РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №4

дисциплина: Архитектура компьютера

Студент:Безлепкина Татьяна Игоревна

Группа:НКАбд-01-25

МОСКВА

2025 г.

Оглавление

1 Цель работы.....

2 Теоретическое введение.....

2.1Основные принципы работы компьютера.....

2.2 Ассемблер и язык ассемблера.....

2.3 Процесс создания и обработки программы на языке ассемблера.....

3 Выполнение лабораторной работы.....

3.1 Программа Hello world!.....

3.2 Транслятор NASM.....

3.3 Расширенный синтаксис командной строки NASM.....

3.4 Компоновщик LD.....

3.5 Запуск исполняемого файла.....

4 Задания для самостоятельной работы.....

5 Вывод .....

6 Список литературы.....

1 Цель работы

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

2 Теоретическое введение

2.1 Основные принципы работы компьютера

Любая ЭВМ состоит из трех основных элементов: центрального процессора (ЦП), памяти и периферийных устройств. Они взаимодействуют через общую шину — набор проводников на материнской плате.

Центральный процессор отвечает за обработку информации и управление узлами компьютера. В его состав входят:

Арифметико - логическое устройство (АЛУ): Выполняет арифметические и логические операции.

Устройство управления (УУ): Координирует работу всех устройств компьютера.

Регистры: Сверхбыстрая память внутри процессора для временного хранения данных и промежуточных результатов.

Знание регистров критически важно для программирования на ассемблере, так как большинство команд используют регистры в качестве операндов для пересылки данных и арифметико - логических преобразований.

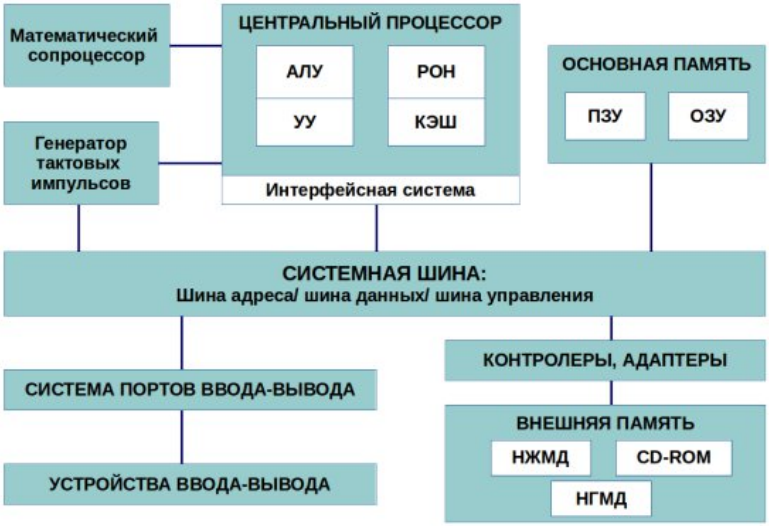

Рисунок 2.1.1 Структурная схема ЭВМ

Доступ к регистрам процессора осуществляется не по адресам, а по их уникальным именам.

В архитектуре x86 основные регистры общего назначения имеют иерархическую структуру и используются в зависимости от требуемой разрядности данных:

RAX, RCX, RDX, RBX, RSI, RDI — 64 - битные версии

EAX, ECX, EDX, EBX, ESI, EDI —младшие 32 бита 64-битных регистров

AX, CX, DX, BX, SI, DI — младшие 16 бит

AH/AL, CH/CL, DH/DL, BH/BL — старшие (High) и младшие (Low) 8 бит 16-битных регистров (AX, CX, DX, BX соответственно).

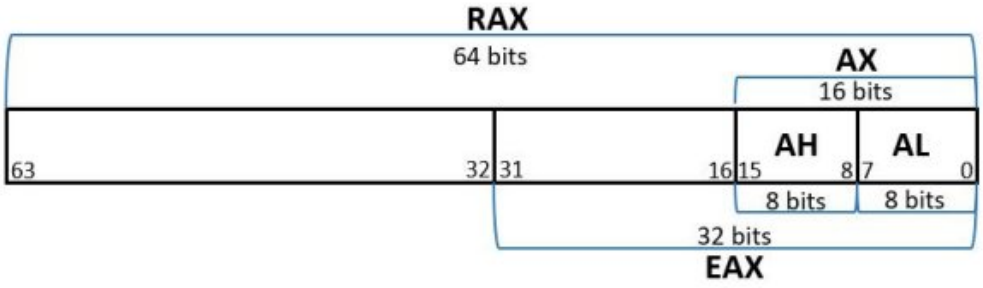

Рисунок 2.1.2 64-битный регистр процессора ‘RAX’

Оперативная память (ОЗУ) — это быстродействующее энергозависимое устройство для хранения программ и данных, с которыми процессор работает

непосредственно в текущий момент. Память состоит из пронумерованных ячеек, где номер ячейки является адресом данных.

