Лабораторная работа №10

Понятие подпрограммы. Отладчик GDB.

Киньябаева Аиша Иделевна

Содержание

Список иллюстраций

1 Цель работы

Целью работы является приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможно- стями.

2 Задание

Научиться отладке программы с помощью отладчика GDB и изучение подпрограмм.

3 Выполнение лабораторной работы

Пишу программу с использованием с подпрограммы для вычисления выражения: 2x + 7 (рис. 3.1), (рис. 3.2)

```
%include 'in_out.asm'

SECTION .data
 msg: DB 'Введите x: ',0
 result: DB '2x+7=',0
SECTION .bss
 x: RESB 80
 rezs: RESB 80
SECTION .text
GLOBAL _start
 _start:
mov eax, msg
call sprint

mov ecx, x
mov edx, 80
call sread

mov eax,x
call atoi

call _calcul
mov eax,result
call sprint
mov eax,[rezs]
call iprintLF

call quit

_calcul:
mov ebx,2
mul ebx
add eax,7
mov [rezs],eax

ret
```

Рис. 3.1: код

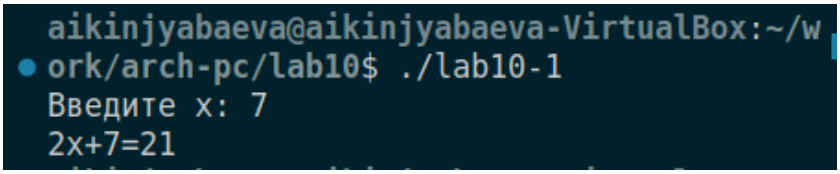

Рис. 3.2: lab10-1.asm_вывод

Меняем программу, чтобы она вычисляла: f(g(x)) = 2(3x - 1) + 7 (рис. [#fig:fig3]), (рис. 3.3), (рис. 3.5)

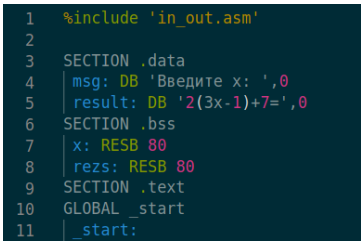

Рис. 3.3: код

```
 7   x: RESB 80
 8   rezs: RESB 80
 9   SECTION .text
10   GLOBAL _start
11   _start:
12   mov eax, msg
13   call sprint
14
15   mov ecx, x
16   mov edx, 80
17   call sread
18
19   mov eax,x
20   call atoi
21
22   call _calcul
23   mov eax,result
24   call sprint
25   mov eax,[rezs]
26   call iprintLF
27
28   call quit
29
30   _calcul:
31   call _subcalcul
32   mov ebx,2
33   mul ebx
34   add eax,7
35   mov [rezs],eax
36
37   ret
38
39   _subcalcul:
40   mov ebx,3
41   mul ebx
42   dec eax
43   mov [rezs],eax
44
45   ret
```

Рис. 3.4: код

```
aikinjyabaeva@aikinjyabaeva-VirtualBox:~/work/arc
h-pc/lab10$ ./lab10-1
Введите x: 3
2(3x-1)+7=23
aikinjyabaeva@aikinjyabaeva-VirtualBox:~/work/arc
```

Рис. 3.5: lab10-1.asm_вывод

Создаем программу для вывода надписи: Hello world, компилируем листинг файл с отладочной информацией, проверяем с помощью дебаггера GDB.(рис. 3.6), (рис. 3.7)

