Архитектура компьютера

Отчёт по лабораторной работе №9

Арбатова Варвара Петровна

Содержание

Список таблиц

Список иллюстраций

1 Цель работы

Приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

2 Задание

1. Создайте каталог для выполнения лабораторной работы № 9, перейдите в него и создайте файл lab09-1.asm
2. Внимательно изучите текст программы (Листинг 9.1). Введите в файл lab09-1.asm текст программы из листинга 9.1. Создайте исполняемый файл и проверьте его работу
3. Создайте файл lab09-2.asm с текстом программы из Листинга 9.2
4. Проверьте работу программы, запустив ее в оболочке GDB

3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно 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 быст-

рее 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.

4 Выполнение лабораторной работы

1) Создаю каталог для выполнения лабораторной работы № 9, перехожу в него и создаю файл lab09-1.asm

Рис. 4.1: 1)Создание файла

2) Ввожу в файл lab09-1.asm текст программы из листинга 9.1. Создаю исполняемый файл и проверяю его работу (2-3)

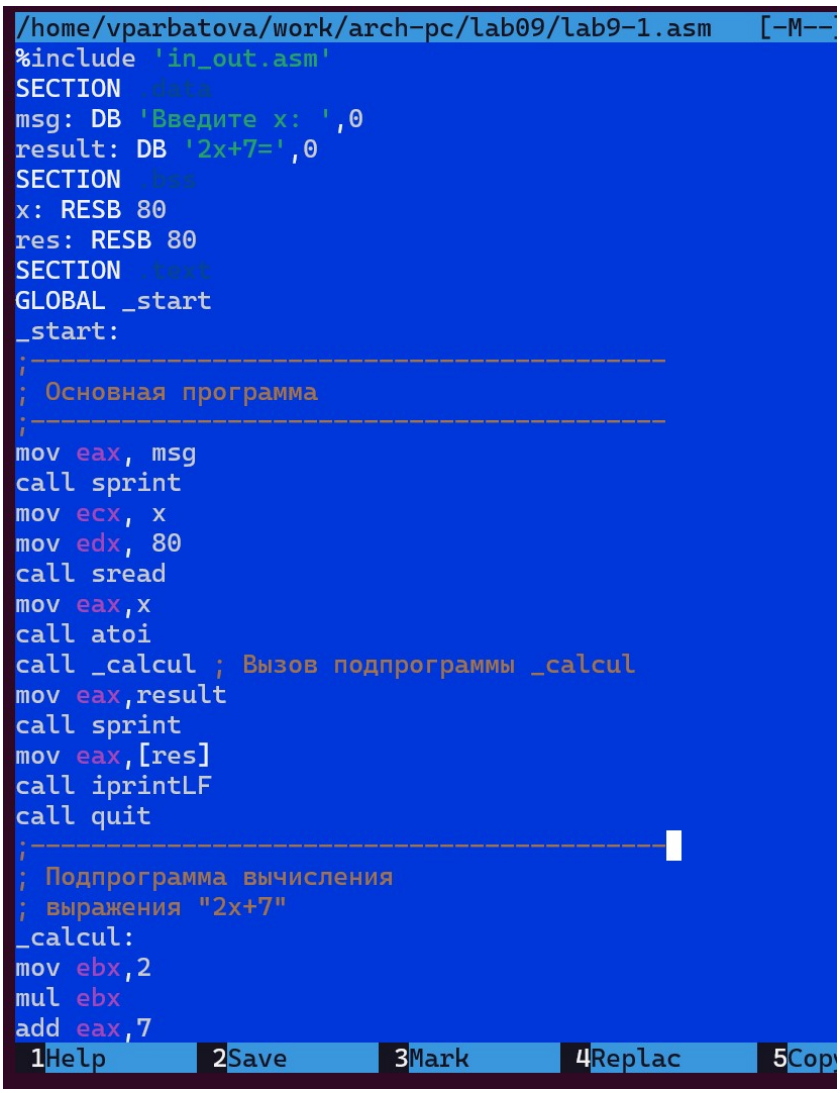

Рис. 4.2: 2)Текст файла

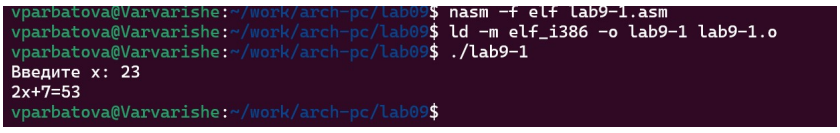

Рис. 4.3: 3)Создание файла

3) Изменяю текст программы, добавив подпрограмму _subcalcul в подпрограмму _calcul, для вычисления выражения ⊠(⊠(⊠)), где ⊠ вводится с клавиатуры, ⊠(⊠) = 2⊠ + 7, ⊠(⊠) = 3⊠ − 1. Создаю файл и проверяю его работу (4-5)

```
;---------------------------------------------
; Основная программа
