

Отчёт по лабораторной работе №5

Дисциплина: Архитектура компьютера

Даровских Александра Сергеевна

Содержание

1 Цель работы

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

2 Задание

1. Создание программы Hello world!
2. Работа с транслятором NASM.
3. Работа с расширенным синтаксисом командной строки NASM.
4. Работа с компоновщиком LD.
5. Запуск исполняемого файла.
6. Выполнение заданий для самостоятельной работы.

3 Теоретическое введение

Основными функциональными элементами любой ЭВМ являются центральный процессор, память и периферийные устройства. 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. Основной задачей процессора является обработка информации, а также организация координации всех узлов компьютера. В состав центрального процессора 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 (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): - 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 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. Периферийные устройства в составе ЭВМ: - устройства внешней памяти, которые предназначены для долговременного хранения больших объёмов данных. - 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. 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. 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 (assembly language, сокращённо asm) — машинно-ориентированный язык низкого уровня. 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 4.1 Создание программы Hello world!

С помощью команды `cd` перемещаемся в каталог, в котором будем работать(рис.@fig:001)

```
[asdarovskikh@fedora ~]$ cd work/study/2023-2024/Архитектура\ компьютера/arch-pc/labs/lab04
[asdarovskikh@fedora lab04]$
```

Перемещение между директориями

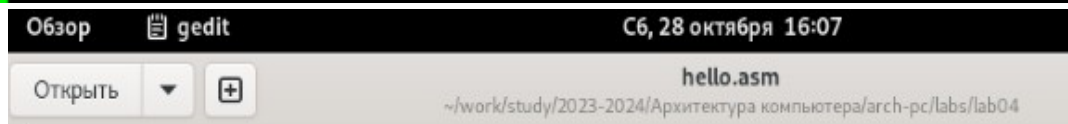
Создаем в текущем каталоге пустой текстовый файл `hello.asm` с помощью команды `touch` (рис.@fig:002).

```
[asdarovskikh@fedora lab04]$ touch hello.asm
```

Создание пустого файла

Открываем созданный файл в текстовом редакторе `gedit` (рис.??) (рис.??).

```
[asdarovskikh@fedora lab04]$ gedit hello.asm
```



Заполняем файл, вставляя в него программу для вывода “Hello word!”

```
Открыть ▼ + *hello.asm
~/work/study/2023-2024/Архитектура компьютера/arch-pc/labs/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
7 SECTION .text ; Начало секции кода
8     GLOBAL _start
9
10 _start: ; Точка входа в программу
11     mov eax,4 ; Системный вызов для записи (sys_write)
12     mov ebx,1 ; Описатель файла '1' - стандартный вывод
13     mov ecx,hello ; Адрес строки hello в ecx
14     mov edx,helloLen ; Размер строки hello
15     int 80h ; Вызов ядра
16
17     mov eax,1 ; Системный вызов для выхода (sys_exit)
18     mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
19     int 80h ; Вызов ядра
```

(рис.@fig:005).

4.2 4.2 Работа с транслятором NASM

Превращаем текст программы для вывода “Hello world!” в объектный код с помощью транслятора NASM, используя команду `nasm -f elf hello.asm`, ключ `-f` указывает транслятору `nasm`, что требуется создать бинарный файл в формате ELF (рис. ??). Далее проверяем правильность выполнения команды с помощью команды `ls`: действительно, создан файл “hello. O”.

```
[asdarovskikh@fedora lab04]$ nasm -f elf hello.asm
[asdarovskikh@fedora lab04]$ ls
hello.asm hello.o presentation report
```

4.3 4.3 Работа с расширенным синтаксисом командной строки NASM

Вводим команду, которая скомпилирует файл `hello.asm` в файл `obj.o`, при этом в файл будут включены символы для отладки (ключ `-g`), также с помощью ключа `-l 10` будет создан файл листинга `list.lst` (рис.@fig:007). Далее проверяем с помощью команды `ls` правильность выполнения команды.

```
[asdarovskikh@fedora lab04]$ nasm -o obj.o -f elf -g -l list.lst hello.asm
[asdarovskikh@fedora lab04]$ ls
hello.asm hello.o list.lst obj.o presentation report
```

Компиляция текста программы

4.4 4.4 Работа с компоновщиком LD

Передаем объектный файл `hello.o` на обработку компоновщику LD, чтобы получить исполняемый файл `hello` (рис.@fig:008). Ключ `-o` задает имя создаваемого исполняемого файла. Далее проверяем с помощью команды `ls` правильность выполнения команды.

```
[asdarovskikh@fedora lab04]$ ld -m elf_i386 hello.o -o hello
[asdarovskikh@fedora lab04]$ ls
hello hello.asm hello.o list.lst obj.o presentation report
[asdarovskikh@fedora lab04]$
```

Передача объектного файла на обработку компоновщику

Выполняем следующую команду (рис.@fig:009). Исполняемый файл будет иметь имя `main`, т.к. после ключа `-o` было задано значение `main`. Объектный файл, из которого собран этот исполняемый файл, имеет имя `obj.o`

```
[asdarovskikh@fedora lab04]$ ld -m elf_i386 obj.o -o main
[asdarovskikh@fedora lab04]$ ls
hello hello.asm hello.o list.lst main obj.o presentation report
[asdarovskikh@fedora lab04]$
```