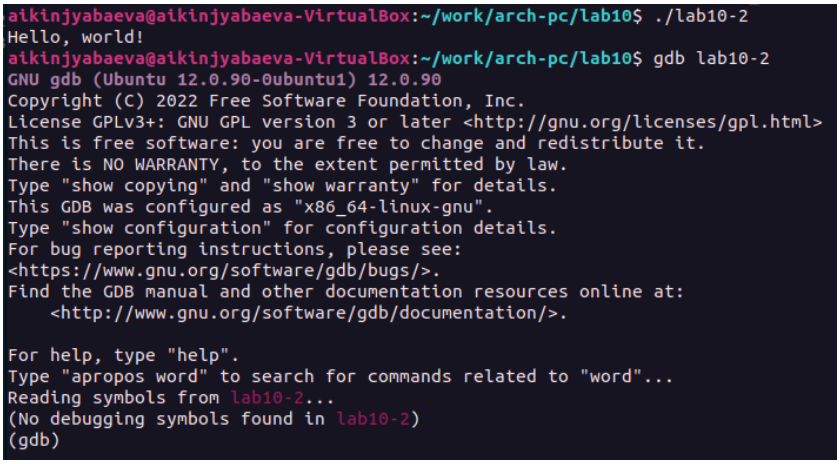

Рис. 3.6: GDB

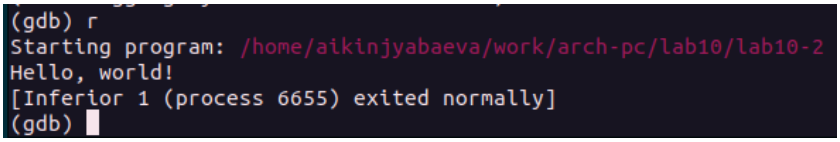

Рис. 3.7: проверка через run

Устанавливаем break на метку _start, смотрим работу программы и далее запускаем диссамилированный код. Переключаемся на Intel'овский синтаксис. Различия синтаксисов заключается в разном отображении аргументов и значений(в Intel без доп. символов), а так же изменен порядок их отображения(рис. 3.8), (рис. 3.9)

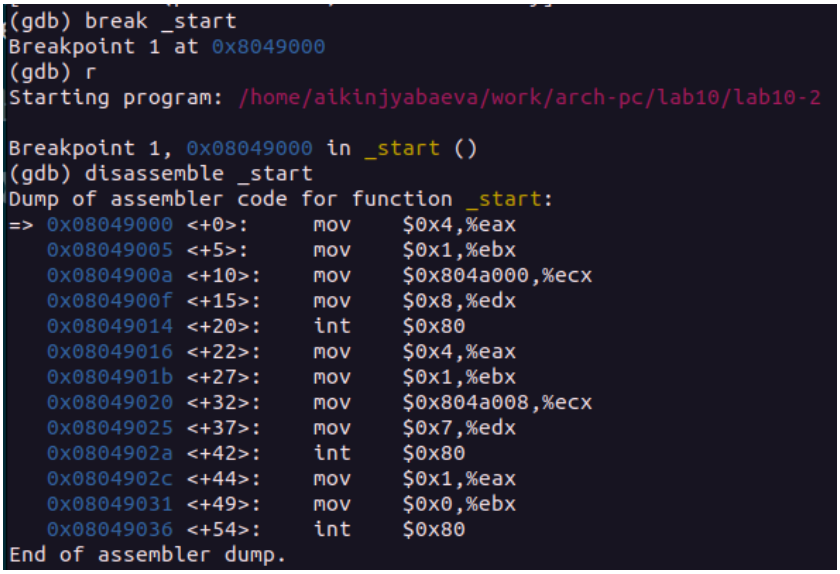

Рис. 3.8: break point

Рис. 3.9: Синтаксис Intel

Два режима псевдографики(рис. 3.10), (рис. 3.11)

