Шаблон отчёта по лабораторной работе №10

Мугари Абдеррахим , НКАбд-03-22

Содержание

# 1 Цель работы :

В этой лабораторной работе мы научимся писать программы с использованием подпрограмм и познакомимся со способами отладки с использованием GDB и его основными функциями

# 2 Выполнение лабораторной работы :

## 2.1 Реализация циклов в NASM :

1. Здесь мы начали с создания каталога для программаы лабораторной работы №10, а затем переместились в десятой каталог лаборатории “*~/work/arch-pc/lab10*”, после чего мы создали файл “**lab10-1.asm**”. (рис. 1)

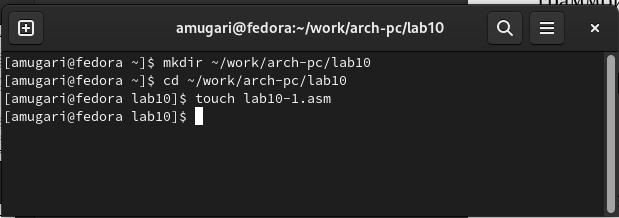


Рис. 1: Ресунок 1

1. Затем мы заполнили код нашей программы в файле **lab10-1.asm**.(рис. 2)

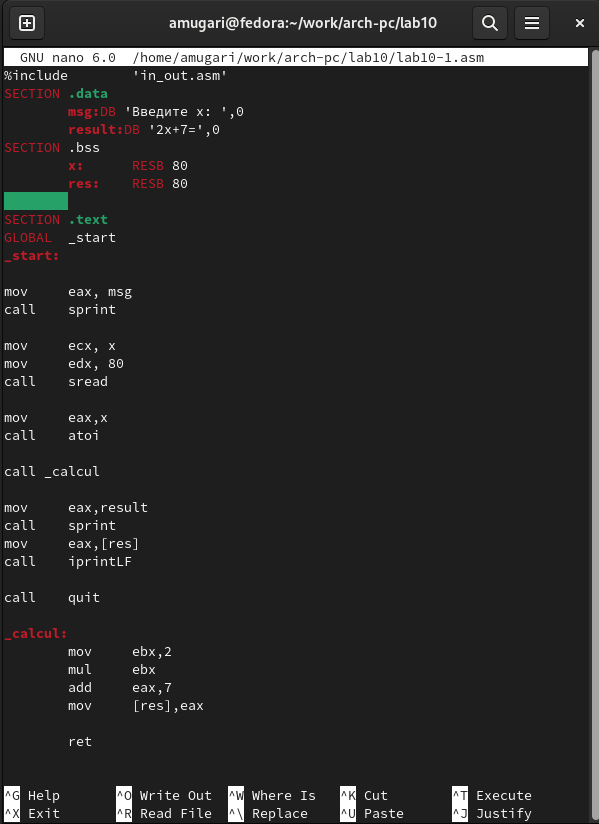


Рис. 2: Ресунок 2

* После этого мы скомпилировали файл, создали исполняемый файл и проверили его работу.(рис. 3)

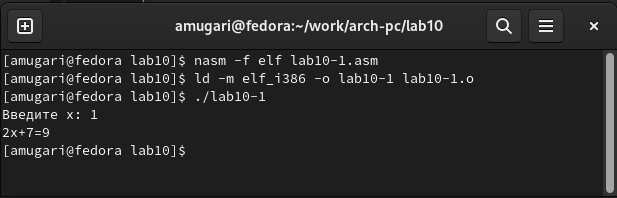


Рис. 3: Ресунок 3

* Мы внесли изменения в наш код ,чтобы она вычислила это уравнение , где вводится с клавиатуры и а затем создали исполняемый файл.(рис. 4) (рис. 5)

