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: Ибрагимов Гаджимурад Шамильевич

Группа: НКАбд-02-25

МОСКВА

2025 г.

Содержание

Цель работы.

Освоить базовые навыки работы с ассемблером.

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 арифметиче-
ские 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.

Онакомимся со структурой ЭВМ на следующем рисунке

Рис. 4.1. Структурная схема ЭВМ

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

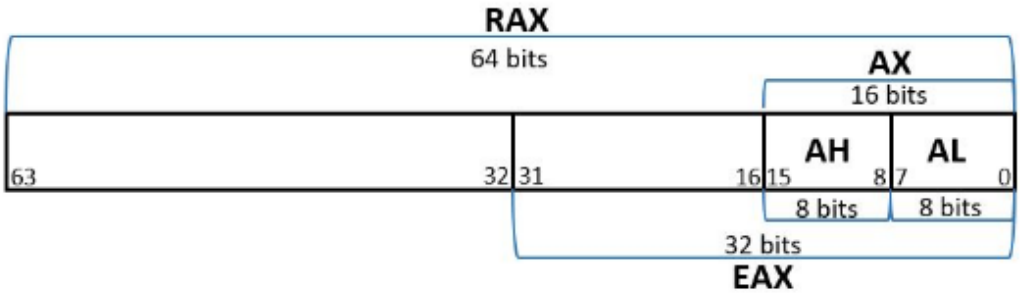
эти регистры чаще всего используются при написании программ):

• 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.



Язык ассемблера (assembly language, сокращённо asm) — ЭТО машинноориентированный 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. Преобраование и трансляция команд с языка ассемблера в исполняемый машинный код осуществляется специальной

программой транслятором — Ассемблер

Также затроним процесс создания и обработки программы на ассемблере

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

Выполнение лабораторной работы

Прописываем программу для вывода строки с символами.

```
Пакет                                                  Арх.          Версия                                                  Репозиторий                                 Размер
Установка:
 nasm                                                  x86_64        2.16.03-3.fc42                                          fedora                                     2.5 MiB

Сводка транзакции:
 Установка:         1 пакета

Общий размер входящих пакетов составляет 356 KiB. Необходимо загрузить 356 KiB.
После этой операции будут использоваться дополнительные 2 MiB (установка 2 MiB, удаление 0 B).
Is this ok [y/N]: Y^[[D^[[3~^[[Dy
Is this ok [y/N]: y
^[[B
[1/1] nasm-0:2.16.03-3.fc42.x86_64                                                                                                  100% |  30.7 KiB/s | 356.5 KiB |  00m12s
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
[1/1] Total                                                                                                                         100% |   3.6 KiB/s | 356.5 KiB |  01m40s
Выполнение транзакции
[1/3] Проверить файлы пакета                                                                                      100% |  52.0   B/s |   1.0   B |  00m00s
[2/3] Подготовить транзакцию                                                                                      100% |   0.0   B/s |   1.0   B |  00m03s
[3/3] Установка nasm-0:2.16.03-3.fc42.x86_64                                                                                 100% | 348.3 KiB/s |   2.5 MiB |  00m07s
Завершено!
```

Для начала установим расширение на наше устройство

Создаем каталог по данному пути и добавляем текстовый файл hello.asm.

```
GSIbragimov@GSI:~/work/study/2025-2026/ARCH/arch-pc/labs/lab04
~/work/study/2025-2026/ARCH/arch-pc/labs/lab04

GSIbragimov@GSI:~/work/study/2025-2026/ARCH/arch-pc/labs/lab04$ nasm -f elf hello.asm
GSIbragimov@GSI:~/work/study/2025-2026/ARCH/arch-pc/labs/lab04$ ls
hello.asm  hello.asm.save  hello.o
GSIbragimov@GSI:~/work/study/2025-2026/ARCH/arch-pc/labs/lab04$
```