Периферийные устройства делятся на:

Устройства внешней памяти (жёсткие диски, SSD) для долговременного хранения данных.

Устройства ввода-вывода для взаимодействия процессора с внешней средой.

Принцип программного управления заключается в том, что компьютер выполняет задачу как последовательность машинных команд (программу). Команда состоит из:

Операционной части — код операции.

Адресной части — данные или адреса данных для операции.

Командный цикл процессора — это последовательность шагов для выполнения каждой команды:

Формирование адреса следующей команды.

Считывание и расшифровка команды.

Выполнение команды.

Переход к следующей команде.

Этот алгоритм позволяет процессору выполнять программу, хранящуюся в ОЗУ.

2.2 Ассемблер и язык ассемблера

Язык ассемблера — это машинно-ориентированный язык низкого уровня, который обеспечивает наиболее полный доступ к аппаратным возможностям компьютера на уровне обращений к ядру ОС. В отличие от языков высокого уровня, программа на ассемблере содержит только код, написанный программистом.

Процессор выполняет программы в машинных кодах (последовательностях нулей и единиц). Ассемблер — это программа-транслятор, которая преобразует читаемые человеком мнемонические команды в машинный код. Программы на ассемблере по скорости работы не уступают программам в машинных кодах.

Каждая архитектура процессора (x86, ARM и др.) имеет свой язык ассемблера.

Формат команд в NASM (используется Intel-синтаксис):

text

[метка:] мнемокод [операнд {, операнд}] [; комментарий]

Мнемокод — мнемоника инструкции процессору (обязательна).

Операнды — данные, адреса регистров или памяти.

Метка — идентификатор, связанный с адресом команды в памяти.

Программа также может содержать директивы — инструкции для управления работой транслятора (например, для определения данных), которые не переводятся напрямую в машинные команды.

2.3 Процесс создания и обработки программы на языке ассемблера.

Процесс создания ассемблерной программы можно изобразить в виде следующей схемы

Рисунок 2.2.1 процесс создания ассемблерной программы.

3 Выполнение лабораторной работы

3.1 Программа Hello world!

Создам каталог для работы с программами на языке ассемблера NASM. Перейду в созданный каталог. Создам текстовый файл с именем hello.asm и открою этот файл с помощью любого текстового редактора, например, nano.

```
tibezlepkina1@localhost-live:~$ mkdir -p ~/work/arch-pc/lab04
tibezlepkina1@localhost-live:~$ cd ~/work/arch-pc/lab04
tibezlepkina1@localhost-live:~/work/arch-pc/lab04$ touch hello.asm
```

Рисунок 3.1.1 создание каталога с помощью(команды mkdir, -p — ключа, который означает:Создать все промежуточные директории, если они не существуют.Не выводить ошибку, если конечная директория уже существует).Переход в созданную директорию lab04 с помощью (команды cd).Создание файла hello.asm с помощью (команды touch).

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab04$ nano hello.asm
```

Рисунок 3.1.2 открытие файла с помощью текстового редактора (nano).

```
; 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
```

Рисунок 3.1.3 текст, необходимый вписать в файл.

3.2 Транслятор NASM

NASM превращает текст программы в объектный код. Для компиляции приведённого выше текста программы «Hello World» необходимо написать:

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab04$ nasm -f elf hello.asm
bash: nasm: command not found...
```

Рисунок 3.2.1 попытка превращения текста программы в объектный код.

Но для этого необходимо установить команду (nasm) предварительно.

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab04$ sudo dnf install nasm
Updating and loading repositories:
 Fedora 42 - x86_64 - Updates           100% | 310.4 KiB/s |   9.1 MiB |  00m30s
 Fedora 42 openh264 (From Cisco) - x86_ 100% |   2.6 KiB/s |   5.8 KiB |  00m02s
 Fedora 42 - x86_64                      91% |  70.6 KiB/s |  34.8 MiB |  00m44 Fedo
r Fedora 42 - x86_64       93% [================= ] | 451.7 KiB/s |  35.3 MiB |  00m0
 Fedora 42 - x86_64                     100% | 264.2 KiB/s |  35.4 MiB |  02m17s
Repositories loaded.
Package                 Arch    Version                   Repository          Size
Installing:
 nasm                   x86_64  2.16.03-3.fc42            fedora           2.5 MiB

Transaction Summary:
 Installing:         1 package
```

Рисунок 3.2.2 установка nasm.

Пробуем еще раз превратить текст программы в объектный код. NASM не запускают без параметров. Ключ -f указывает транслятору, что требуется создать бинарные файлы в формате ELF.C помощью команды (ls) проверю создание объектного файла.Имя файла hello.o

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab04$ nasm -f elf hello.asm
tibezlepkina1@localhost-live:~/work/arch-pc/lab04$ ls hello.o
hello.o
```

