Отчёт по лабораторной работе №4

Дисциплина: архитектура компьютеров

Симонова Виктория Игоревна

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

2 Задание

Здесь приводится описание задания в соответствии с рекомендациями методического пособия и выданным вариантом.

3 Теоретическое введение

Основными функциональными элементами любой электронно-вычислительной машины (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, преоб-

разование (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 (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 долговременного 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]

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. # Выполнение лабораторной работы

3.1 Программа Hello world!

Создание каталога для работы с программами на языке ассемблера NASM с помощью команды mkdir и проверка выполнения команды. (рис. [3.1]).

```
[visimonova@fedora ~]$ mkdir -p ~/work/arch-pc/lab04
[visimonova@fedora ~]$ cd ~/work
[visimonova@fedora work]$ ls
arch-pc  study
[visimonova@fedora work]$
```

Рис. 3.1: Создание каталога

Перехожу в директорию ~/work/arch-pc и создайю текстовый файл с именем hello.asm (рис. [3.2]).

```
[visimonova@fedora work]$ cd ~/work/arch-pc/lab04
[visimonova@fedora lab04]$ touch hello.asm
[visimonova@fedora lab04]$ ls
hello.asm
[visimonova@fedora lab04]$
```

Рис. 3.2: Создание текстового файла

Открываю созданный файл с помощью текстового редактора и ввожу текст (рис. [3.3]).

```
; 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. 3.3: Введение текста

3.2 Транслятор NASM

Превращаю 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. [3.4]).

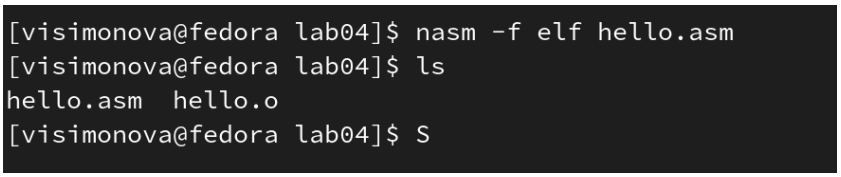


Рис. 3.4: Компиляция текста текста

3.3 Расширенный синтаксис командной строки NASM

Ввожу команду nasm -o obj.o -f elf -g -l list.lst hello.asm. Данная команда компилирует исходный файл в 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). (рис. [3.5]).

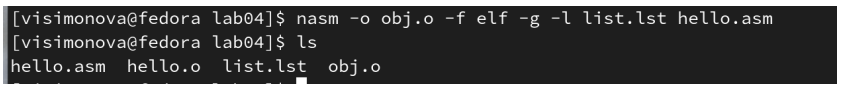


Рис. 3.5: Компиляция текста

3.4 Компоновщик LD

Передаю объектный файл на обработку компановщику LD, получаю исполняемый файл hello(рис. [3.6]).

Рис. 3.6: Передача файла

Выполняю команду ld -m elf_i386 obj.o -o main Исполняемый файл будет иметь имя main, т.к. после ключа -o было задано значение main. Объектный файл имеет имя obj.o. (рис. [3.7]).

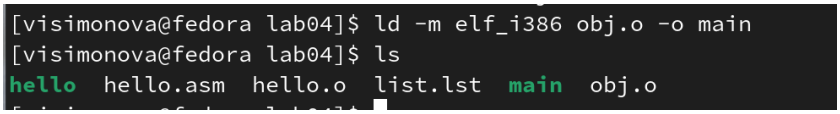
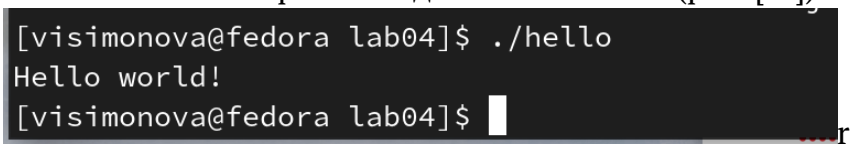


Рис. 3.7: Передача файла на обработку компановщику

3.5 Запуск исполняемого файла

Запускаю на выполнение созданный исполняемый файл, находящийся в текущем каталоге . На экран выводятся Hello word! (рис. [**??**]).



r # Задание для самостоятельной работы

Перехожу в каталог ~/work/arch-pc/lab04, создаю копию файла hello.asm с именем lab4.asm с помощью команды cp. (рис. [3.8]).

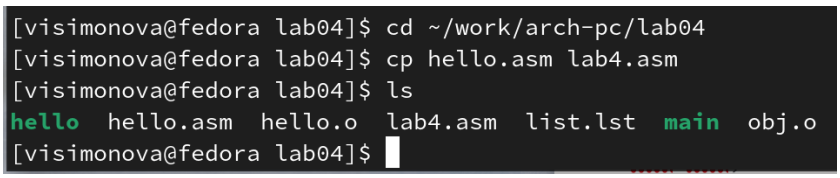


Рис. 3.8: Копирование файла

С помощью текстового редактора открываю файл lab4.asm и вношу изменения в программу так, чтобы она выводила мои имя и фамилию.(рис. [3.9]).

