Отчёт по лабораторной работе № 9

Дисциплина: архитектура компьютера

Кайнова Алина Андреевна

Содержание

# 1 Цель работы

Приобрест навыки написания программ с использованием подпрограмм и познакомиться с методами отладки при помощи GDB и с его основными возможностями.

# 2 Задание

1. Реализация подпрограмм в NASM
2. Отладка программ с помощью GDB
3. Добавление точек останова
4. Работа с данными программы в GDB
5. Обработка аргументов командной строки в GDB
6. Выполнение заданий для самостоятельной работы

# 3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам. GDB (GNU Debugger — отладчик проекта GNU) работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ. Отладчик не содержит собственного графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Однако для GDB существует несколько сторонних графических надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки. Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя. Команда run (сокращённо r) — запускает отлаживаемую программу в оболочке GDB. Команда kill (сокращённо k) прекращает отладку программы, после чего следует вопрос о прекращении процесса отладки. Если в ответ введено y (то есть «да»), отладка программы прекращается. Командой run её можно начать заново, при этом все точки останова (breakpoints), точки просмотра (watchpoints) и точки отлова (catchpoints) сохраняются. Для выхода из отладчика используется команда quit (или сокращённо q). Если есть файл с исходным текстом программы, а в исполняемый файл включена информация о номерах строк исходного кода, то программу можно отлаживать, работая в отладчике непосредственно с её исходным текстом. Чтобы программу можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом -g. Установить точку останова можно командой break (кратко b). Типичный аргумент этой команды — место установки. Его можно задать как имя метки или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка». Информацию о всех установленных точках останова можно вывести командой info (кратко i). Для того чтобы сделать неактивной какую-нибудь ненужную точку останова, можно воспользоваться командой disable. Обратно точка останова активируется командой enable. Если же точка останова в дальнейшем больше не нужна, она может быть удалена с помощью команды delete. Для продолжения остановленной программы используется команда continue (c). Выполнение программы будет происходить до следующей точки останова. В качестве аргумента может использоваться целое число N, которое указывает отладчику проигнорировать N − 1 точку останова (выполнение остановится на N-й точке). Команда stepi (кратко sI) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию. Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом. Если в программе встречается одинаковый участок кода, его можно оформить в виде подпрограммы, а во всех нужных местах поставить её вызов. При этом подпрограмма будет содержаться в коде в одном экземпляре, что позволит уменьшить размер кода всей программы. Для вызова подпрограммы из основной программы используется инструкция call, которая заносит адрес следующей инструкции в стек и загружает в регистр eip адрес соответствующей подпрограммы, осуществляя таким образом переход. Затем начинается выполнение подпрограммы, которая, в свою очередь, также может содержать подпрограммы. Подпрограмма завершается инструкцией ret, которая извлекает из стека адрес, занесённый туда соответствующей инструкцией call, и заносит его в eip. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией call.

# 4 Выполнение лабораторной работы

## 4.1 Реализация подпрограмм в NASM

Создаю каталог для выполнения данной лабораторной работы и в нём файл lab09-1.asm

Figure 1: Создание каталога и файла

Figure 1: Создание каталога и файла

Ввожу в этот файл текст программы с использованием подпрограммы из листинга 9.1

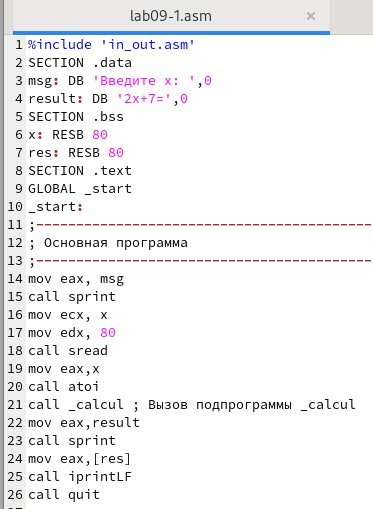


Figure 2: Ввод текста программы

Создаю исполняемый файл и проверяю его работу

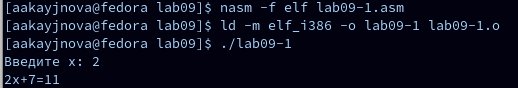


Figure 3: Создание и запуск исполняемого файла

Изменяю текст программы, добавив подпрограмму \_subcalcul в подпрограмму \_calcul для вычисления выражения f(g(x)), где x вводится с клавиатуры, f(x)=2x+7, g(x)=3x-1