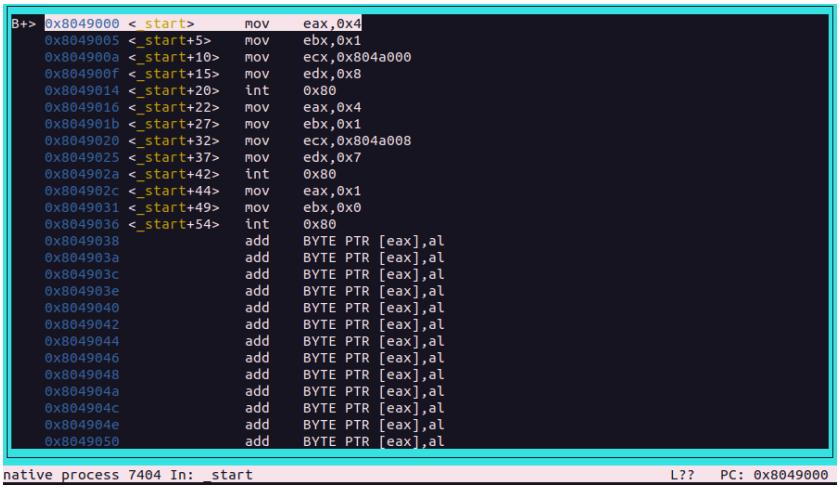

Рис. 3.10: layout asm

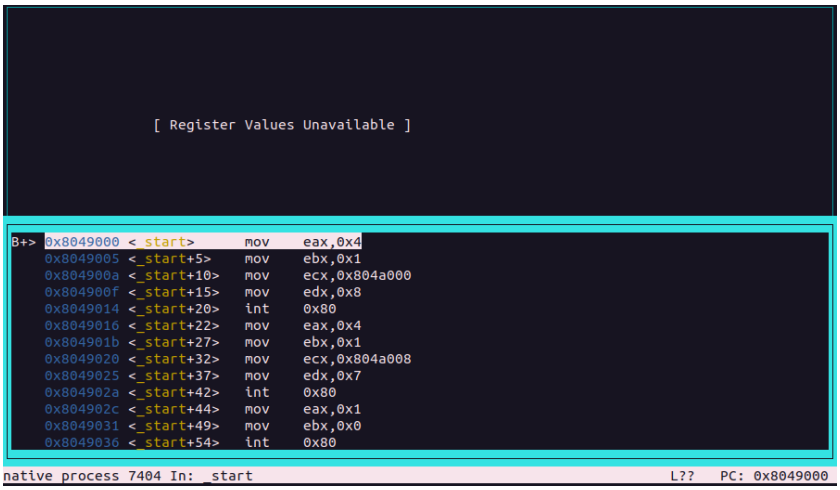

Рис. 3.11: layout rex

В режиме дебаггинга проверяем точки остановы, и видим, что можем установить еще одну, используя адрес инструкции (рис. 3.12)

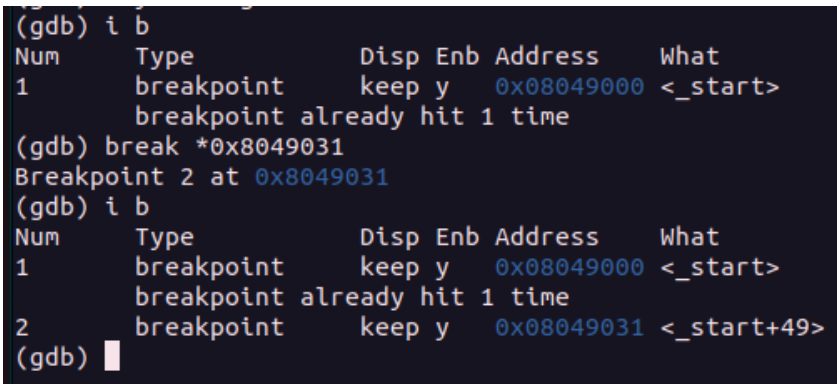

Рис. 3.12: info breakpoints

Проверяем значения регистров(рис. 3.13)

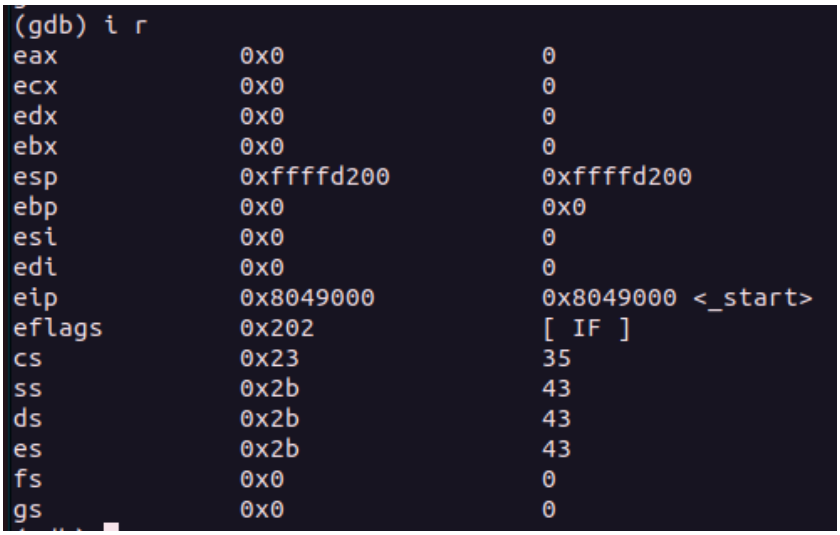

Рис. 3.13: info registers

Провела 5 инструкций с помощью команды si, меняются значения регистров: eax, ebx, ecx, edx. (рис. 3.14)

