Отчёт по лабораторной работе №9

Дисциплина: Архитектура Компьютера

Курилко-Рюмин Е.М

Содержание

1 Цель работы **4**

2 Задание **5**

3 Теоретическое введение **6**

4 Выполнение лабораторной работы **10**

5 Выводы **27**

Список литературы **28**

Список иллюстраций

4.1 Создание и запуск исполняемого файла 11
4.2 Редактирование файла 11
4.3 Создание и запуск исполняемого файла 12
4.4 Редактирование файла 12
4.5 Создание исполняемого файла, отладчик gdb 13
4.6 Отладчик gdb 14
4.7 Установка метки 14
4.8 Дисассимилированный код программы с синтаксисом intel 15
4.9 Режим псевдографики 16
4.10 Режим псевдографики 16
4.11 Проверка точек остановки 16
4.12 Редактирование файла 17
4.13 Создание и запуск исполняемого файла 17
4.14 Изменение значений регистров и переменных 18
4.15 Изменение значений регистров и переменных 18
4.16 Содержимое регистров 19
4.17 Содержимое переменной, изменение в ней символов 19
4.18 Содержимое переменной, изменение в ней символов 20
4.19 Значения регистра edx 20
4.20 Изменение значений регистра 21
4.21 Загрузка файла в отладчик 21
4.22 Установка точки останова, запуск программы 22
4.23 Просмотр позиций стека 22
4.24 Копирование, редактирование файла 23
4.25 Создание и запуск исполяемого файла 23
4.26 Изменение значений регистров в gdb 24
4.27 Редактирование файла 24
4.28 Создание и запуск исполяемого файла 25

1 Цель работы

Целью данной работы является приобретение практического опыта в написании программ с использованием подпрограмм, а также знакомство с методами отладки при помощи gdb и его основными возможностями.

2 Задание

1. Общее ознакомление с подпрограммами в Nasm и отладкой при помощи gdb.
2. Реализация подпрограмм в NASM.
3. Отладка программ при помощи gdb.
4. Выполнение заданий для самостоятельной работы

3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае егоможно разделить на четыре этапа:

• обнаружение ошибки;

• поиск её местонахождения;

• определение причины ошибки;

• исправление ошибки.

Можно выделить следующие типы ошибок:

• синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка;

• семантические ошибки — являются логическими и приводят к тому, что программа запускается, отрабатывает, но не даёт желаемого результата;

• ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль).

Второй этап — поиск местонахождения ошибки. Некоторые ошибки обнаружить довольно трудно. Лучший способ найти место в программе, где находится ошибка, это разбить программу на части и произвести их отладку отдельно друг от друга.

Третий этап — выяснение причины ошибки. После определения местонахождения ошибки обычно проще определить причину неправильной работы программы.

Последний этап — исправление ошибки. После этого при повторном запуске

программы, может обнаружиться следующая ошибка, и процесс отладки начнётся заново.

Наиболее часто применяют следующие методы отладки:

• создание точек контроля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран — так называемые диагностические сообщения);

• использование специальных программ-отладчиков.

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. Пошаговое выполнение — это выполнение программы с остановкой после каждой строчки, чтобы программист мог проверить значения переменных и выполнить другие действия. Точки останова — это специально отмеченные места в программе, в которых программа-отладчик приостанавливает выполнение программы и ждёт команд. Наиболее популярные виды точек останова:

• Breakpoint — точка останова (остановка происходит, когда выполнение доходит до определённой строки, адреса или процедуры, отмеченной программистом);

• Watchpoint — точка просмотра (выполнение программы приостанавливается, если программа обратилась к определённой переменной: либо считала её значение, либо изменила его).

Точки останова устанавливаются в отладчике на время сеанса работы с кодом программы, т.е. они сохраняются до выхода из программы-отладчика или до смены отлаживаемой программы. GDB (GNU Debugger — отладчик проекта GNU) [1] 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. GDB может выполнять следующие действия:

• начать выполнение программы, задав всё, что может повлиять на её поведение;

• остановить программу при указанных условиях;

• исследовать, что случилось, когда программа остановилась;

• изменить программу так, чтобы можно было поэкспериментировать с устранением эффектов одной ошибки и продолжить выявление других.

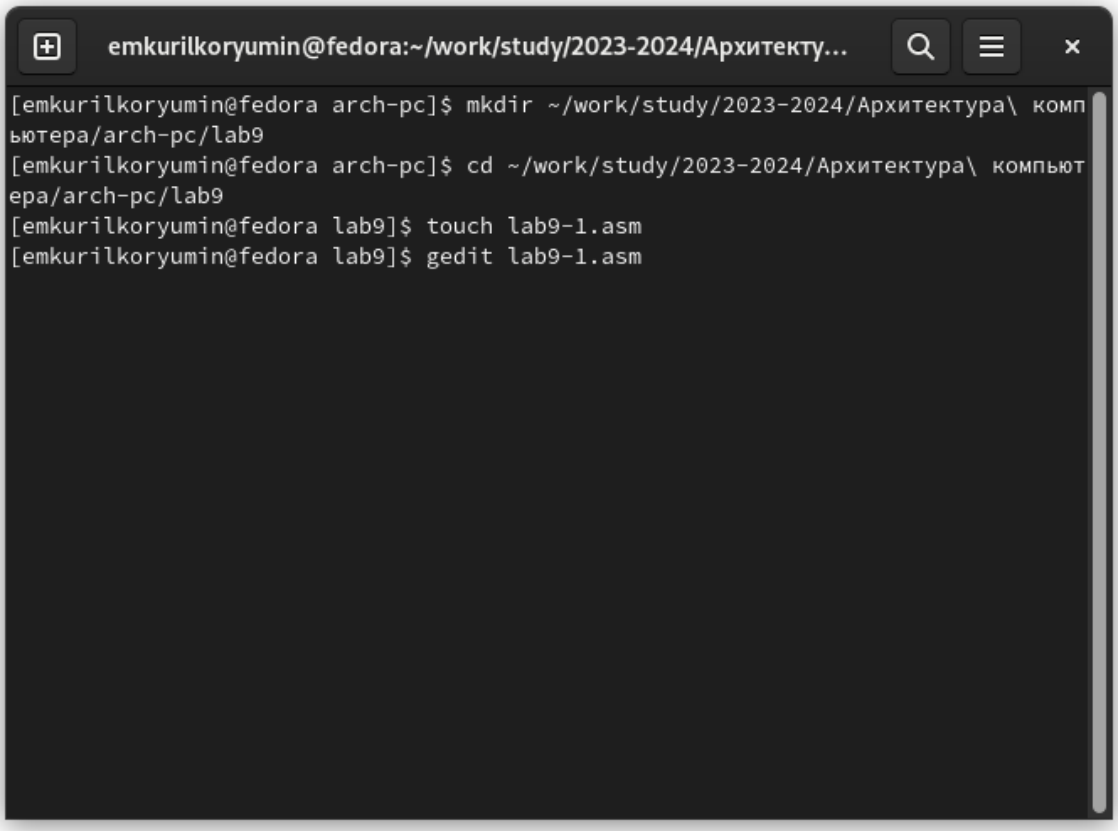
После запуска gdb выводит текстовое сообщение — так называемое «nice GDB logo». В следующей строке появляется приглашение (gdb) для ввода команд. Далее приведён список некоторых команд GDB. Команда run (сокращённо r) — запускает отлаживаемую программу в оболочке GDB. Если точки останова были заданы, то отладчик останавливается на соответствующей команде и выдаёт номер точки останова, адрес и дополнительную информацию — текущую строку, имя процедуры, и др. 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). Типич-