Передача объектного файла на обработку компоновщику

4.5 4.5 Запуск исполняемого файла

Запускаем на выполнение созданный исполняемый файл hello (рис.@fig:010).

```
[asdarovskikh@fedora lab04]$ ./hello
Hello world!
[asdarovskikh@fedora lab04]$
```

Запуск исполняемого файла

4.6 4.6 Выполнение заданий для самостоятельной работы

С помощью команды `cp` создаем в текущем каталоге копию файла `hello.asm` с именем `lab04.asm` (рис. ??).

```
[asdarovskikh@fedora lab04]$ cp hello.asm lab04.asm
```

Создание копии файла

С помощью текстового редактора `gedit` открываем файл `lab04.asm` и вносим изменения в программу так, чтобы она выводила имя и фамилию. (рис. ??).

```
1 ; hello.asm
2 SECTION .data ; Начало секции данных
3     lab04: DB 'Aleksandra Darovskikh',10
4
5     lab04Len: EQU $-lab04 ; Длина строки lab04
6
7 SECTION .text ; Начало секции кода
8     GLOBAL _start
9
10 _start: ; Точка входа в программу
11     mov eax,4 ; Системный вызов для записи (sys_write)
12     mov ebx,1 ; Описатель файла '1' - стандартный вывод
13     mov ecx,lab04 ; Адрес строки lab04 в ecx
14     mov edx,lab04Len ; Размер строки lab
15     int 80h ; Вызов ядра
16
17     mov eax,1 ; Системный вызов для выхода (sys_exit)
18     mov ebx,0 ; Выход с кодом возврата '0' (без ошибок)
19     int 80h ; Вызов ядра
```

Изменение программы

Компилируем текст программы в объектный файл (рис. ??). Проверяем с помощью команды `ls`, что файл `lab04.o` создан.

```
[asdarovskikh@fedora lab04]$ nasm -f elf lab04.asm
[asdarovskikh@fedora lab04]$ ls
hello hello.asm hello.o lab04.asm lab04.o list.lst main obj.o presentation report
```

Компиляция текста программы

Передаем объектный файл lab04.o на обработку компоновщику LD, чтобы получить исполняемый файл lab4 (рис. ??).

```
[asdarovskikh@fedora lab04]$ ld -m elf_i386 lab04.o -o lab04
[asdarovskikh@fedora lab04]$ ls
hello hello.asm hello.o lab04 lab04.asm lab04.o list.lst main obj.o presentation report
```

Передача объектного файла на обработку компоновщику

Запускаем исполняемый файл lab4, на экран действительно выводятся мои имя и фамилия (рис.@fig:015).

```
[asdarovskikh@fedora lab04]$ ./lab04
Aleksandra Darovskikh
```

Запуск исполняемого файла

Так как по заданию нужно было создать каталог для работы с программами на языке ассемблера NASM, чего мною было не сделано, то я создаю другую директорию lab04 с помощью mkdir, прописывая полный путь к каталогу, в котором хочу создать эту директорию. Далее копирую из текущего каталога файлы, созданные в процессе выполнения лабораторной работы, с помощью утилиты cp, указывая вместо имени файла символ *, чтобы скопировать все файлы. Команда проигнорирует директории в этом каталоге, т. к. не указан ключ -r, это мне и нужно (рис. ??).

Проверяю с помощью утилиты ls правильность выполнения команды.

```
[asdarovskikh@fedora lab04]$ mkdir ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab04
[asdarovskikh@fedora lab04]$ cp * ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab04
cp: не указан -r; пропускается каталог 'presentation'
cp: не указан -r; пропускается каталог 'report'
[asdarovskikh@fedora lab04]$ ls ~/work/study/2023-2024/Архитектура\ компьютера/arch-pc/lab04
hello hello.asm hello.o lab04 lab04.asm lab04.o list.lst main obj.o
```

Создании копии файлов в новом каталоге

Удаляем лишние файлы в текущем каталоге с помощью команды rm, ведь копии файлов остались в другой директории (рис. ??).


```
[asdarovskikh@fedora lab04]$ rm hello hello.o lab04 lab04.o list.lst main obj.o
[asdarovskikh@fedora lab04]$ ls
hello.asm lab04.asm presentation report
```

Удаление лишних файлов в текущем каталоге

С помощью команд `git add .` и `git commit` добавляем файлы на GitHub, комментируя действие как добавление файлов для лабораторной работы №5 (рис.@fig:018).

```
[asdarovskikh@fedora lab04]$ git add .
[asdarovskikh@fedora lab04]$ git commit -m "Add fales for lab04"
[master 2d48278] Add fales for lab04
2 files changed, 38 insertions(+)
```

Добавление файлов на GitHub

Отправляем файлы на сервер с помощью команды `git push` (рис.@fig:019).

```
[asdarovskikh@fedora lab04]$ git push
Перечисление объектов: 9, готово.
Подсчет объектов: 100% (9/9), готово.
При сжатии изменений используется до 2 потоков
Сжатие объектов: 100% (6/6), готово.
Запись объектов: 100% (6/6), 1.04 КиБ | 1.04 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:asdarovskikh/study_2023-2024_arhpc.git
 acbd711..2d48278 master -> master
```

Отправка файлов

5 Выводы

При выполнении данной лабораторной работы я освоила процедуры компиляции и сборки программ, написанных на ассемблере NASM.

6 Список литературы