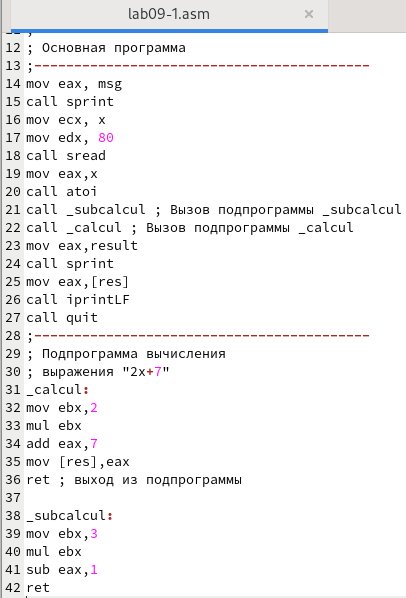


Figure 4: Изменение текста программы

Создаю исполняемый файл и проверяю его работу

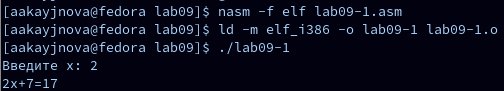


Figure 5: Создание и запуск исполняемого файла

## 4.2 Отладка программ с помощью GDB

Создаю файл lab09-2.asm и вставляю туда текст программы из листинга 9.2

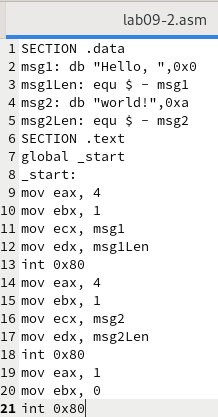


Figure 6: Ввод текста программы

Получаю исполняемый файл для работы с GDB с ключом ‘-g’

Figure 7: Получение исполняемого файла

Figure 7: Получение исполняемого файла

Загружаю исполняемый файл в отладчик GDB

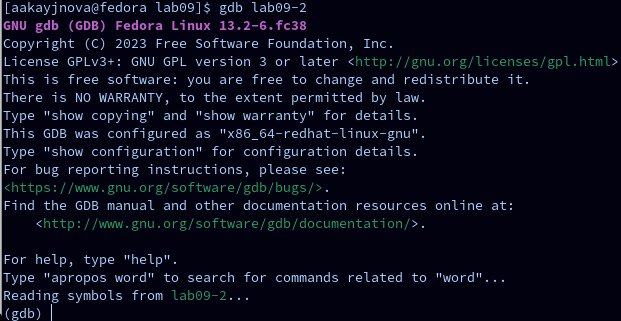


Figure 8: Загрузка исполняемого файла в отладчик

Проверяю работу программы, запустив её в GDB

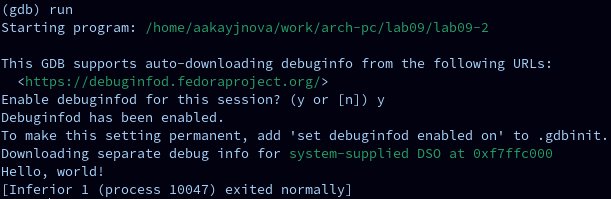


Figure 9: Проверка работы файла

Устанавливаю брейкпоинт на метку \_start и запускаю программу для более подробного анализа

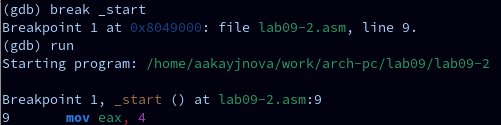


Figure 10: Установка брейкпоинта и запуск программы

Просматриваю дисассимилированный код программы начиная с метки \_start и переключаюсь на отображение команд с Intel’овским синтаксисом



Figure 11: Просмотр кода программы

В режиме ATT имена регистров начинаются с символа %, а имена операндов с $, в то время как в Intel используется уже привычный нам синтаксис.