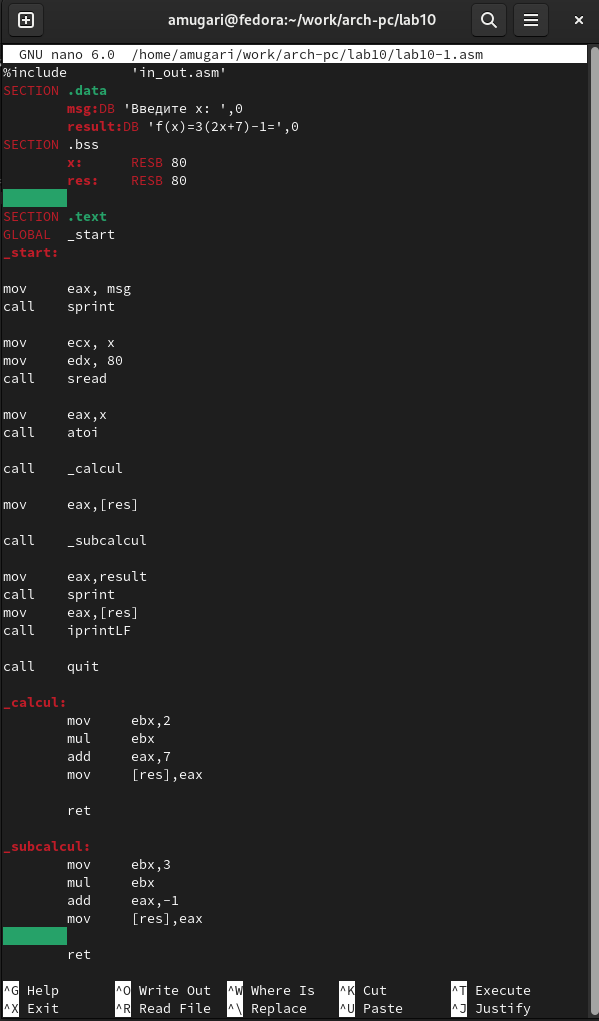


Рис. 4: Ресунок 4

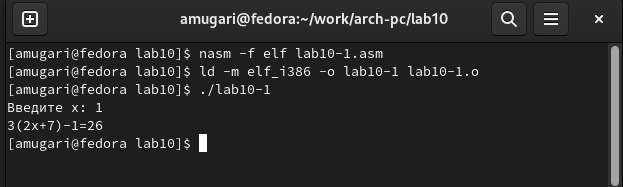


Рис. 5: Ресунок 5

## 2.2 Отладка программам с помощью GDB :

1. На этом шаге мы создали файл **lab10-2.asm** с текстом программы из **листинга 10.2**. ((*Программа печати сообщений Hello world!*). (рис. 6)

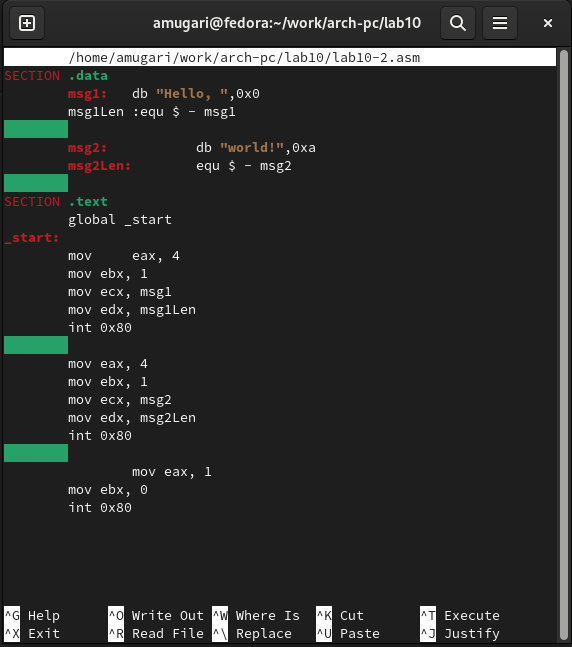


Рис. 6: Ресунок 6

1. После этого мы скомпилировали файл, создали исполняемый файл.Затем мы загрузили исполняемый файл в **отладчик GDM**. (рис. 7)