Испольуя транслятор преобразуем текстовый файл в объектный код

Проверяем результат предыдущих шагов с помощью команды ls

Примечание. Текстовый редактор с кодом на яыке ассемблера

```
SECTION .data
        hello:      db "Hello, world!",0xa
                helloLen:   equ $ - hello
SECTION .text
        global _start

_start:
        mov eax, 4
        mov ebx, 1
        mov ecx, hello
        mov edx, helloLen
        int 0x80

        mov eax, 1
        mov ebx, 0
        int 0x80
```

Скомпилируем исходный hello.asm в obj.o ,а также проверим реультат с помощью команды ls

```
GSIbragimov@GSI:~/work/study/2025-2026/ARCH/arch-pc/labs/lab04$ nasm  -o obj.o -f elf -g -l list.lst hello.asm
GSIbragimov@GSI:~/work/study/2025-2026/ARCH/arch-pc/labs/lab04$ ls
hello.asm  hello.asm.save  hello.o  list.lst  obj.o
GSIbragimov@GSI:~/work/study/2025-2026/ARCH/arch-pc/labs/lab04$
```

```
GSIbragimov@GSI:~/work/study/2025-2026/ARCH/arch-pc/labs/lab04$ ld -m elf_i386 hello.o -o  hello
GSIbragimov@GSI:~/work/study/2025-2026/ARCH/arch-pc/labs/lab04$ ls
hello  hello.asm  hello.asm.save  hello.o  list.lst  obj.o
GSIbragimov@GSI:~/work/study/2025-2026/ARCH/arch-pc/labs/lab04$
```

```
GSIbragimov@GSI:~/work/study/2025-2026/ARCH/arch-pc/labs/lab04$ ld -m elf_i386 obj.o -o main
GSIbragimov@GSI:~/work/study/2025-2026/ARCH/arch-pc/labs/lab04$ ls
hello  hello.asm  hello.asm.save  hello.o  list.lst  main  obj.o
GSIbragimov@GSI:~/work/study/2025-2026/ARCH/arch-pc/labs/lab04$
```

Дальше в работу вступает компановщик ld

Также выполняем следующую команду

Исполняемый файл наывается main

```
GSIbragimov@GSI:~/work/study/2025-2026/ARCH/arch-pc/labs/lab04$ ./hello
Hello, world!
```