;---------------------------------------------
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax,x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax,result
call sprint
mov eax,[res]
call iprintLF
call quit
;---------------------------------------------
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
call _subcalcul ; Вызов подпрограммы _subcalcul
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret ; выход из подпрограммы

_subcalcul:
mov ebx, 3
mul ebx
sub eax, 1
mov [res], eax
ret
```

Рис. 4.4: 4)Текст программы изменённый

```
vparbatova@Varvarishe:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
vparbatova@Varvarishe:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
vparbatova@Varvarishe:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 23
2(3x-1)+7=143
vparbatova@Varvarishe:~/work/arch-pc/lab09$
```

Рис. 4.5: 5)Работа файла

4) Создаю файл lab09-2.asm с текстом программы из Листинга 9.2. (6-8)

Рис. 4.6: 6) Создание файла

Рис. 4.7: 7) Текст файла

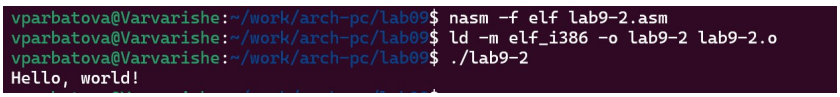

Рис. 4.8: 8) Работа файла

5) Получаю исполняемый файл.Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’. Скачиваю gdb и запускаю его (9-10)

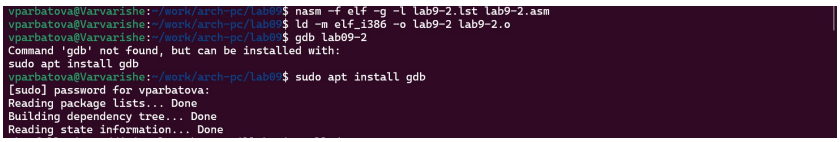

Рис. 4.9: 9) Скачиваю gdb

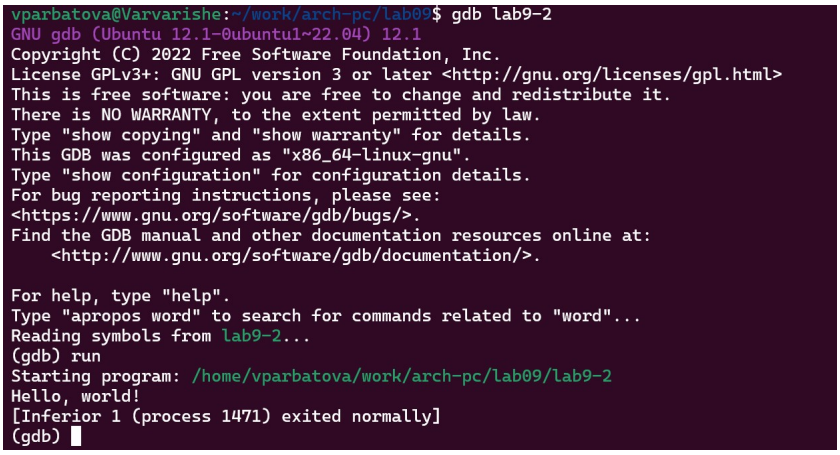

Рис. 4.10: 10) Запуск gdb и файла

6) Для более подробного анализа программы установливаю брейкпоинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запускаю её