ный аргумент этой команды — место установки. Его можно задать как имя метки или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка». Для продолжения остановленной программы используется команда continue (c) (gdb). Выполнение программы будет происходить до следующей точки останова. В качестве аргумента может использоваться целое число ☒, которое указывает отладчику проигнорировать ☒ – 1 точку останова (выполнение остановится на ☒-й точке). Команда stepi (кратко sI) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию. Как уже упоминалось, отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Посмотреть содержимое регистров можно с помощью команды info registers (или i r). 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.

С помощью утилиты mkdir создаю директорию lab9 для выполнения соответствующей лабораторной работы. Перехожу в созданный каталог с помощью утилиты cd. С помощью touch создаю файл lab9-1.asm. Копирую в текущий каталог файл in_out.asm с помощью утилиты cp, ибо он будет использоваться в дальнейшем. Открываю созданный файл lab9-1.asm, вставляю в него следующую программу: (рис.1).



(image/1.1.p

width=70%}

Создаю исполняемый файл и запускаю его. (рис.2).

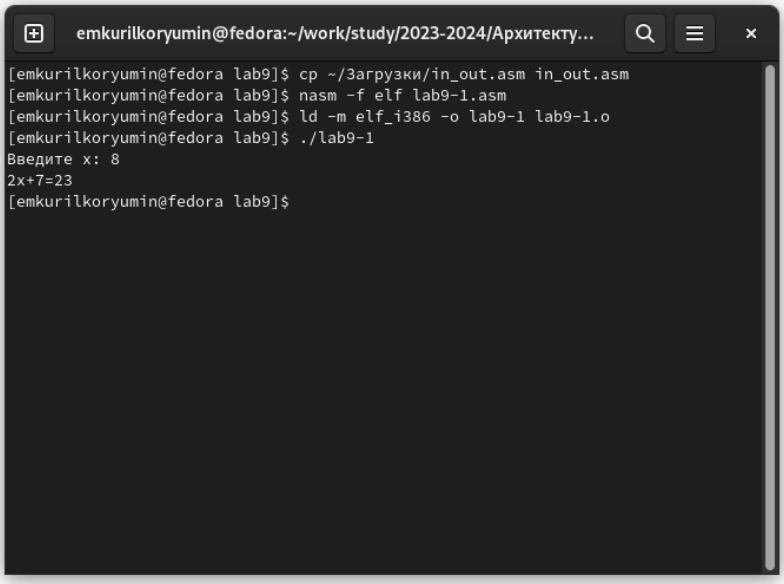


Рис. 4.1: Создание и запуск исполняемого файла

Добавляю подпрограмму subcalcul_, чтобы программа вычисляла значение f(g(x)). (рис.3).

Рис. 4.2: Редактирование файла

Создаю исполняемый файл и убеждаюсь в правильности его работы. (рис.4).

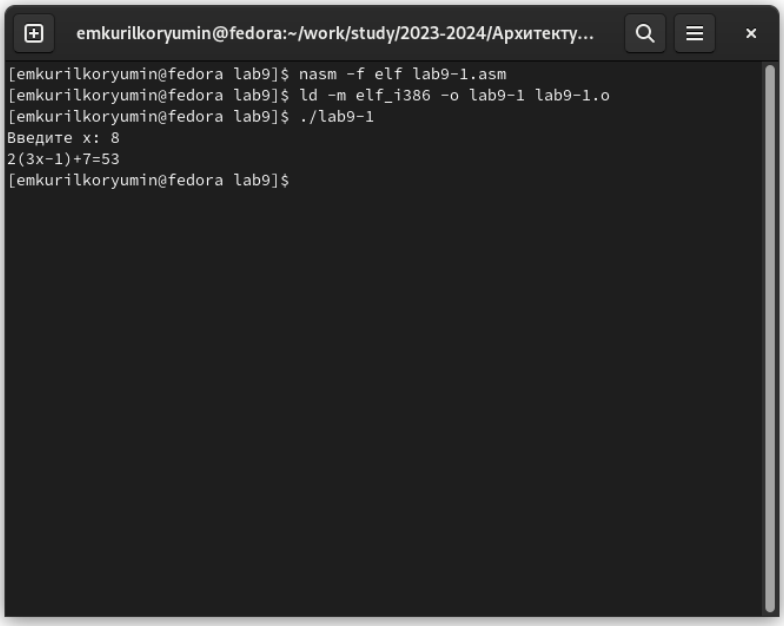


Рис. 4.3: Создание и запуск исполняемого файла

4.2) Отладка программ при помощи gdb.