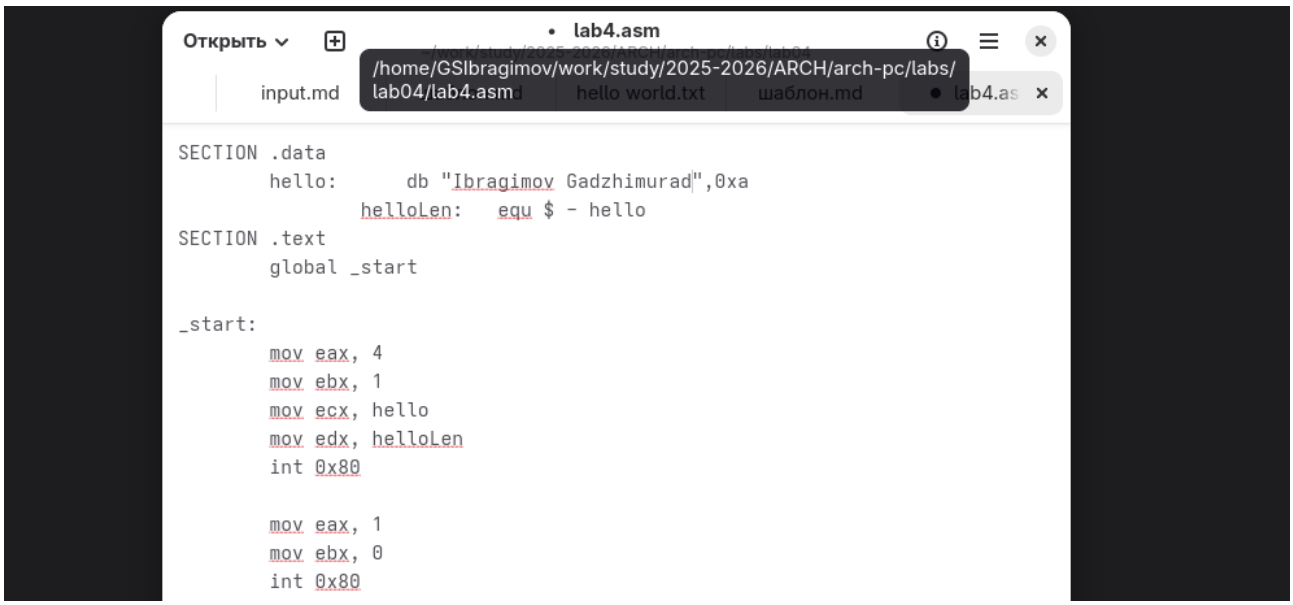
Запускаем нашу программу

Лабораторная работа окончена)

Выполнение самостоятельной работы

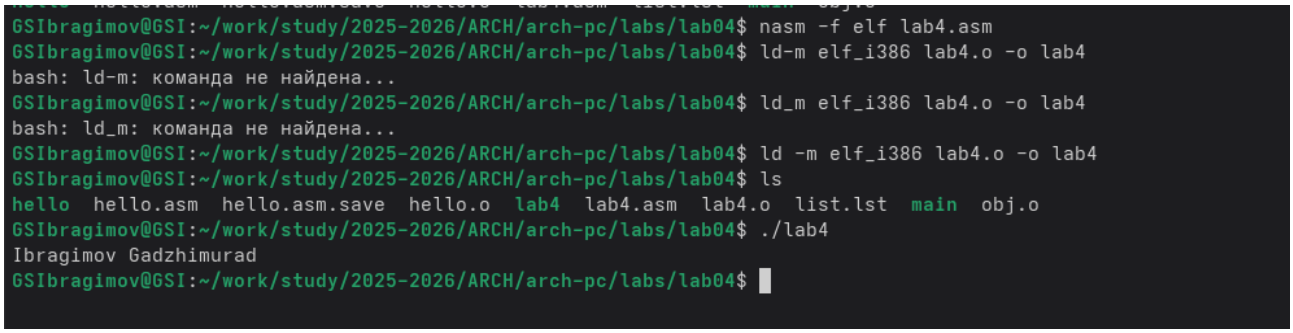
Создаем копию файла hello.asm и переименовываем его. Все действия происходят на локальной ветке репозитория для удобства. В дальнейшем все файлы, которые не отвечают требованиям будут перемещены из данного каталога. В принципе мне ничего не мешало содать локальную файловую ветку, как того просит задача , но я посмешил и заметил это только сейчас.