Рис. 4.11: 11) Установка break_point и запуск программы с ней

7) Смотрю дисассимилированный код программы с помощью команды disassemble начиная с метки _start

```
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:     mov    $0x4,%eax
   0x08049005 <+5>:     mov    $0x1,%ebx
   0x0804900a <+10>:    mov    $0x804a000,%ecx
   0x0804900f <+15>:    mov    $0x8,%edx
   0x08049014 <+20>:    int    $0x80
   0x08049016 <+22>:    mov    $0x4,%eax
   0x0804901b <+27>:    mov    $0x1,%ebx
   0x08049020 <+32>:    mov    $0x804a008,%ecx
   0x08049025 <+37>:    mov    $0x7,%edx
   0x0804902a <+42>:    int    $0x80
   0x0804902c <+44>:    mov    $0x1,%eax
   0x08049031 <+49>:    mov    $0x0,%ebx
   0x08049036 <+54>:    int    $0x80
End of assembler dump.
(gdb)
```

Рис. 4.12: 12)Дисассимилированный код

8) Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду set disassembly-flavor intel

```
(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:     mov    eax,0x4
   0x08049005 <+5>:     mov    ebx,0x1
   0x0804900a <+10>:    mov    ecx,0x804a000
   0x0804900f <+15>:    mov    edx,0x8
   0x08049014 <+20>:    int    0x80
   0x08049016 <+22>:    mov    eax,0x4
   0x0804901b <+27>:    mov    ebx,0x1
   0x08049020 <+32>:    mov    ecx,0x804a008
   0x08049025 <+37>:    mov    edx,0x7
   0x0804902a <+42>:    int    0x80
   0x0804902c <+44>:    mov    eax,0x1
   0x08049031 <+49>:    mov    ebx,0x0
   0x08049036 <+54>:    int    0x80
End of assembler dump.
(gdb)
```

Рис. 4.13: 13)Отображение команд с Intel'овским синтаксисом

9) Включаю режим псевдографики для более удобного анализа программы

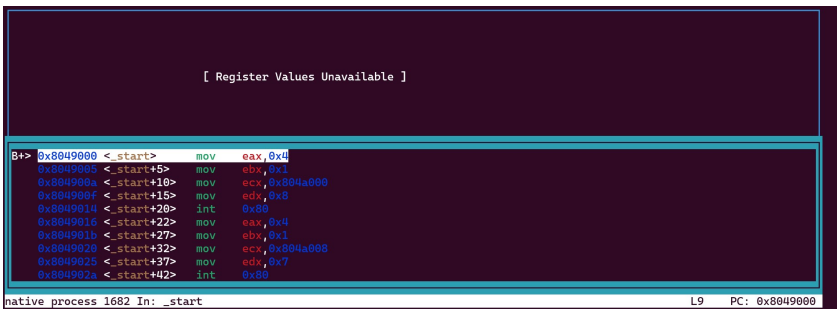

Рис. 4.14: 14)Режим псевдографики

10) С помощью info breakpoints узнаю информацию об установленных точках останова

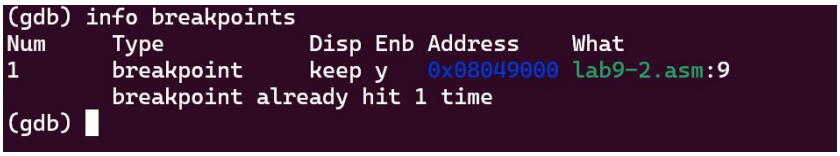

Рис. 4.15: 15)info breakpoints

11) Определяю адрес предпоследней инструкции (mov ebx,0x0) и устанавливаю точку останова. Смотрю информацию об установленных точках останова (16-17)