Создаю файл lab09-2.asm и вношу в него следующий текст программы: (рис.5).

Рис. 4.4: Редактирование файла

Создаю исполняемый файл и загружаю его в отладчик gdb, запускаю программу с помощью команды run. (рис.6).

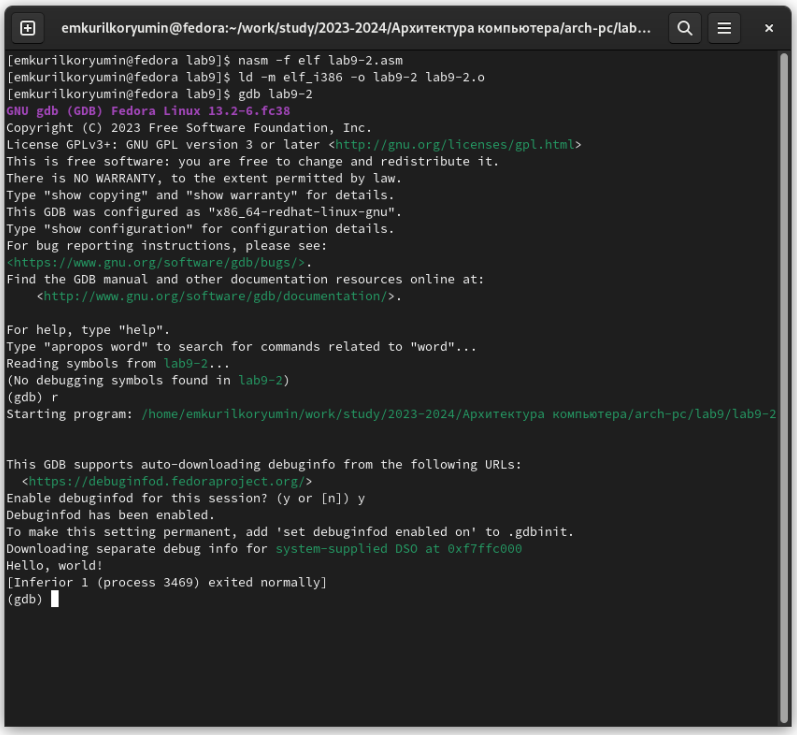


Рис. 4.5: Создание исполняемого файла, отладчик gdb

Убеждаюсь в правильности работы программы. (рис.7).

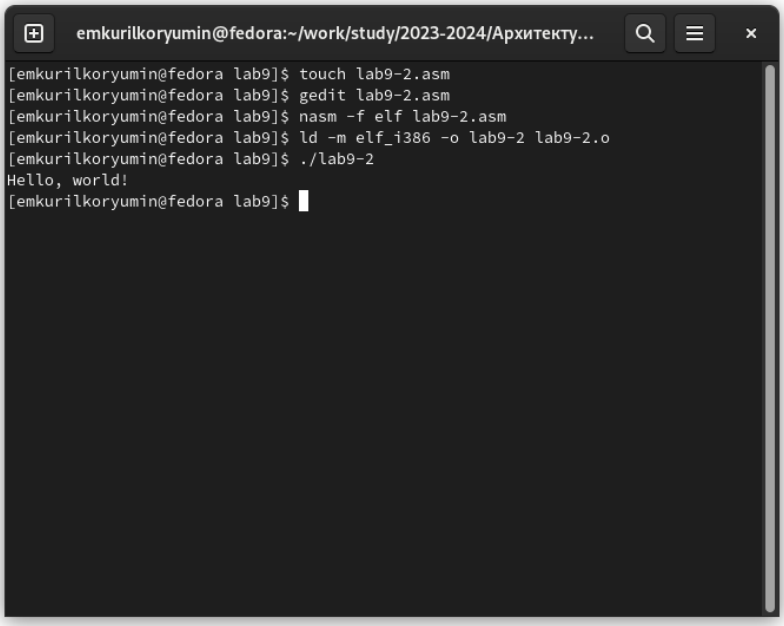


Рис. 4.6: Отладчик gdb

Устанавливаю метку _start и запускаю программу, также увидим работу метки. (рис.8).

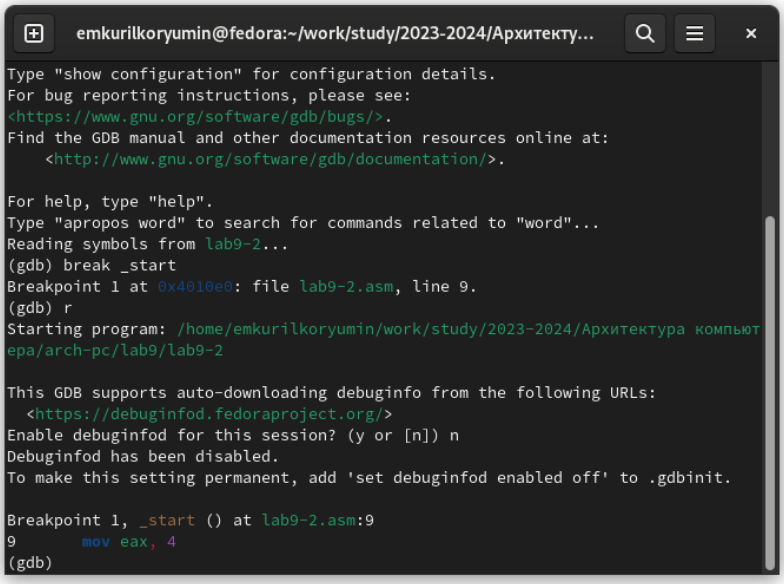


Рис. 4.7: Установка метки

Смотрю дисассимилированный код программы сначала обычный, потом с синтаксисом intel. (рис.9).