Включаю режим псевдографики для более удобного анализа программы



Figure 12: Режим псевдографики

## 4.3 Добавление точек останова

Проверяю правильность установки точки останова по имени метки \_start, устанавливаю ещё одну точку по адресу mov ebx,0x0 и просматриваю информацию о всех установленных точках останова

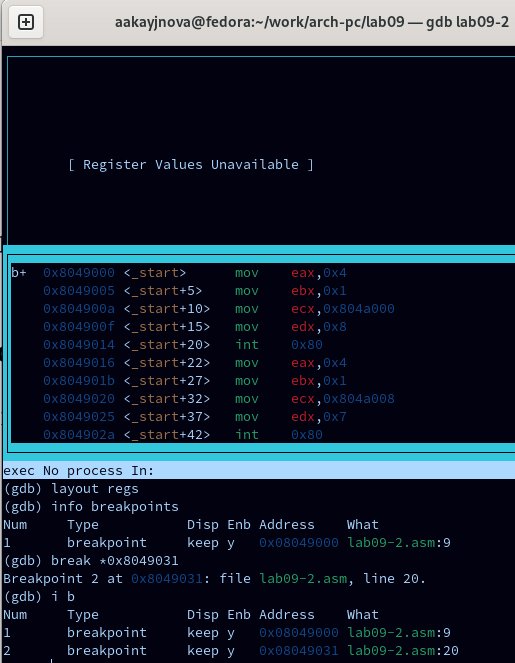


Figure 13: Установление точек останова и просмотр информации о них

## 4.4 Работа с данными программы в GDB

Выполняю 5 инструкций с помощью команды stepi и слежу за изменением значений регистров