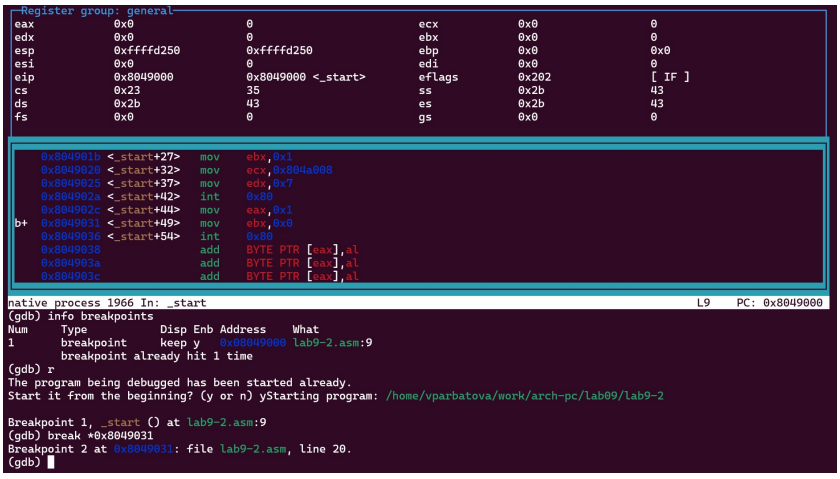

Рис. 4.16: 16)Установка точки останова

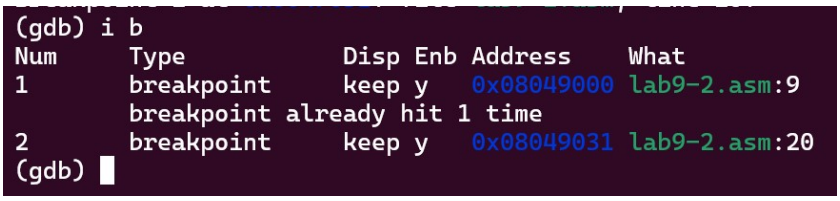

Рис. 4.17: 17)Вывод информации о точках останова

12) Выполняю 5 инструкций с помощью команды stepi (или si) и слежу за изменением значений регистров. Значения регистров eax, edx, ecx, esp, eip, cs, ds, ebx, ss, eflags, es изменяются

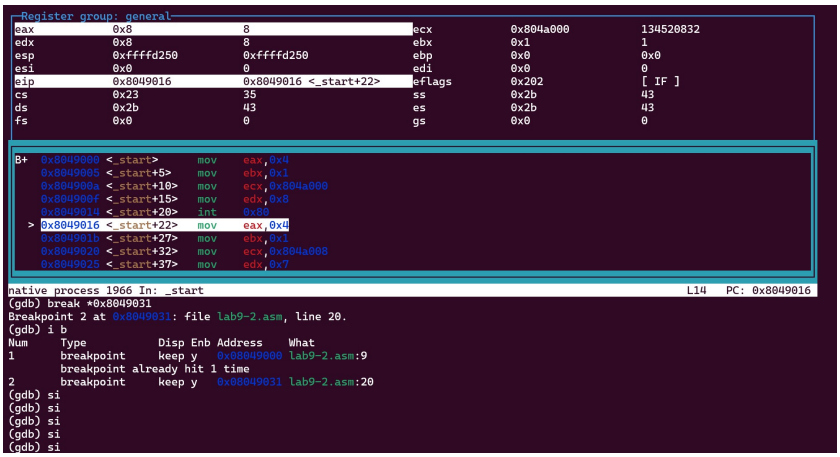

Рис. 4.18: 18)5 инструкций si

13) Посмотреть содержимое регистров также можно с помощью команды info registers

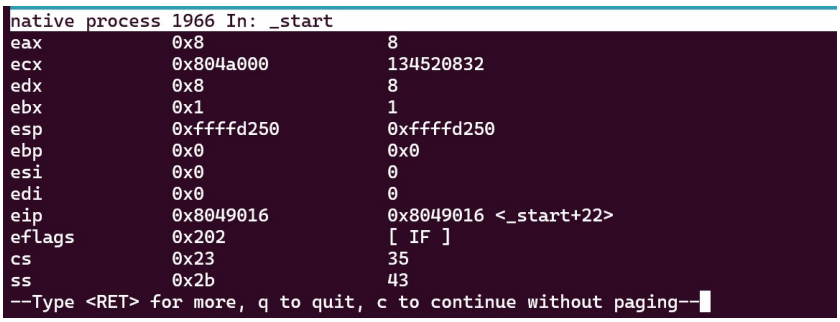

Рис. 4.19: 19)info registers

14) Смотрю значение переменной msg1 по имени и переменной msg2 по адресу (20-21)

Рис. 4.20: 20)Значение переменной msg1 по имени

Рис. 4.21: 21)Значение переменной msg2 по адресу

15) Изменяю первый символ переменной msg1

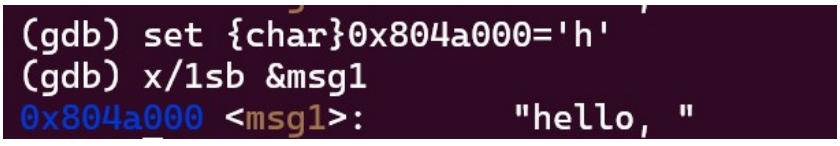

Рис. 4.22: 22)Изменение первого символа переменной msg1

16) Заменяю символ во второй переменной msg2

Рис. 4.23: 23)Изменение символа второй переменной

17) Вывожу в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx.