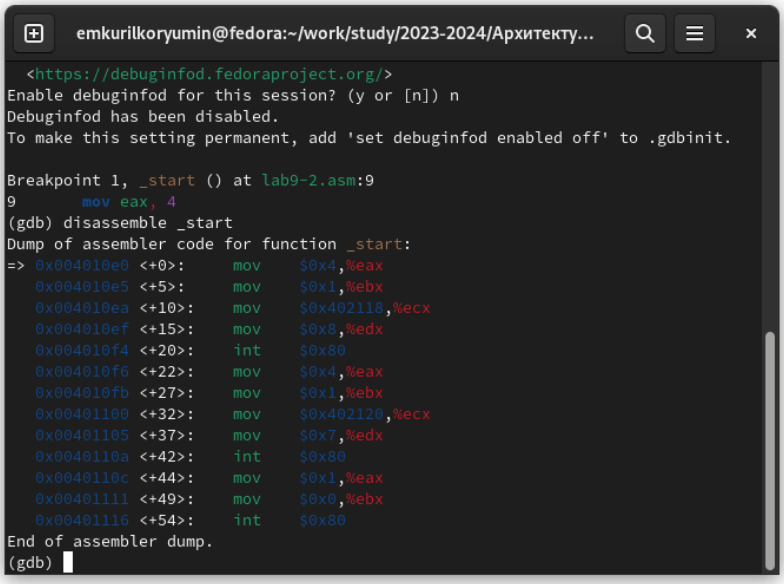


Рис. 4.8: Дисассимилированный код программы с синтаксисом intel

Различия отображения синтаксиса можно наблюдать в правой части окна. Затем я включаю режим псевдографики (рис.10).(рис.11).

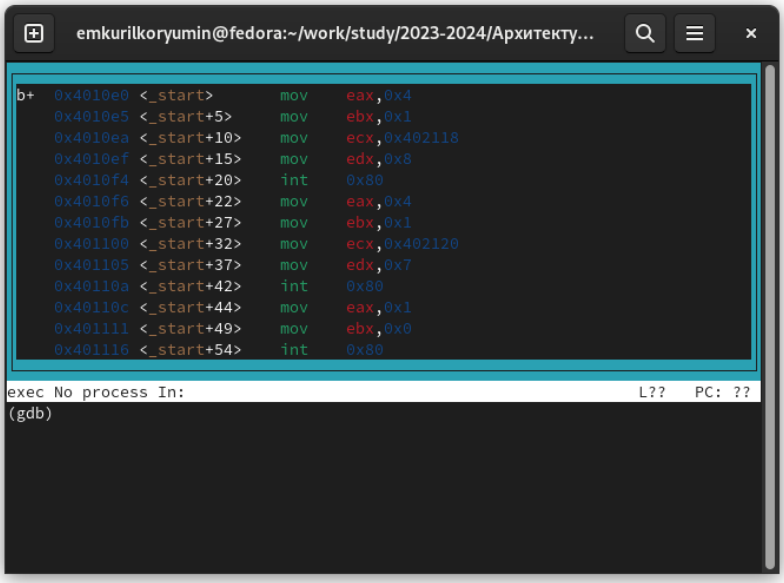


Рис. 4.9: Режим псевдографики

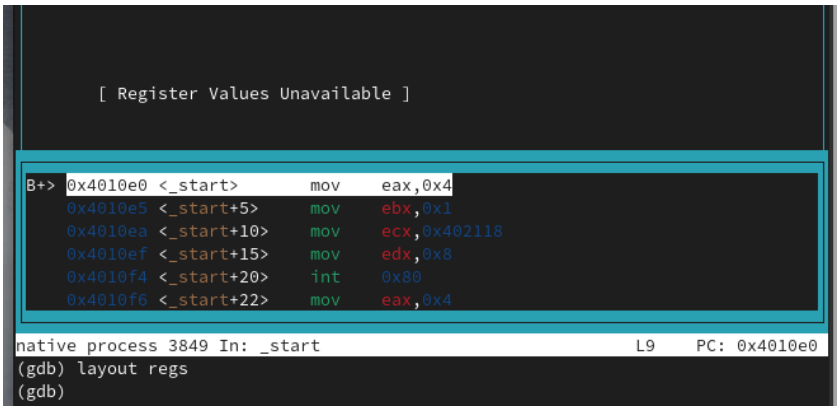


Рис. 4.10: Режим псевдографики

Проверяю точки останова. (рис.12).

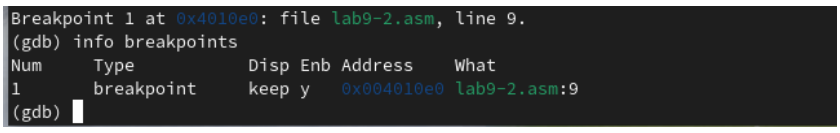


Рис. 4.11: Проверка точек остановки

Устанавливаю точку останова в последней инструкции. (рис.13).

Рис. 4.12: Редактирование файла

Опять же, смотрю информацию обо всех установленных точках останова. (рис.14).

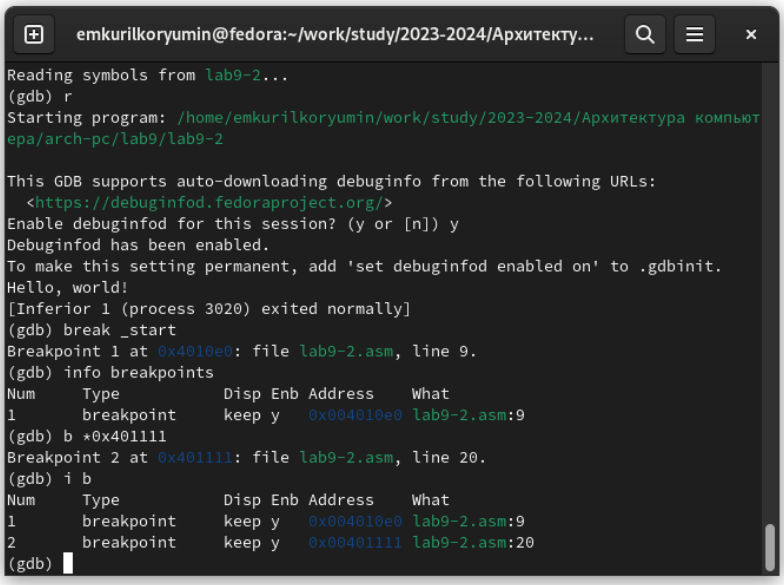


Рис. 4.13: Создание и запуск исполняемого файла

Вручную изменяю значений регистров и переменных с помощью инструкции si. (рис.15).