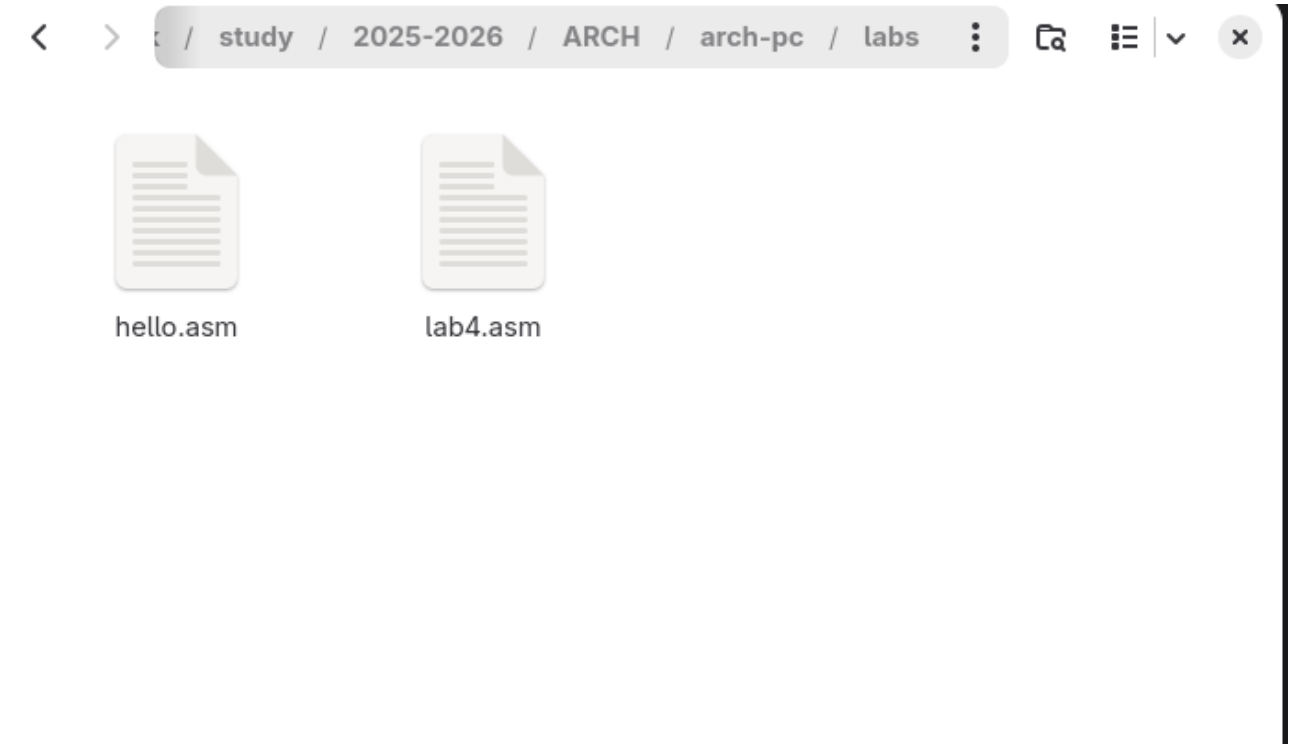
Дальше взаимодействуем с уже переименнованым файлом. Меняем строку для вывода



Переименованный файл и скорректированый файл

Далее выполняем все те действия с файлом, который мы могли заметить во время выполнению лабораторной работы.





Редактируем локальный репоиторий

Загрузим файлы на github

```
GSIbragimov@GSI:~/work/study/2025-2026/ARCH/arch-pc/labs/lab04$ git status
Текущая ветка: master
Эта ветка соответствует «origin/master».

Неотслеживаемые файлы:
  (используйте «git add <файл>...», чтобы добавить в то, что будет включено в коммит)
        ../lab03/
        ./
        ../lab04_temp/

индекс пуст, но есть неотслеживаемые файлы
(используйте «git add», чтобы проиндексировать их)
GSIbragimov@GSI:~/work/study/2025-2026/ARCH/arch-pc/labs/lab04$ git add .
GSIbragimov@GSI:~/work/study/2025-2026/ARCH/arch-pc/labs/lab04$ git commit -m 'feat(labs): add 2 .asm files'
[master e7655e4] feat(labs): add 2 .asm files
 2 files changed, 32 insertions(+)
 create mode 100644 labs/lab04/hello.asm
 create mode 100644 labs/lab04/lab4.asm
GSIbragimov@GSI:~/work/study/2025-2026/ARCH/arch-pc/labs/lab04$ git push
Перечисление объектов: 8, готово.
Подсчет объектов: 100% (8/8), готово.
При сжатии изменений используется до 2 потоков
Сжатие объектов: 100% (6/6), готово.
Запись объектов: 100% (6/6), 640 байтов | 71.00 КиБ/с, готово.
Total 6 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To github.com:Cklassklass/study_2025-2026_arh-pc.git
   d1e764d..e7655e4  master -> master
GSIbragimov@GSI:~/work/study/2025-2026/ARCH/arch-pc/labs/lab04$
```

Самостоятельная работа окончена

Выводы

В ходе выполнения данной лабораторной работы мы онакомились с основными принципами и командами яыка ассемблера. Запустили исполняемый файл, нацеленный на вывод строки на консоль, немного отредактировали код и увидели изменения.

Источники

Лабораторная работа №4. Создание и процесс обработки программ на языке ассемблера NASM - Демидова А. В.

“Архитектура компьютера” -Эндрю Таненбаум