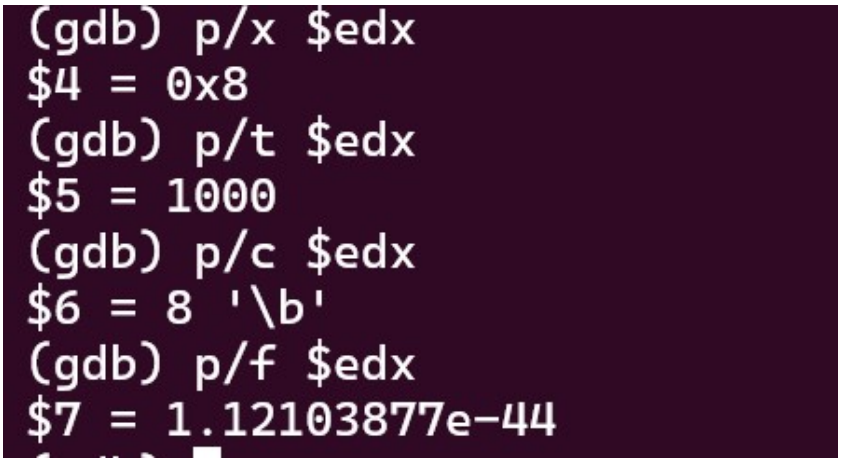

Рис. 4.24: 24)Значение регистра edx

18) С помощью команды set изменяю значение регистра ebx. В первом случае выводит значение символа (его код), во втором - число

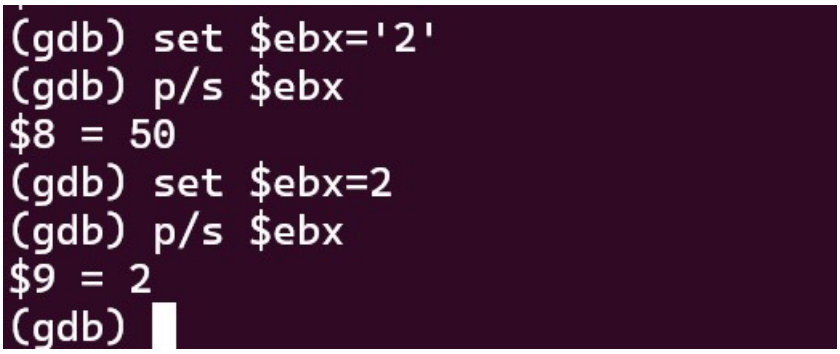

Рис. 4.25: 25)Изменяю значение регистра ebx

19) Завершаю выполнение программы с помощью команды continue (сокращенно c) и выхожу из GDB с помощью команды quit (сокращенно q)

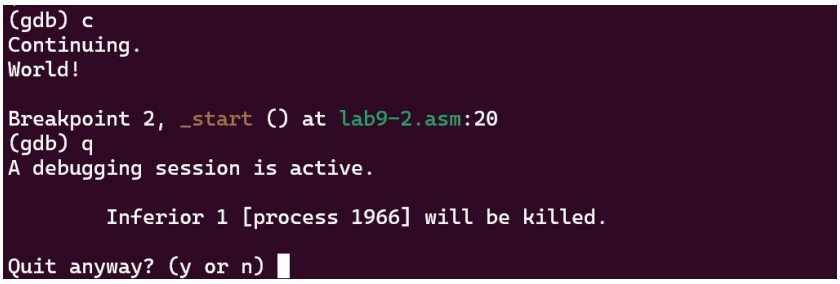

Рис. 4.26: 26)Завершаю программу

20) Копирую файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm, создаю исполняемый файл, загружаю исполняемый файл в отладчик, указав аргументы:

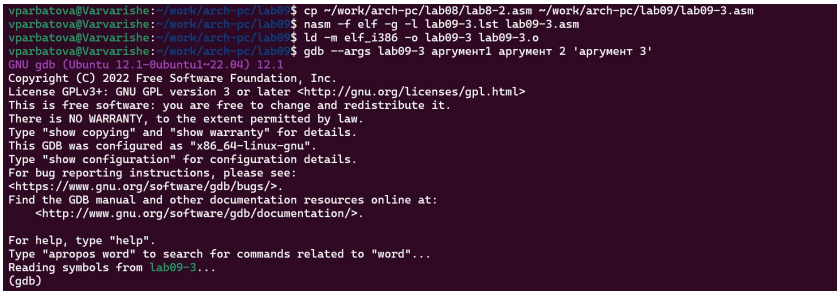

Рис. 4.27: 27)Копирование, создание, загрузка файла

21) Для начала уставливаю точку останова перед первой инструкцией в программе и запускаю её.

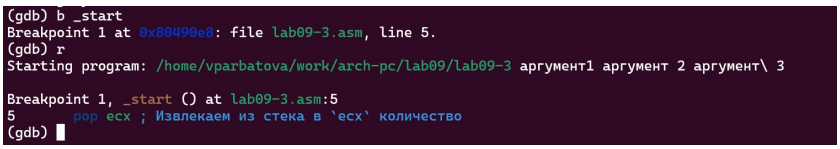

Рис. 4.28: 28)Точка установа и запуск

22) Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы)

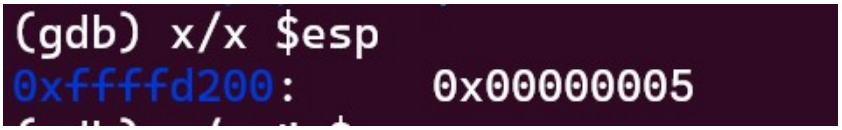

Рис. 4.29: 29)Регистр esp

23) Смотрю остальные позиции стека. Шаг изменения равен 4 потому что шаг - int, а под этот тип данных выделяется 4 байта

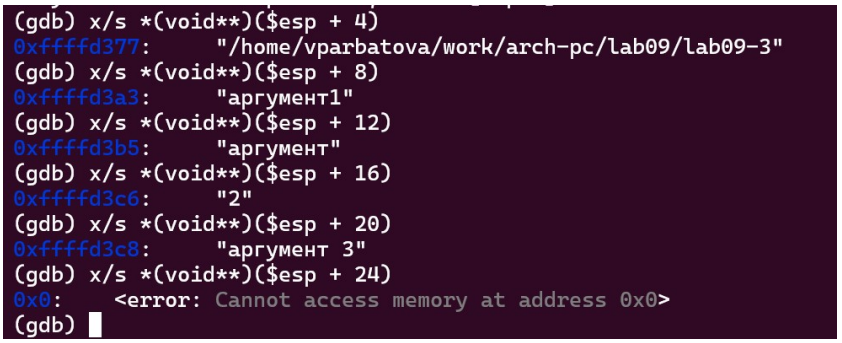

Рис. 4.30: 30)Остальные позиции стека

5 Выполнение заданий для самостоятельной работы

1) Копирую файл

Рис. 5.1: Копирование файла

2) Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции ⊠(⊠) как подпрограмму

Текст программы:

%include ‘in_out.asm’ SECTION .data msg db “Результат:”,0 msg1 db “Функция: f(x) = 2x + 15”,0 SECTION .text global _start _start: 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 _function loop next _end: mov eax, msg1 call sprintLF mov eax, msg ; вывод сообщения “Результат:” call sprint mov eax,

esi ; записываем сумму в регистр `eax` call iprintLF ; печать результата call quit ; завершение программымcedit lab09-4.asm _function: mov ebx, 2 mul ebx add eax, 15 add esi,eax ; добавляем к промежуточной сумме ; след. аргумент `esi=esi+eax` ret