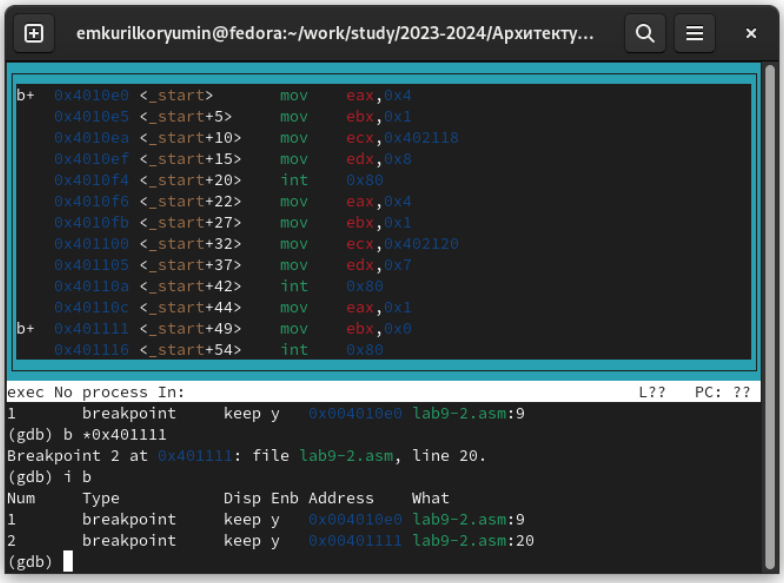


Рис. 4.14: Изменение значений регистров и переменных

Выполняю 5 инструкций si, и последовательно замечанию изменение значений регистров на экране соответственно. (рис.16).

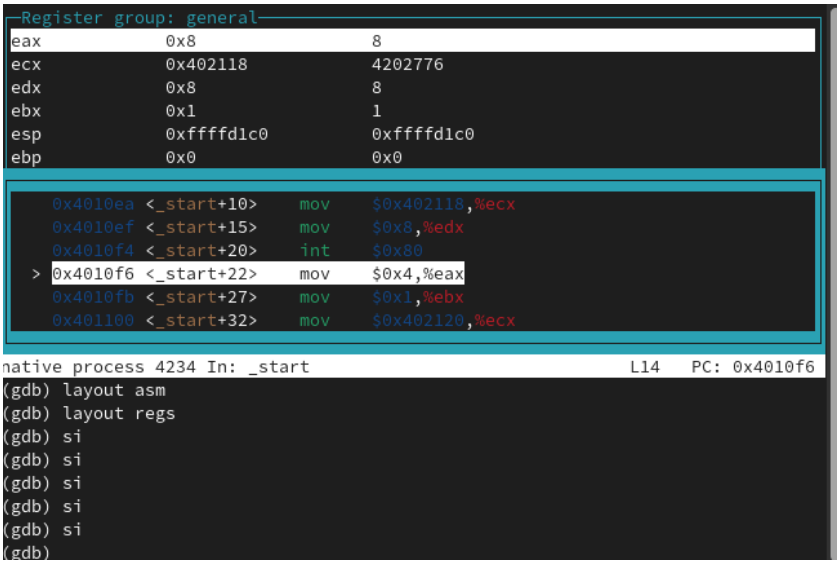


Рис. 4.15: Изменение значений регистров и переменных

Просматриваю содержимое регистров. (рис.17).

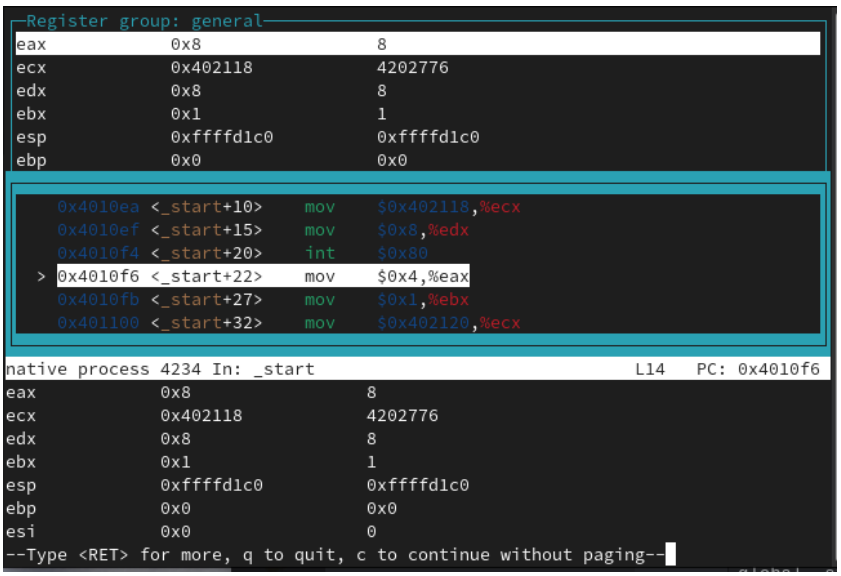


Рис. 4.16: Содержимое регистров

Затем я просматриваю содержимое переменной msg1 и изменяю в ней символ с помощью команды {char}. (рис.18).