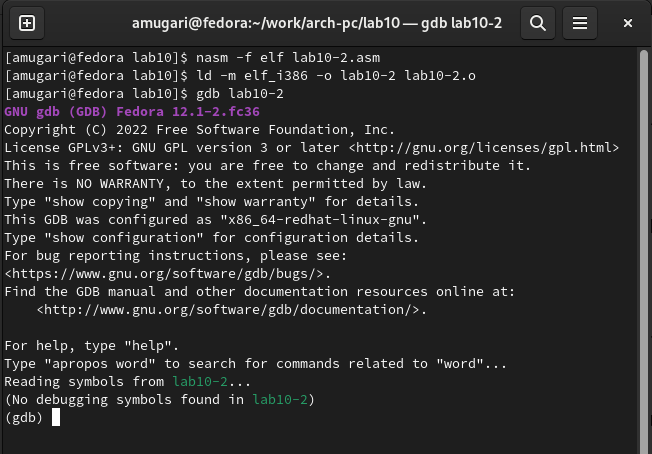


Рис. 7: Ресунок 7

1. затем мы проверили работу программы, запустив ее в оболочке GDB с помощью команды **run**. (рис. 8)

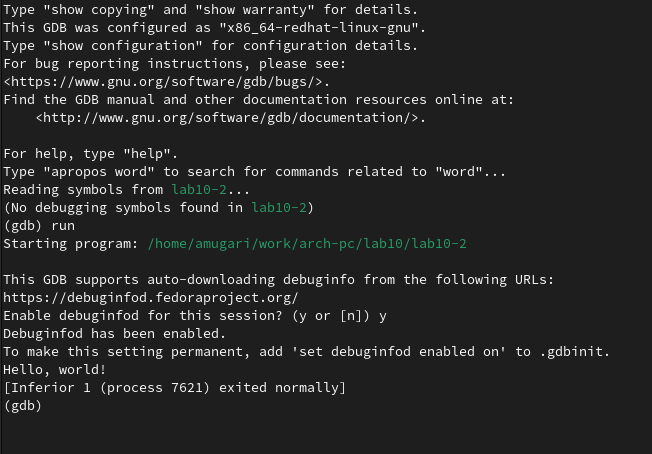


Рис. 8: Ресунок 8

1. затем мы установили точку останова на метке \*\*\_start\*\*, которая запускает выполнение любой программы на ассемблере, и запустили ее. (рис. 9)

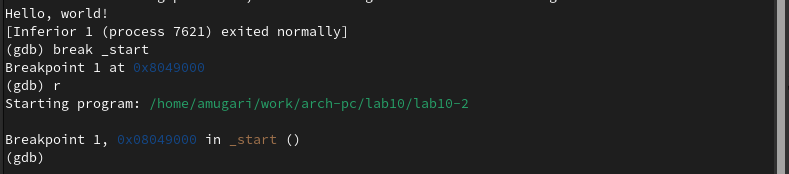


Рис. 9: Ресунок 9

1. Затем мы просмотрели разобранный программный код, используя команду **disassemble**, начинающуюся с метки \*\*\_start\*\*. (рис. 10)

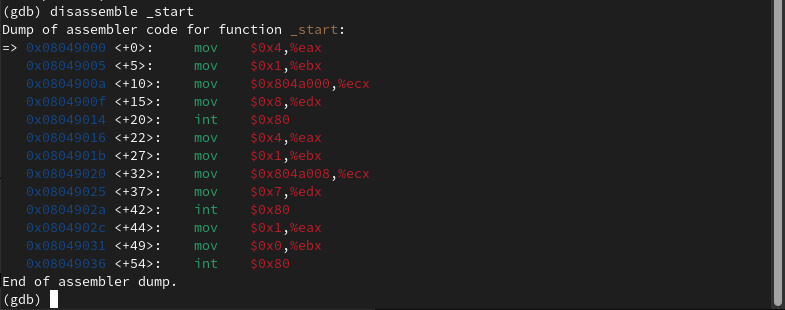


Рис. 10: Ресунок 10

1. после этого мы переключились на отображение команд с **синтаксисом Intel**, введя команду **set disassembly-flavor intel**. (рис. 11)