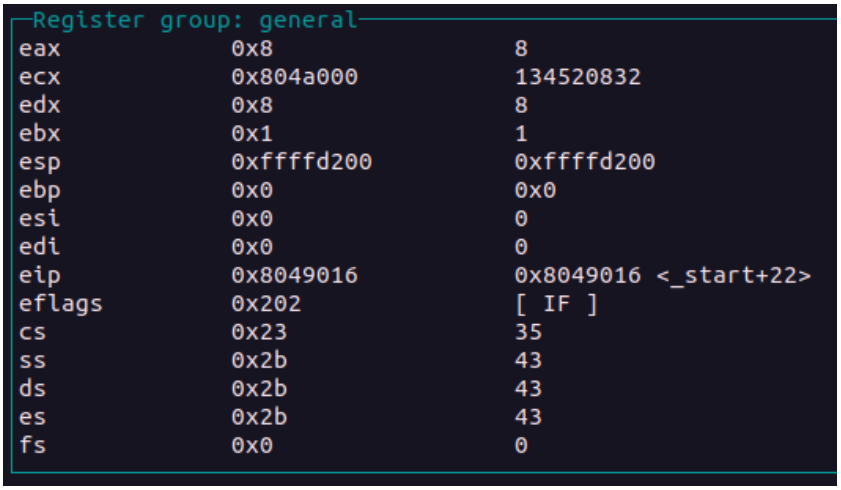

Рис. 3.14: si 5

Видим, что с помощью различных команд можно смотреть содержимое переменных(в конце скриншота просмотрено значение переменной msg2) (рис. 3.15)

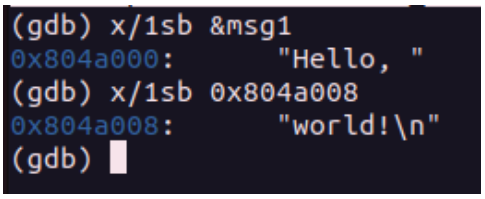

Рис. 3.15: x/1sb

А с помощью команды set можно менять значение регистра(в конце скриншота изменен символ второй переменной msg2)(рис. 3.16)

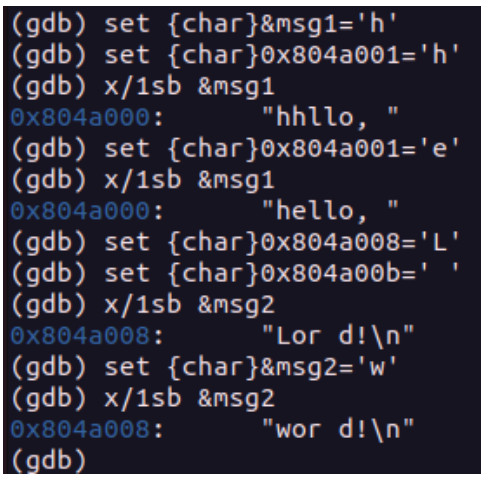

Рис. 3.16: set

Изучаем использование команды print /F, которое смотрит значение регистров(можно регулировать формат отображения). Вывела в различных форматах значение регистра edx(16ричных, 2чный и символьный вид соотвественно)(рис. 3.17)

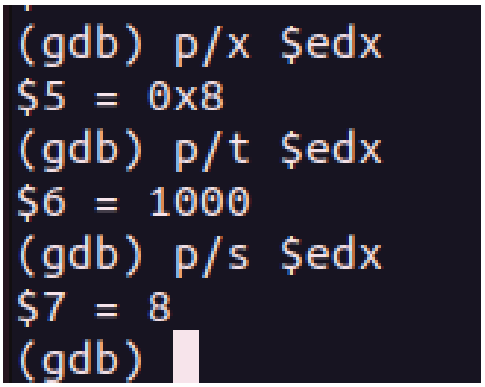

Рис. 3.17: значение edx

С помощью команды set, меняем значение регистра ebx (рис. 3.18)