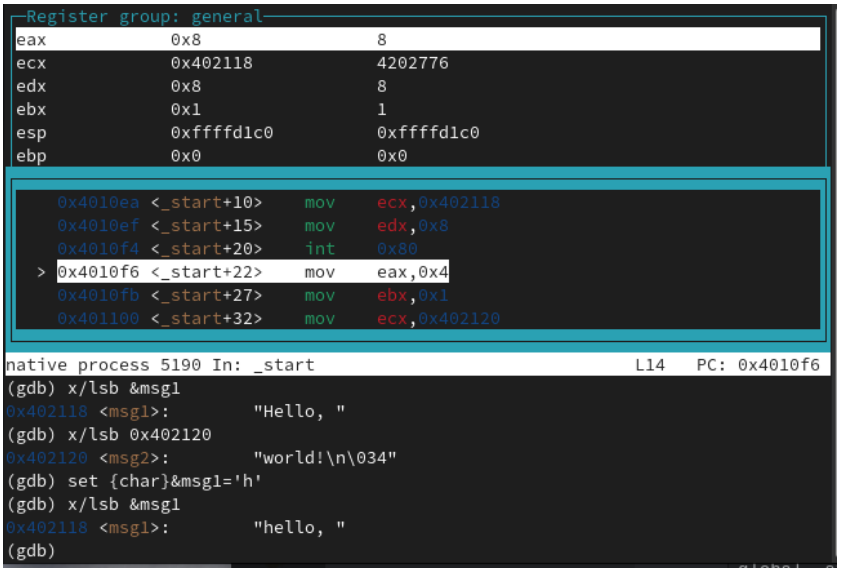


Рис. 4.17: Содержимое переменной, изменение в ней символов

Аналогичные действия проделываю с переменной msg2. (рис.19).

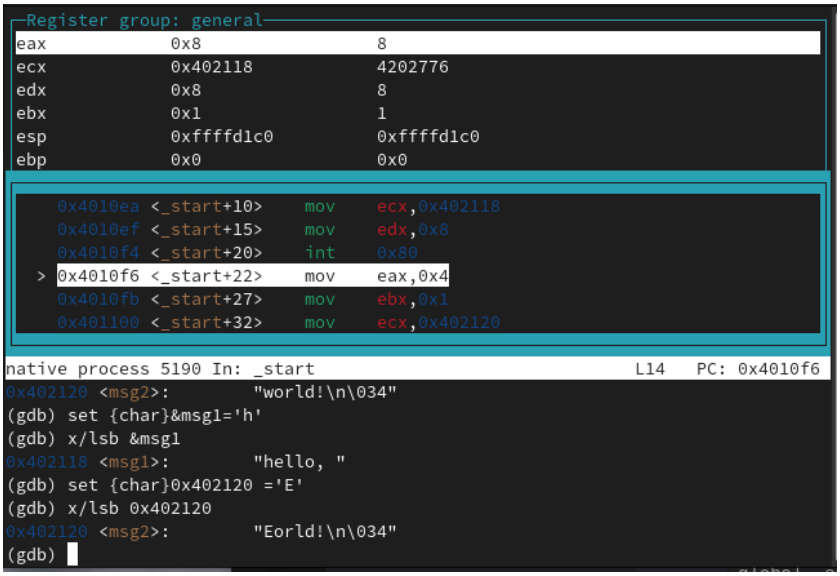


Рис. 4.18: Содержимое переменной, изменение в ней символов

Ввожу в различных форматах значение регистра edx. (рис.20).

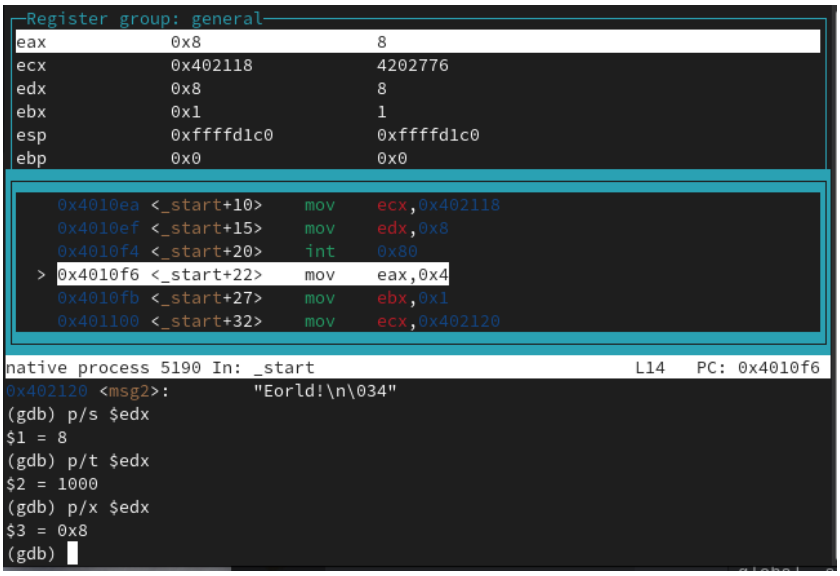


Рис. 4.19: Значения регистра edx

Изменяю значение регистра ebx с помощью команды set. (рис.21).

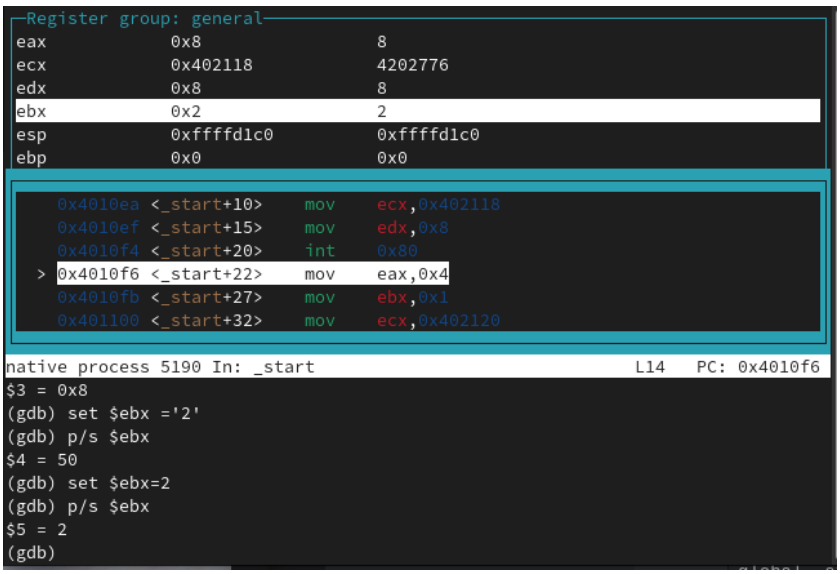


Рис. 4.20: Изменение значений регистра

Разница в выводе команд объяняется в значении: при бескавычном значении 2, мы её и получаем в итоге, а в другом случае переменная воспринимается иначе, и на выходе мы видим значение 50.