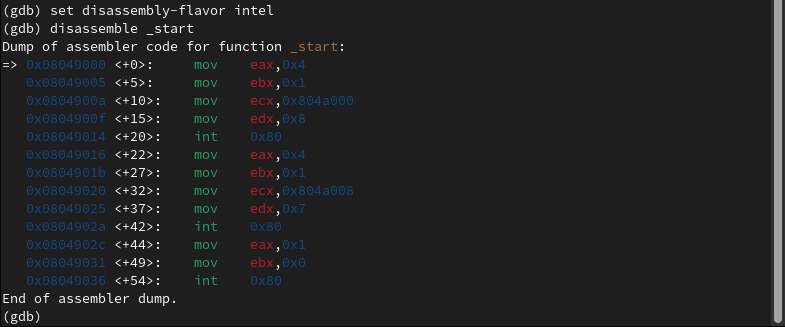


Рис. 11: Ресунок 11

* Разница в синтаксисе между **AT&T** и **INTEL** заключается в том, что **AT&T** использует синтаксис **mov $0x4,%eax**, который популярен среди пользователей **Linux**, с другой стороны, **INTEL** использует синтаксис **mov eax,0x4** , который является популярен среди пользователей Windows.

1. Затем мы включили псевдографический режим для более удобного анализа программы. (рис. 12)

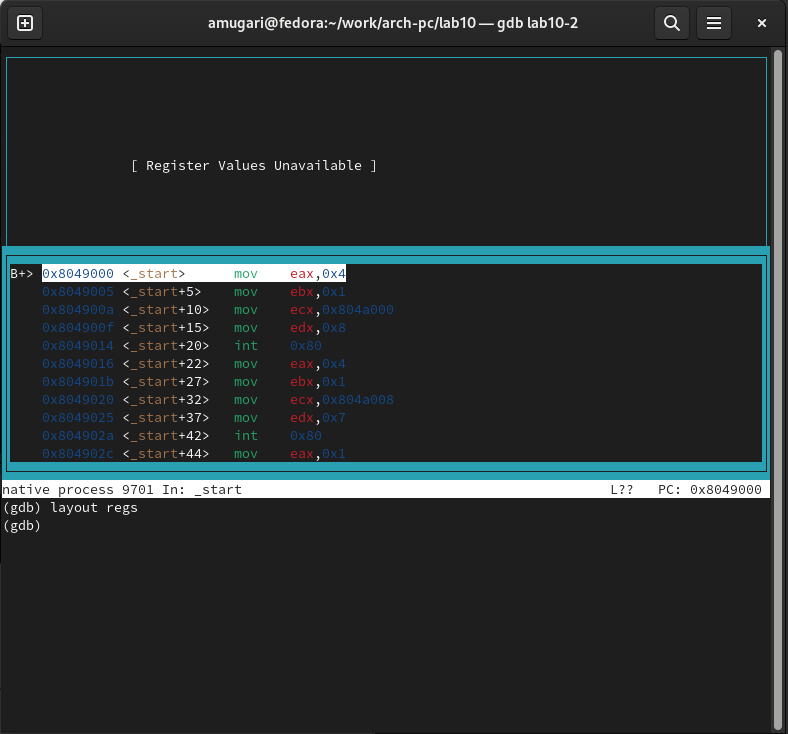


Рис. 12: Ресунок 12

## 2.3 Добавление точек останова :

1. Мы проверили точку останова с помощью информационных точек останова. (рис. 13)

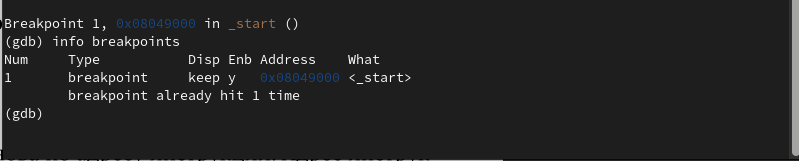


Рис. 13: Ресунок 13

1. Mы определили адрес предпоследней инструкции (**mov ebx,0x0**) и установили точку останова.(рис. 14)

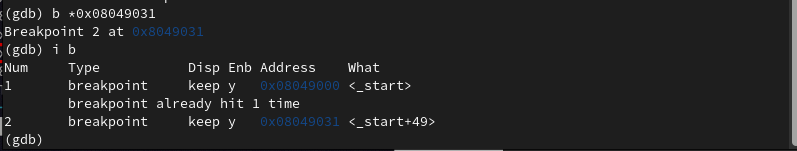


Рис. 14: Ресунок 14

## 2.4 Работа с данными программы в GDB :

1. На этом шаге мы следовали 5 инструкциям, используя командный шаг i, и отслеживали изменение значений регистров, но перед этим мы проверили предыдущие значения регистров.(рис. 15) (рис. 16)