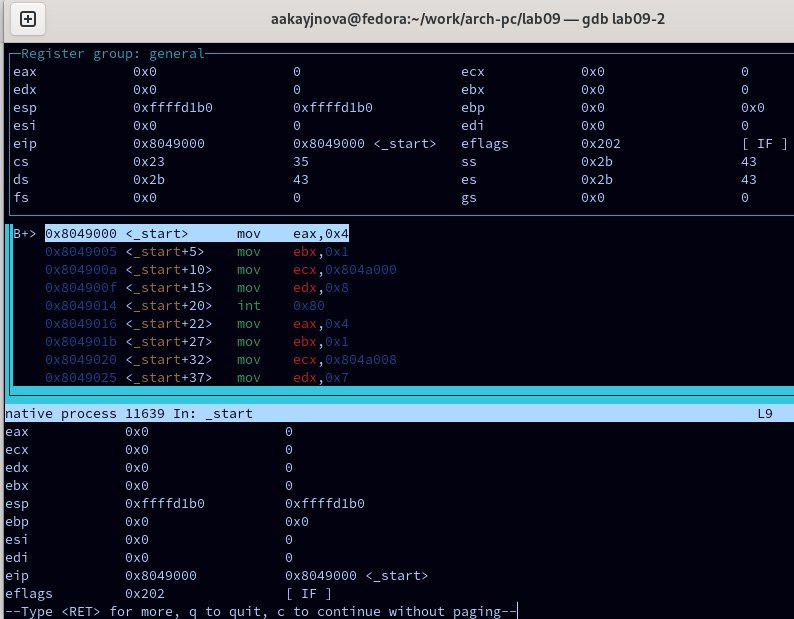


Figure 14: До использования команды stepi

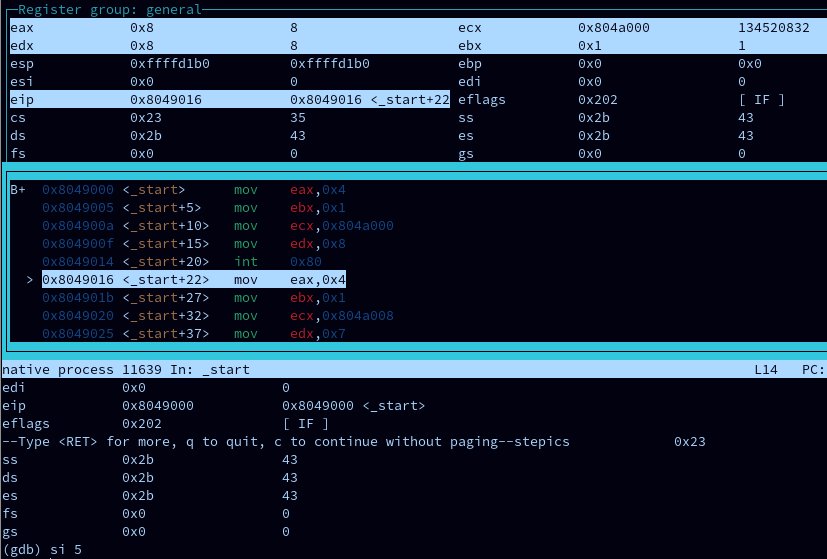


Figure 15: После использования команды stepi

Изменились значения регистров eax,ecx,edx,ebx.

Просматриваю значение переменной msg1 и msg2 по их адресам

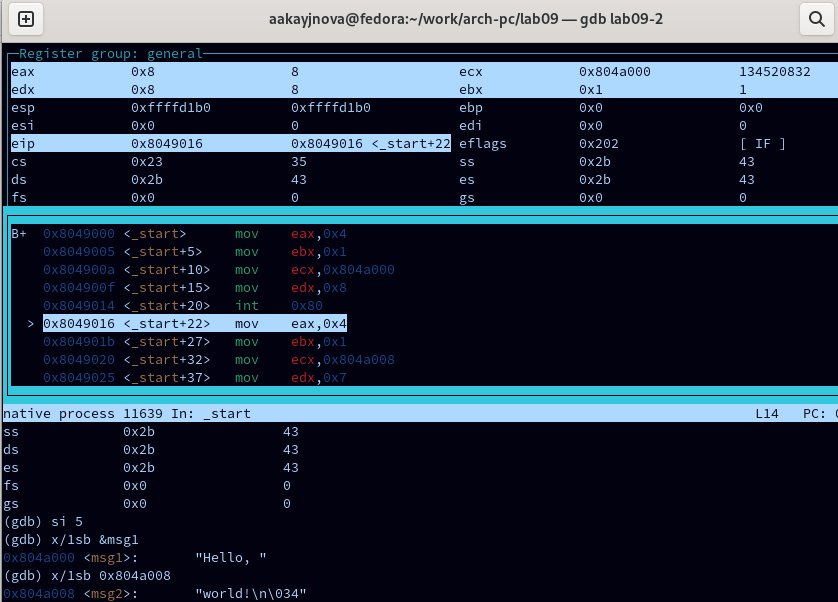


Figure 16: Просмотр значений переменных

Изменяю первый символ переменной msg1 и заменяю первый символ в переменной msg2

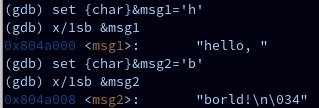


Figure 17: Изменение первого символа переменных

Вывожу в 16-теричном формате, в 2-ичном формате и в символьном виде значение регистра edx

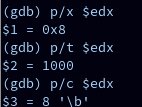


Figure 18: Вывод значения регистра в разных форматах

Изменяю значение регистра ebx согласно заданию

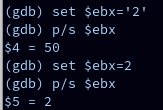


Figure 19: Изменение значения регистра

Разница вывода команд в том, что в первом случае мы переводим символ в его строковый вид, а во втором случае число в строковом виде не изменяется.