Завершаю выполнение программы с помощью continue и выхожу из gdb с помощью quit.

Копирую файл lab8-2.asm, полученный во время выполнения лабораторной работы №8, содержащий программу для вывода аргументов командной строки.Загружаю исполняемый файл в отладчик, указав нужные аргументы. (рис.22).

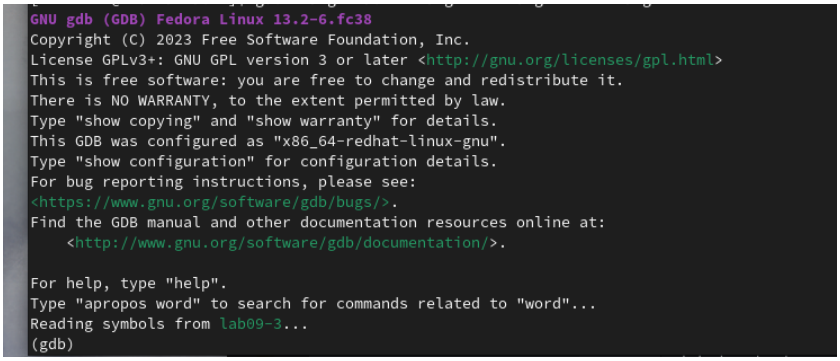


Рис. 4.21: Загрузка файла в отладчик

Устанавливаю точку останова перед первой инструкцией и запускаю программу. (рис.23).

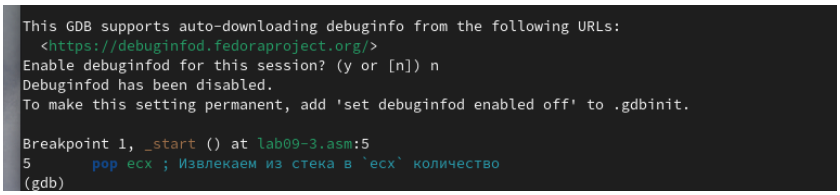


Рис. 4.22: Установка точки останова, запуск программы

Далее просматриваю позиции стека. (рис.24).

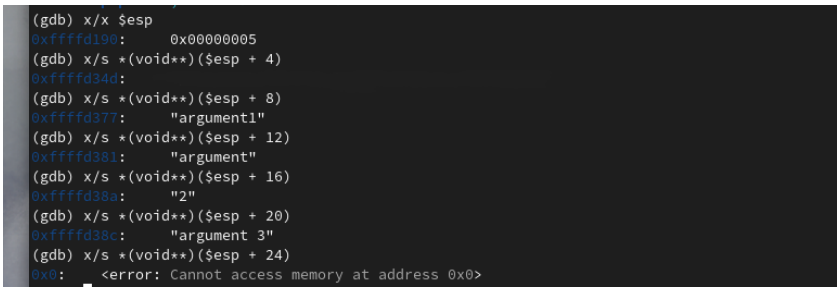


Рис. 4.23: Просмотр позиций стека

Шаг изменения равен 4, т.к. каждый следующий адрес на стеке находится на расстоянии в 4 байта от предыдущего.

4.3) Выполнение заданий для самостоятельной работы

Копирую файл задания для самостоятельной работы, и реализую вычисление значения функции через подпрограмму. (рис.25).

Рис. 4.24: Копирование, редактирование файла

Создаю исполняемый файл и убеждаюсь в правильности работы программы. (рис.26).

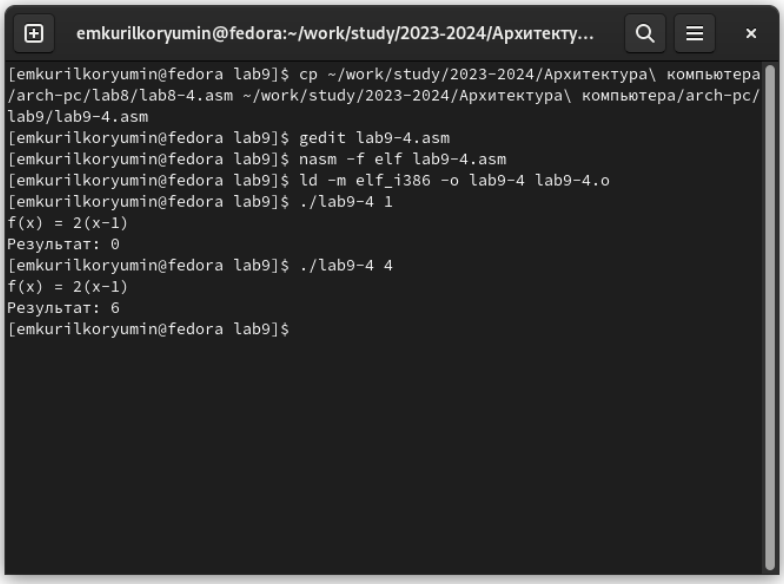


Рис. 4.25: Создание и запуск исполяемого файла

Создаю файл lab9-5.asm и вношу в него программу из последнего листинга. При запуске программа дает неверный результат, и чтобы исправить эту ситуацию, нужно проанализировать изменения значений регистров. (рис.27).