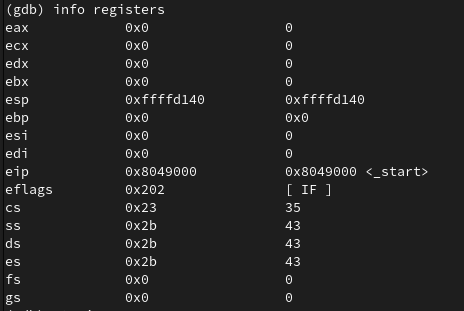


Рис. 15: Ресунок 15

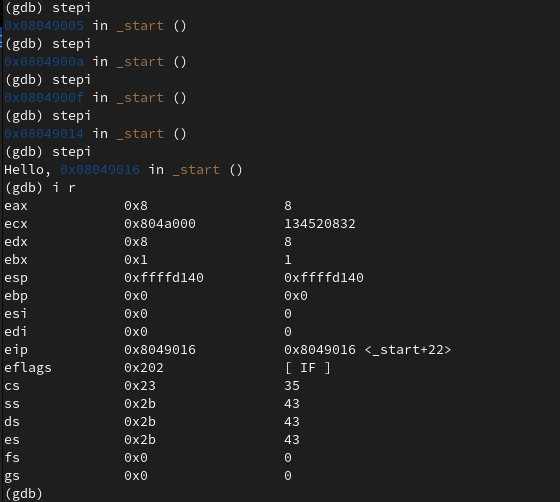


Рис. 16: Ресунок 16

* После проверки мы видим, что регистры : **eax,ecx,edx,ebx,esp** изменили свое значение.

1. Mы рассмотрели значение переменной msg1 по имени, используя команду **x/1sb**(рис. 17)

Рис. 17: Ресунок 17

Рис. 17: Ресунок 17

1. Здесь мы рассмотрели значение переменной msg2, используя адрес.(рис. 18)

Рис. 18: Ресунок 18

Рис. 18: Ресунок 18

1. Здесь мы изменили первую букву переменной **msg1**, которая имеет тип **char**.(рис. 19)

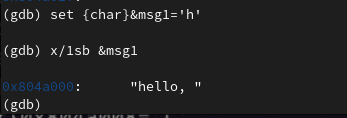


Рис. 19: Ресунок 19

1. После этого мы изменили первую букву переменной **msg2**.(рис. 20)

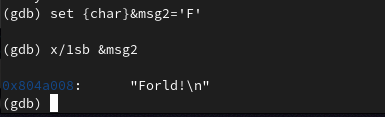


Рис. 20: Ресунок 20

1. Затем мы выводим значение регистра **edx** в различных форматах (шестнадцатеричном, двоичном и символьном).(рис. 21)

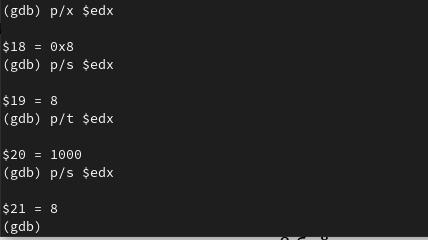


Рис. 21: Ресунок 21

1. Используя команду **set**, мы изменили значение регистра **ebx**, когда раз, введя ‘2’, а в другой раз, введя **2**.(рис. 22)

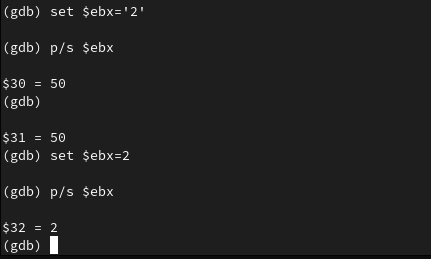


Рис. 22: Ресунок 22

* но когда мы напечатали значение регистра, мы получили значение **50** и это потому, что машина интерпретировала введенное значение как символ, и в таблице **ASCII** символ ‘2’ имеет значение **50** в десятичной системе, но когда мы ввели значение **2** машина интерпретировала **2** как число в десятичной системе.