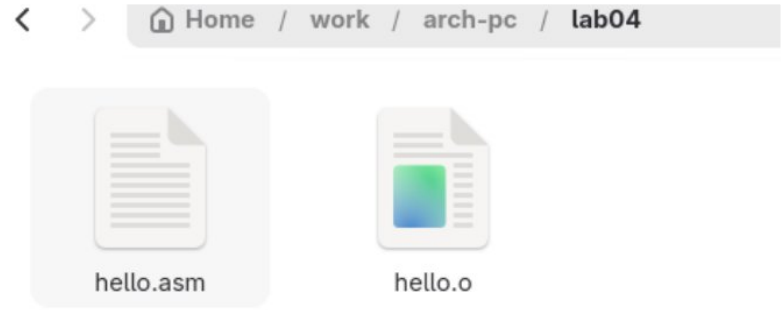

Рисунок 3.2.3 превращение текса программы в объектный код,а также проверка существования объектного файла.

3.3 Расширенный синтаксис командной строки NASM.

Выполним следущую команду.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 проверю, что файлы были созданы.

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab04$ nasm -o obj.o -f elf -g -l list.ls
t hello.asm
```

Рисунок 3.3.1 компиляция файла hello.asm в obj.o

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab04$ ls obj.o
obj.o
```

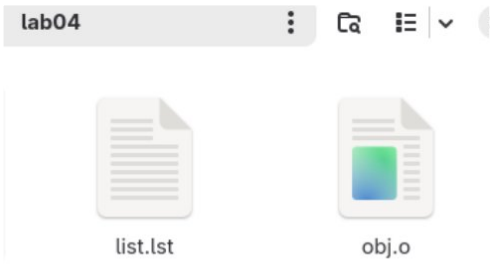

Рисунок 3.3.2 проверка создания файлов.

3.4 Компоновщик LD

Чтобы получить исполняемую программу, объектный файл необходимо передать на обработку компоновщику:

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab04$ ld -m elf_i386 hello.o -o hello
tibezlepkina1@localhost-live:~/work/arch-pc/lab04$ ld -m elf_i386 obj.o -o main
```

Рисунок 3.4.1 передача файла на обработку компановщика.

Исполняемый файл main , а объектный obj.o

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab04$ ls
hello  hello.asm  hello.o  list.lst  main  obj.o
```

Рисунок 3.4.2 проверка того,что файлы созданы.

3.5 Запуск исполняемого файла.

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab04$ ./hello
Hello world!
```

Рисунок 3.5.1 Запуск на выполнение созданный исполняемый файл.

4 Задания для самостоятельной работы.

Открываю файл с помощью текстового редактора (nano), вношу изменения,с помощью команды cp создаю копию файла hello.asm с именем lab4.asm, используя команду (nasm -f elf lab4.asm) превращаю текс программы в объектный код,передаю объектный файл компановщику с помощью команды (ld -m elf_i386 lab4.o -o hello),запускаю исполняемый файл.

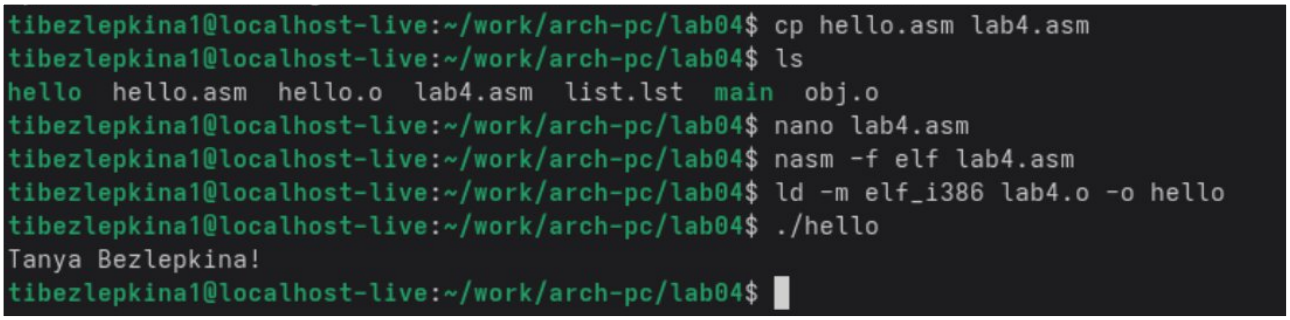

Рисунок 4.1 самостоятельная работа.

Убедившись в корректности работы программы, копирую рабочие файлы всвой локальный репозиторий.

5 Вывод.

В результате выполнения данной лабораторной я научилась компилировать и собирать программы,написанные на ассемблере NASM.

5. Список литературы.
    1. https://esystem.rudn.ru/mod/resource/view.php?id=1030495
    2. https://esystem.rudn.ru/mod/page/view.php?id=1030492
    3. https://esystem.rudn.ru/mod/resource/view.php?id=1030496
    4. https://esystem.rudn.ru/mod/resource/view.php?id=1030548