```
; lab4.asm
SECTION .data                                ; Начало секции данных
    hello:     DB 'Victoriia Simonova',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. 3.9: Изменение программа

Компилирую текст программы в объектный файл, передаю на обработку компановщику. Полученный исполняемый файл запускаю. На экран выводятся мои имя и фамилия.(рис. [3.10]).

```
[visimonova@fedora lab04]$ nasm -f elf lab4.asm
[visimonova@fedora lab04]$ ld -m elf_i386 lab4.o -o lab4
[visimonova@fedora lab04]$ ./lab4
Victoriia Simonova
```

Рис. 3.10: Компиляция, запуск и обработка исполняемого файла

Копирую файлы hello.asm и lab4.asm в локальный репозиторий в каталог ~/work/study/2023-2024/“Архитектура компьютера”/study_2023-2024_arh-pc/labs/lab04 (рис. [3.11]).

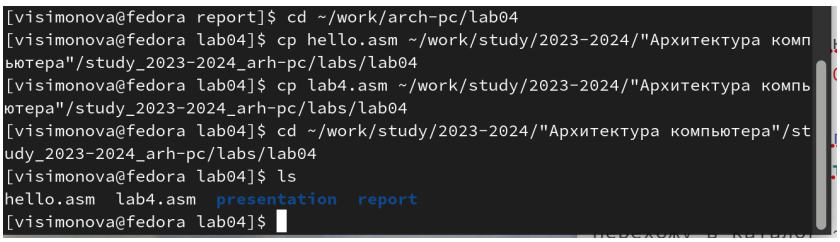


Рис. 3.11: Копирование файлов

Добавляю файлы на Github (рис. [3.12]).

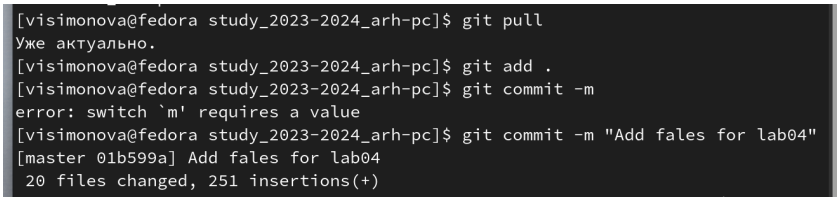


Рис. 3.12: Добавление файлов

Отправляю файлы на сервер с помощью команды git push (рис. [3.13]).

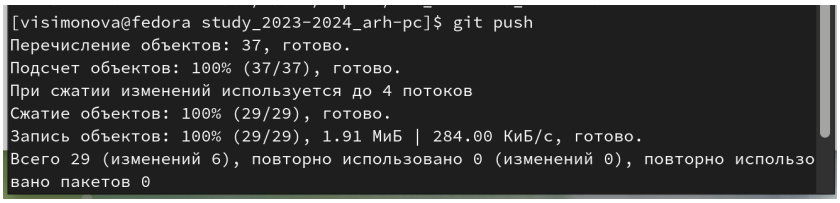


Рис. 3.13: Отправление файлов

4 Выводы

Я освоила процедуры компиляции и сборки программ, наптсанных на ассемблере NASM.

Список литературы

1. https://esystem.rudn.ru/pluginfile.php/1584628/mod_resource/content/1/%D0%9B%D0%B0%D