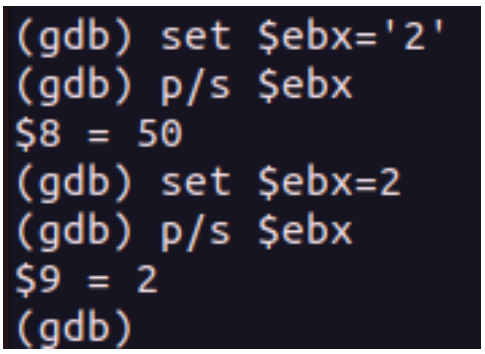

Рис. 3.18: значение ebx

Указываем аргументы файла в отладчик и далее смотрим позиции аргумента в стеке. Заметим, что [esp + 24] не выводит значения, так как у нас нет 6го аргумента. Шаг изменения адреса равен 4, потому что память имеет 16ричный вид, и 2^4=16.(рис. 3.19)

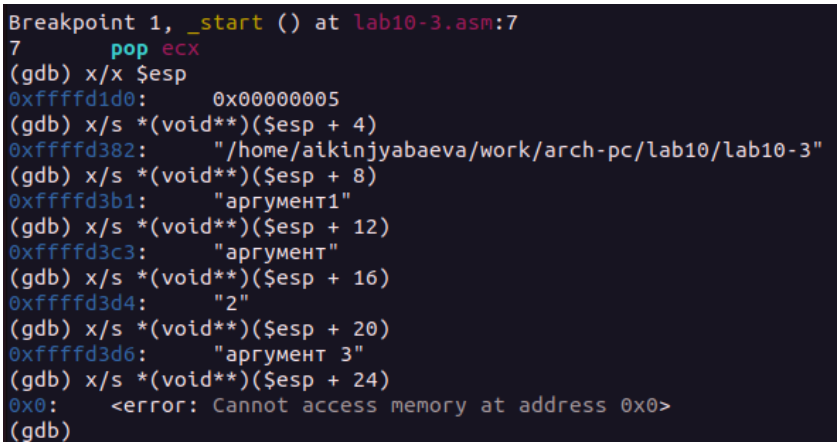

Рис. 3.19: аргументы

САМОСТОЯТЕЛЬНАЯ РАБОТА

1.Меняю программу из 9 лаб. работы, которая вычисляла 2(x-1) с использованием подпрограмм (рис. 3.20), (рис. 3.21)

```
mywork.asm
1   %include 'in_out.asm'
2
3   SECTION .data
4   msg: DB 'Введите x: ',0
5   result: DB '2(x-1)=',0
6   SECTION .bss
7   x: RESB 80
8   rezs: RESB 80
9   SECTION .text
10  GLOBAL _start
11  _start:
12  mov eax, msg
13  call sprint
14
15  mov ecx, x
16  mov edx, 80
17  call sread
18
19  mov eax,x
20  call atoi
21
22  call _calcul
23  mov eax,result
24  call sprint
25  mov eax,[rezs]
26  call iprintLF
27
28  call quit
29
30  _calcul:
31  dec eax
32  mov ebx,2
33  mul ebx
34  mov [rezs],eax
35  ret
```

Рис. 3.20: код

```
aikinjyabaeva@aikinjyabaeva-VirtualBox:~/work/arch-pc/lab10$ ./mywork
Введите x: 7
2(x-1)=12
aikinjyabaeva@aikinjyabaeva-VirtualBox:~/work/arch-pc/lab10$ S
```

Рис. 3.21: 2(x-1) вывод

2.Находим ошибка в листинге, для этого используем GDB, смотрим значение регистров. Просматриваю каждый шаг программы с помощью si (рис. 3.22), (рис. 3.23), (рис. 3.24), (рис. 3.25), (рис. 3.26), (рис. 3.27), (рис. 3.28)

```
mywork-1.asm
        7  ; ---- Вычисление выражения (3+2)*4+5
B+>     8  mov ebx,3
        9  mov eax,2
       10  add ebx,eax
       11  mov ecx,4
       12  mul ecx
       13  add ebx,5
native process 4272 In: _start                    L8    PC: 0x80490e8
```