1. Наконец, мы завершили программу с помощью **stepi** и вышли из GDB с помощью команды **quit**.(рис. 23)

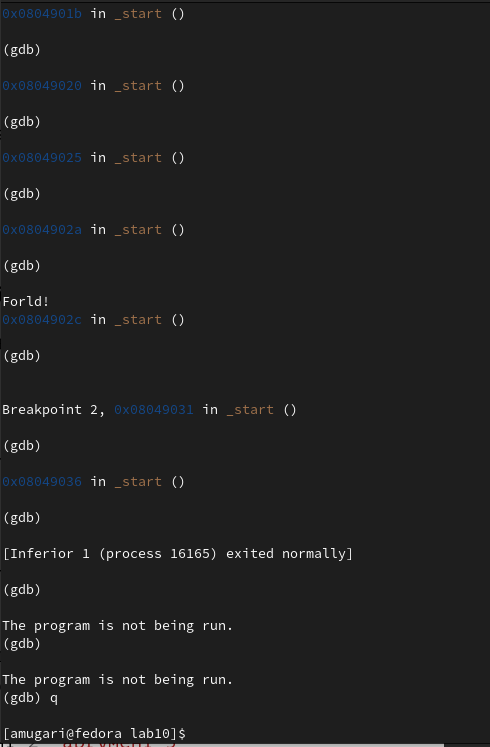


Рис. 23: Ресунок 23

## 2.5 Обработка аргументов командной строки в GDB :

1. На этом этапе мы скопировали файл **lab9-2.asm**, созданный при выполнении лабораторной работы **№9** с программой, отображающей аргументы командной строки на экране (листинг 9.2), в файл с именем **lab 10-3.asm**, а затем мы скомпилировали этот файл и установил точку останова в \*\*\_start\*\* и запустил отладчик.(рис. 24)

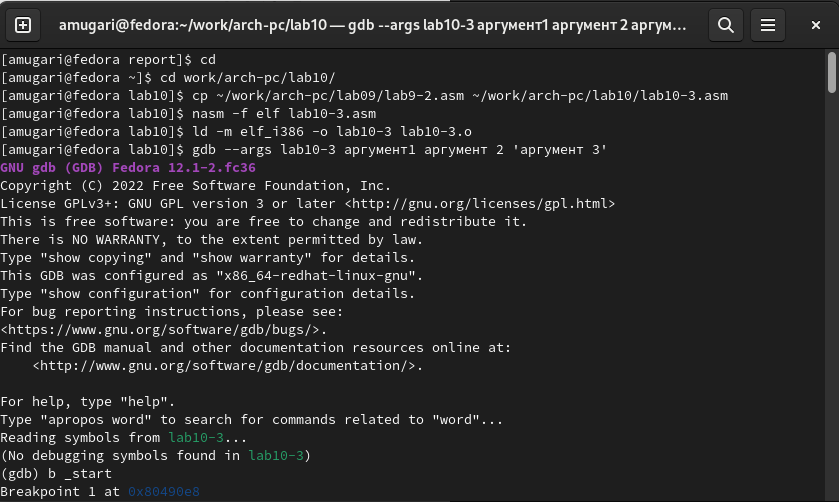


Рис. 24: Ресунок 24

1. Затем мы посмотрели на остальные позиции стека – адрес в памяти, где находится имя программы, находится в *[esp + 4]*, адрес первого аргумента хранится в *[esp +8]*, в *[esp +12]*.(рис. 25)

