.asm и lab4.asm в нужный репозиторий:

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab04$ cp ~/work/arch-pc/lab04/hello.asm ~/study_2023-2024_arch-pc/arch-pc/labs/lab04
dvvybornov@vvv-zenbook:~/work/arch-pc/lab04$ cp ~/work/arch-pc/lab04/lab4.asm ~/study_2023-2024_arch-pc/arch-pc/labs/lab04
```

Рис. 4.13: Первый шаг четвёртого задания.

Загружаю файлы на Github:

```
dvvybornov@vvv-zenbook:~/study_2023-2024_arch-pc/arch-pc/labs/lab04$ git add .
dvvybornov@vvv-zenbook:~/study_2023-2024_arch-pc/arch-pc/labs/lab04$ git commit -m "Lab04 update"
[master 0beae90] Lab04 update
 2 files changed, 38 insertions(+)
 create mode 100644 arch-pc/labs/lab04/hello.asm
 create mode 100644 arch-pc/labs/lab04/lab4.asm
dvvybornov@vvv-zenbook:~/study_2023-2024_arch-pc/arch-pc/labs/lab04$ git push
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 4 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (7/7), 1.06 KiB | 135.00 KiB/s, done.
Total 7 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 2 local objects.
To github.com:DVybornov1/study_2023-2024_arch-pc.git
   9c5225c..0beae90  master -> master
```

Рис. 4.14: Второй шаг четвёртого задания.

5 Выводы

Выполнив эту лабораторную работу, я освоил процедуры компиляции и сборки программ, написанных на ассемблере NASM.