Рис. 3.22: проверка программы

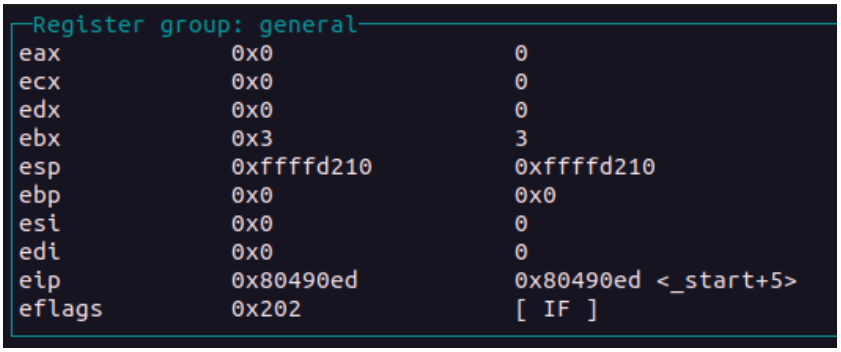

Рис. 3.23: проверка программы

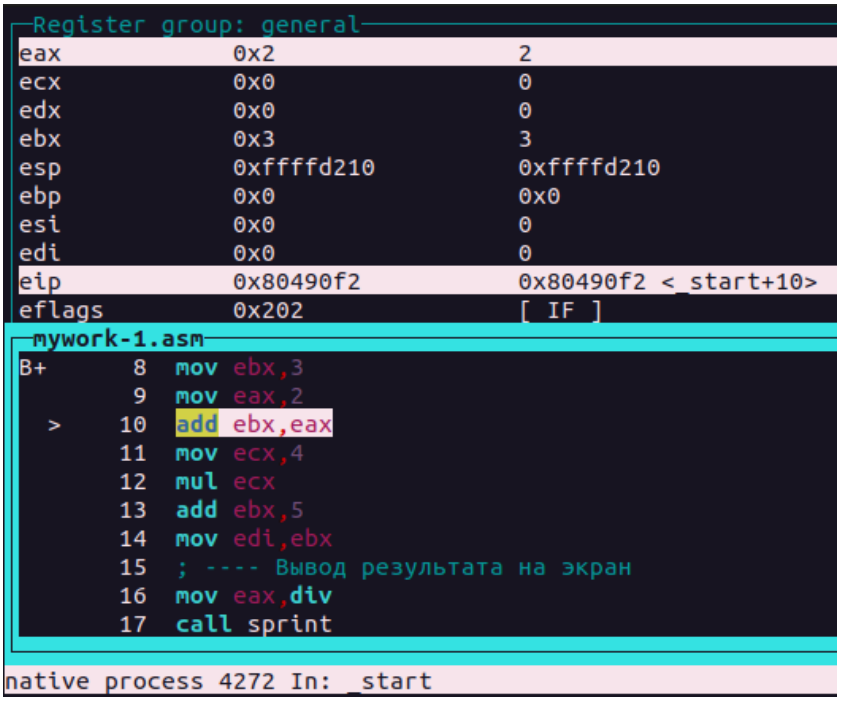

Рис. 3.24: проверка программы

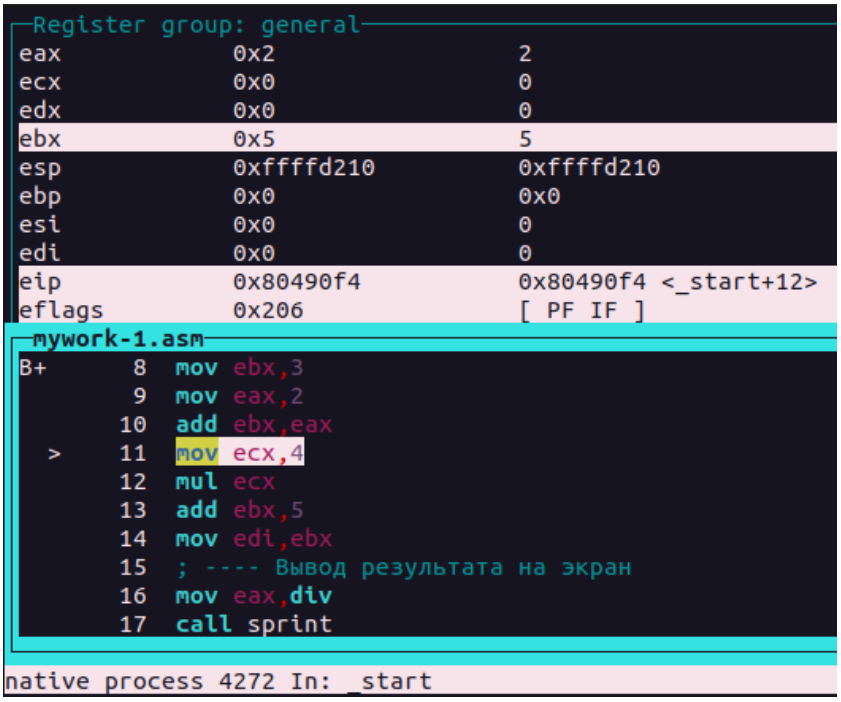

Рис. 3.25: проверка программы

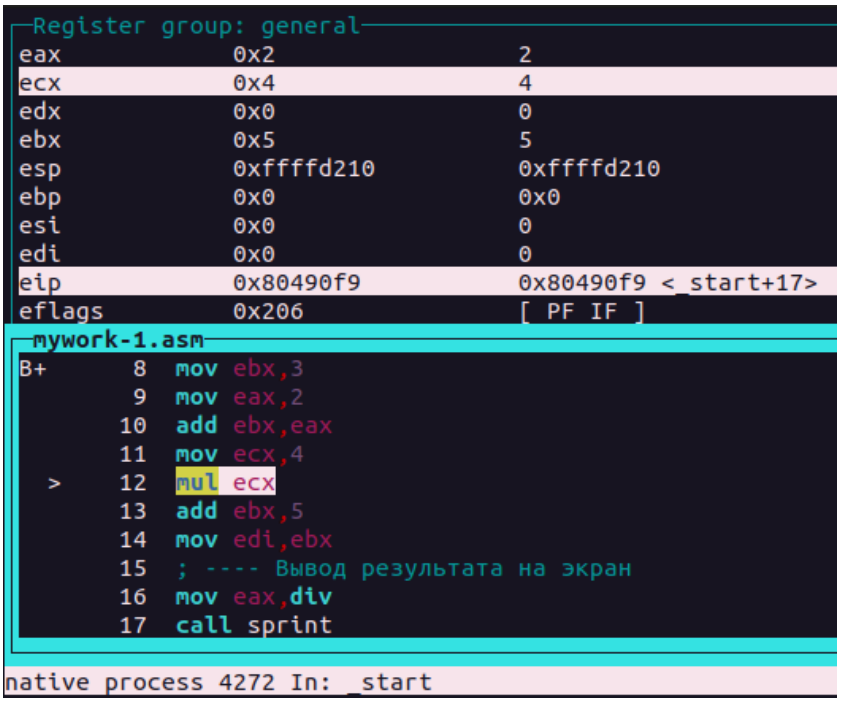

Рис. 3.26: проверка программы