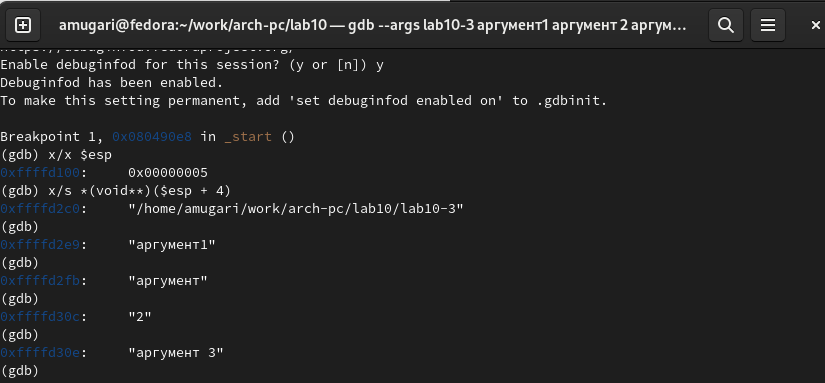


Рис. 25: Ресунок 25

* Шаг изменения адреса равен 4, потому что размер регистра **esp** равен **32битам = 4 байтам**, а количество памяти равно количеству аргументов плюс имя программы, поэтому мы получили 5 шагов с 4 байтами для каждого шага.(рис. 26)

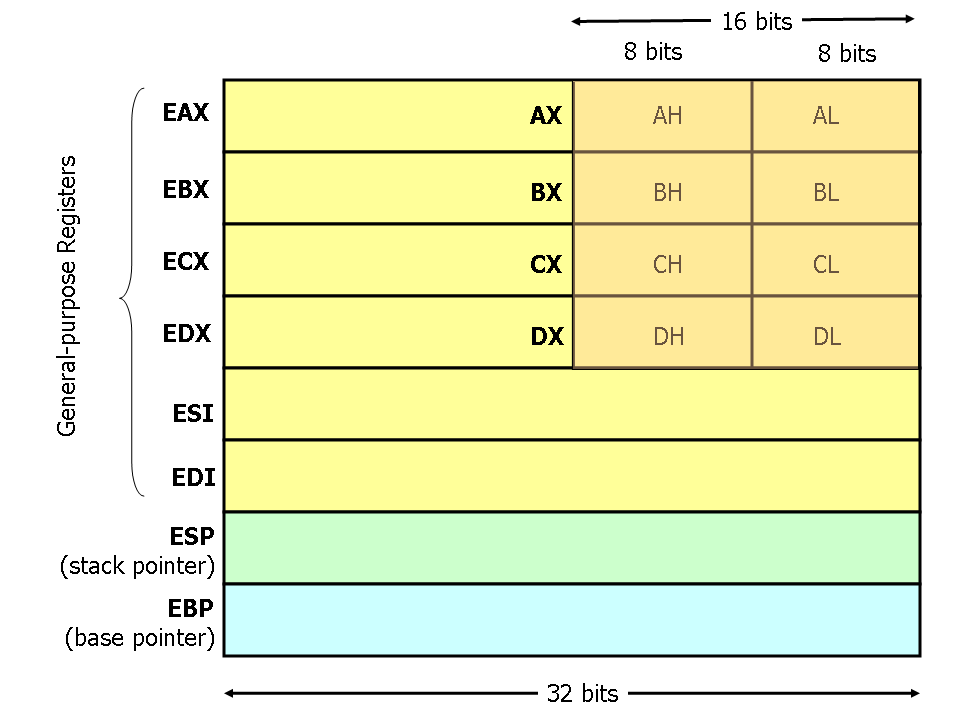


Рис. 26: Ресунок 26

## 2.6 Выводы по результатам выполнения заданий :

* В этой части работы мы узнали, как работать с отладчиком GDB, и получили более близкое представление о том, как работают подпрограммы.

# 3 Задание для самостоятельной работы :

## 3.1 Выводы по результатам выполнения заданий :

* В этой части мы узнали, как превратить программу в подпрограмму, но у нас возникла проблема с подпрограммой **atoi** , поэтому мы не смогли вычислить результат.

# 4 Выводы, согласованные с целью работы :

* В этой лабораторной работе мы научимся писать программы с использованием подпрограмм и познакомимся со способами отладки с использованием GDB и его основными функциями