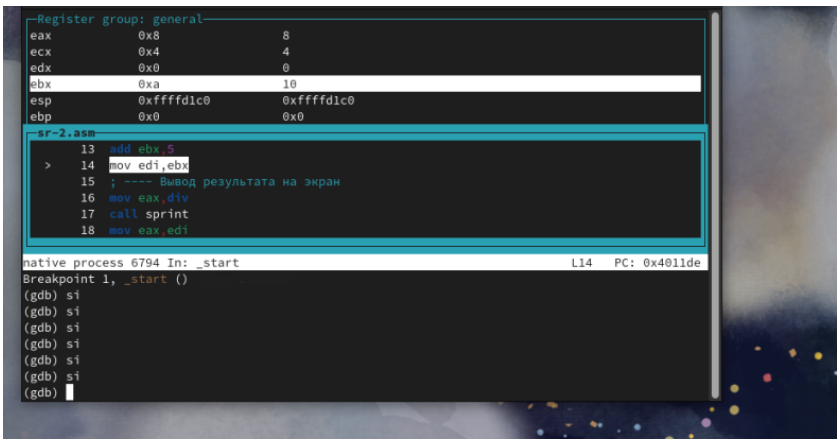


Рис. 4.26: Изменение значений регистров в gdb

Благодаря этому мне удалось вычислить ошибку и исправить её в тексте программы. (рис.28).

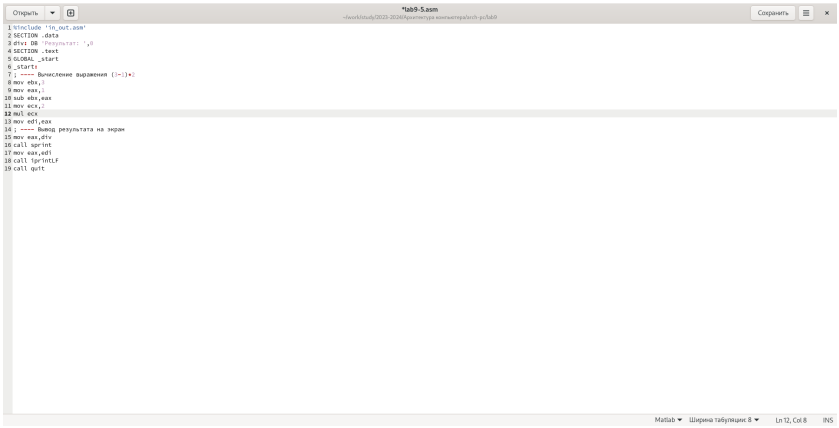


Рис. 4.27: Редактирование файла

Создаю исполняемый файл, и, выполнив устную проверку, убеждаемся в правильности работы программы. (рис.29).

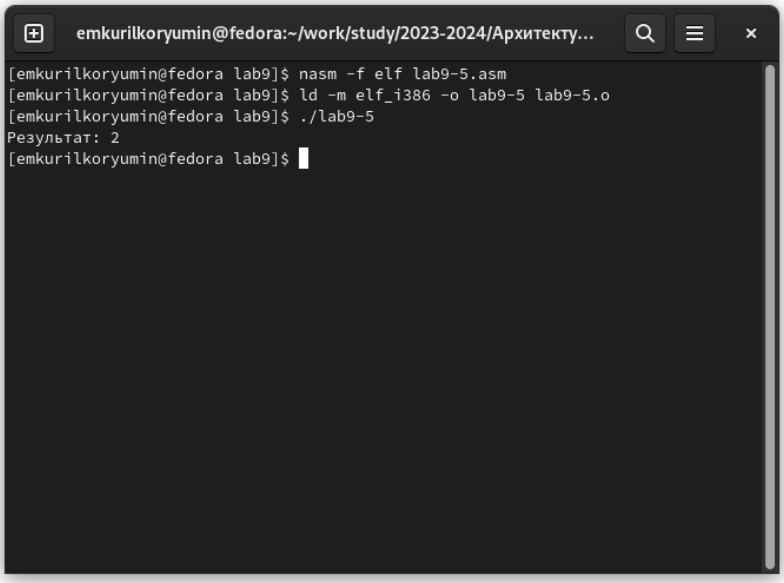


Рис. 4.28: Создание и запуск исполяемого файла

Листинг 4.1 - Преобразованная программа из лабораторной работы №8.

```
%include 'in_out.asm' SECTION .data msg db "Результат: ",0 a:
db 'f(x) = 2(x-1)',0 SECTION .text global _start _start: mov eax,
a call sprintLF pop ecx ; Извлекаем из стека в `ecx` количество ;
аргументов (первое значение в стеке) pop edx ; Извлекаем из стека в
`edx` имя программы ; (второе значение в стеке) sub ecx,1 ; Уменьшаем
`ecx` на 1 (количество ; аргументов без названия программы) mov esi,
0 ; Используем `esi` для хранения ; промежуточных сумм next: cmp
ecx,0h ; проверяем, есть ли еще аргументы jz _end ; если аргументов
нет выходим из цикла ; (переход на метку `_end`) pop eax ; иначе
извлекаем следующий аргумент из стека call atoi ; преобразуем символ
в число call _calc add esi,eax ; добавляем к промежуточной сумме loop
next ; переход к обработке следующего аргумента _end: mov eax, msg ;
вывод сообщения "Результат: " call sprint mov eax, esi ; записываем
сумму в регистр `eax` call iprintLF ; печать результата call quit
; завершение программы _calc: sub eax,1 imul eax,2 ret
```

Листинг 4.2 -

Исправленная программа для вычисления значения выражения.

“‘%include ‘in_out.asm’ %include ‘in_out.asm’ SECTION .data div: DB ‘Результат:’,0 SECTION .text GLOBAL _start _start: ; —- Вычисление выражения (3-1)*2 mov ebx,3 mov eax,1 sub ebx,eax mov ecx,2 mul ecx,2 mov edi,eax ; —- Вывод результата на экран mov eax,div call sprint mov eax,edi call iprintLF call quit

5 Выводы

При выполнении лабораторной работы я приобрел практический опыт в написании программ в написании программ с использованием подпрограмм, а также ознакомился с методами отладки при помощи gdb и его основными возможностями.

Список литературы

Архитектура компьютера и ЭВМ