Завершаю выполнение программы и выхожу из GDB

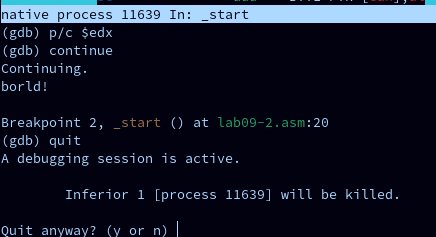


Figure 20: Завершение работы GDB

## 4.5 Обработка аргументов командной строки в GDB

Копирую файл lab8-2.asm в файл lab09-2.asm и создаю исполняемый файл

Figure 21: Копирование и создание файла

Figure 21: Копирование и создание файла

Загружаю исполняемый файл в отладчик GDB, указав необходимые аргументы

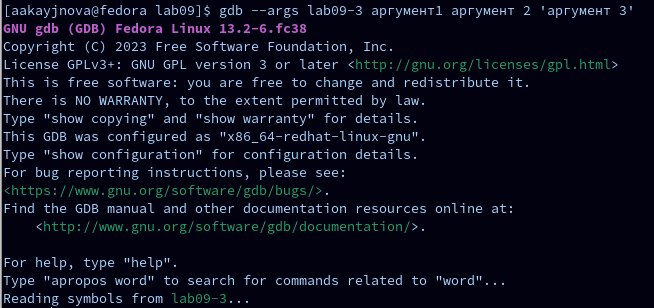


Figure 22: Загрузка файла с аргументами в отладчик

Устанавливаю точку останова перед первой инструкцией в программе и запускаю её

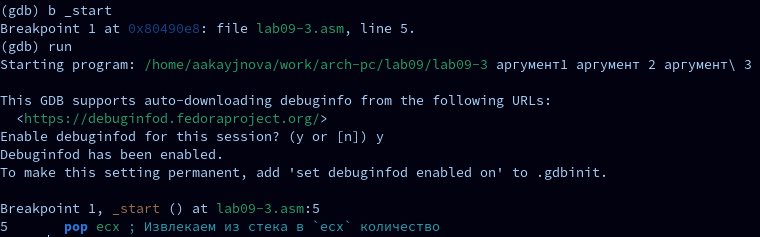


Figure 23: Установление точки останова и запуск программы

Просматриавю вершину стека и позиции стека по адресам

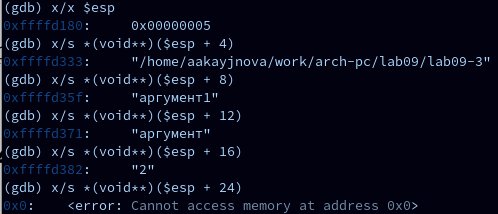


Figure 24: Просмотр значений, введённых в стек

Шаг изменения адреса равен 4, так как количество аргументов командной строки - 4.

## 4.6 Выполнение заданий для самостоятельной работы

Преобразовываю программу из лабораторной работы № 8 (задание № 1), реализовав вычисление значения функции f(x) как подпрограмму

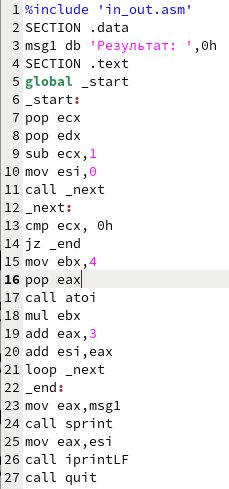


Figure 25: Написание кода программы

Запускаю код и проверяю правильность его работы

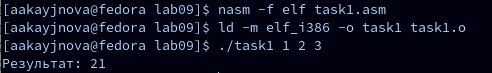


Figure 26: Запуск программы и проверка её вывода

Ввожу в файл task2.asm текст программы из листинга 9.3

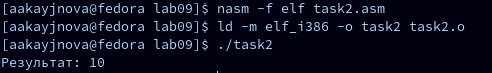


Figure 27: Ввод текста программы

Если программа работает верно, то на экран выведется “25”.

Создаю исполняемый файл и проверяю его

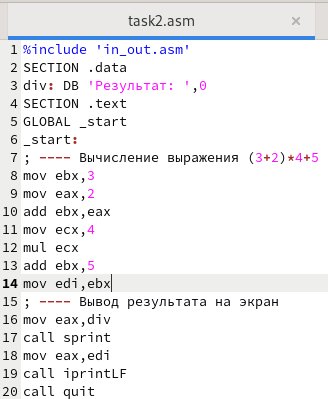


Figure 28: Создание и запуск исполняемого файла

Выводится неправильный ответ.

Получаю исполняемый файл для GDB, запускаю его и ставлю брейкпоинты на каждой инструкции, связанной с вычислениями. Прохожусь по каждому брейкпоинту и слежу за изменениями значений регистров.