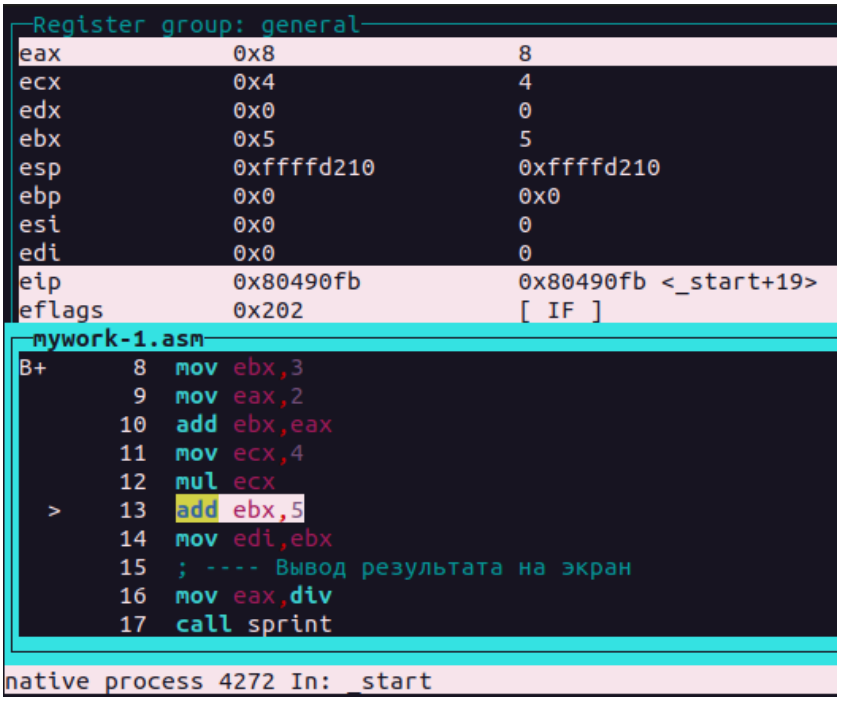

Рис. 3.27: проверка программы

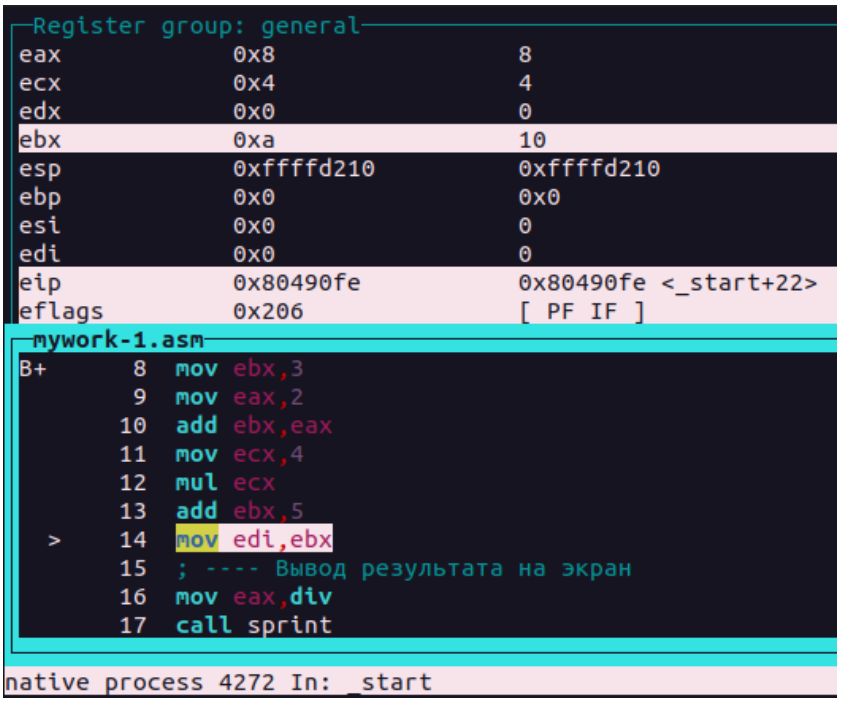

Рис. 3.28: проверка программы

Видим, что ошибка в том, что программа умножает на 4 двойку из eax. Исправ-

ляем программу в редакторе. (рис. 3.29), (рис. 3.30)

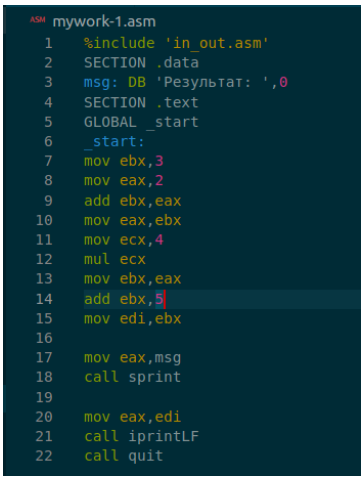

Рис. 3.29: код

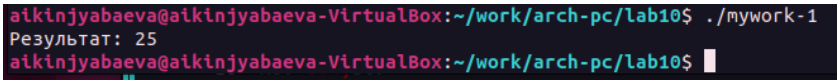

Рис. 3.30: (3 + 2) * 4 + 5 вывод

Загрузка всех файлов на Git.

Далее создается отчет по 10й лабораторной работе с помощью Markdown.

4 Выводы

В ходе данной лабораторной работы были изучены подпрограммы, освнен дебагинг с помощью GDB.