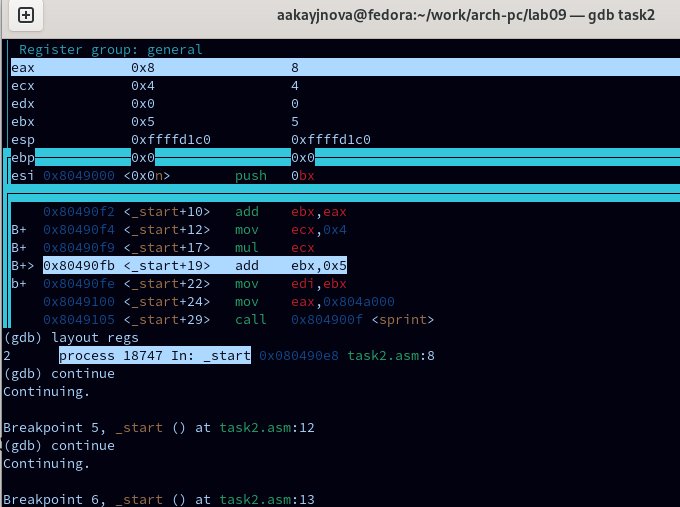


Figure 29: Поиск ошибки в GDB

Тут мы и получаем ошибку

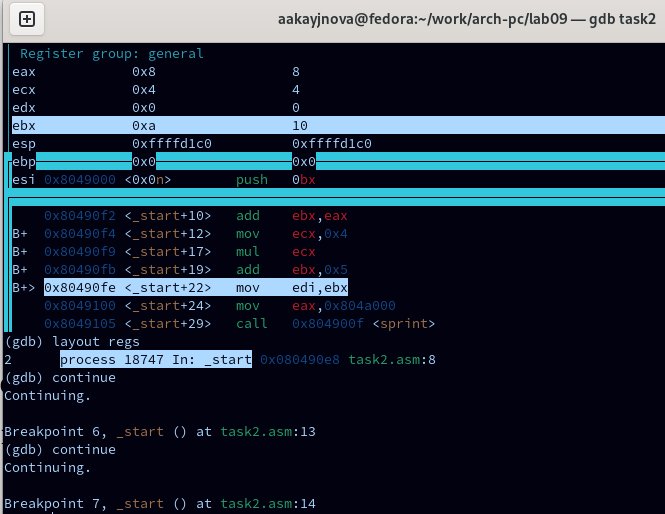


Figure 30: Ошибка в программе

Мы получаем неправильный ответ при выполнении инструкции mul ecx, во время которой ecx умножается на eax, то есть 4 умножается на 2, вместо правильного умножения на 5(то есть на регистр ebx). Причина в том, что инструкция dd ebx,eax (перед mov ecx,4) никак не связана с mul ecx, однако с mul ecx связана инструкция mov eax,2.

Исправлю ошибку, добавив необходимые инструкции

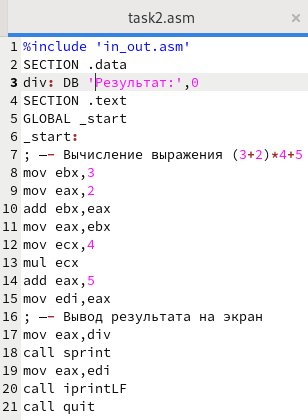


Figure 31: Правка текста программы

Создаю исполняемый файл и проверяю его

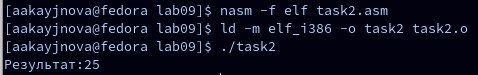


Figure 32: Создание и проверка файла

Выводится правильный результат, значит ошибка исправлена.

# 5 Выводы

В ходе данной лабораторной работы мы научились писать программы с использованием подпрограмм и ознакомились с методами отладки через GDB.

# Список литературы

1. https://esystem.rudn.ru/pluginfile.php/2089096/mod\_resource/content/0/%D0%9B%D0%B0%D0%B1%D0%BE%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BD%D0%B0%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%B0%20%E2%84%969.%20%D0%9F%D0%BE%D0%BD%D1%8F%D1%82%D0%B8%D0%B5%20%D0%BF%D0%BE%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D1%8B.%20%D0%9E%D1%82%D0%BB%D0%B0%D0%B4%D1%87%D0%B8%D0%BA%20..pdf