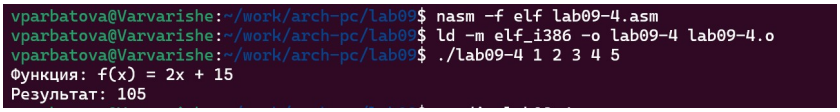

Рис. 5.2: Работа файла

3) Создаю файл, ввожу туда текст программы

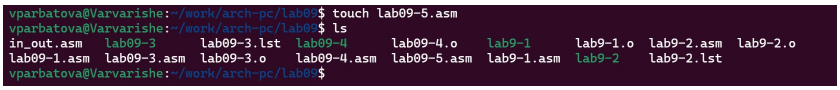

Рис. 5.3: Создание файла

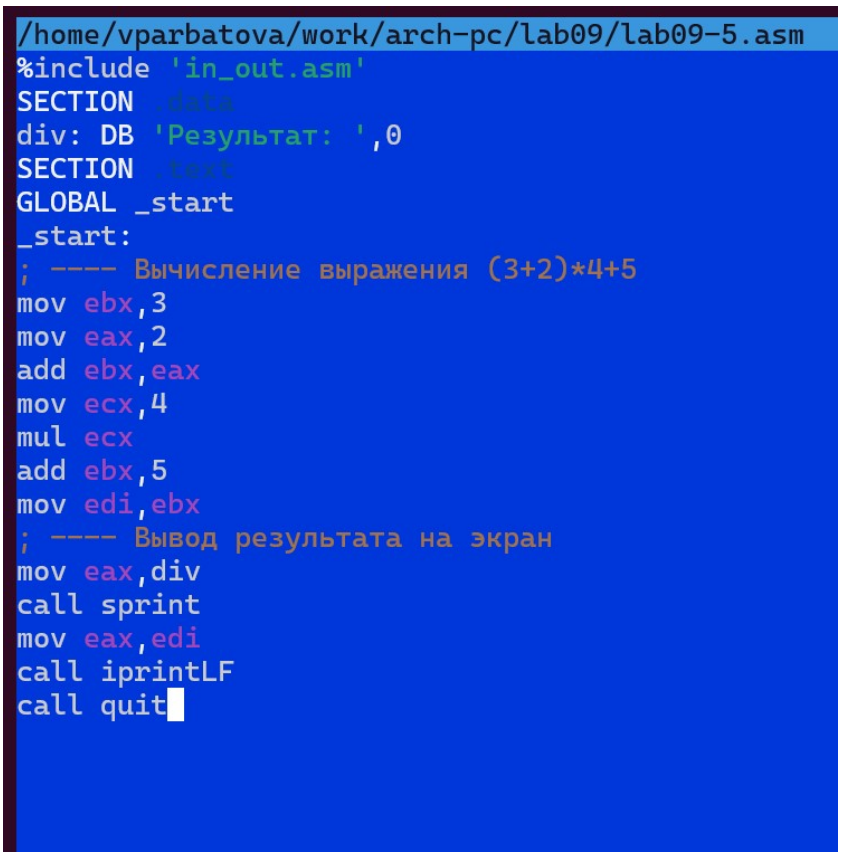

Рис. 5.4: Текст файла

4) Передаю файл на отладку

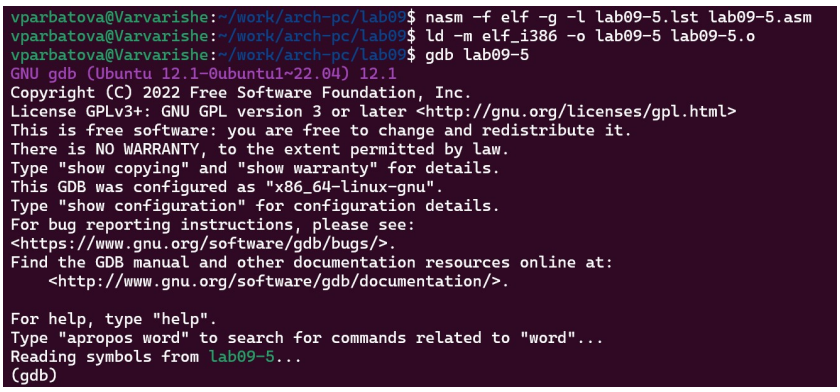

Рис. 5.5: Передача файла

5) Устанавливаю точки останова

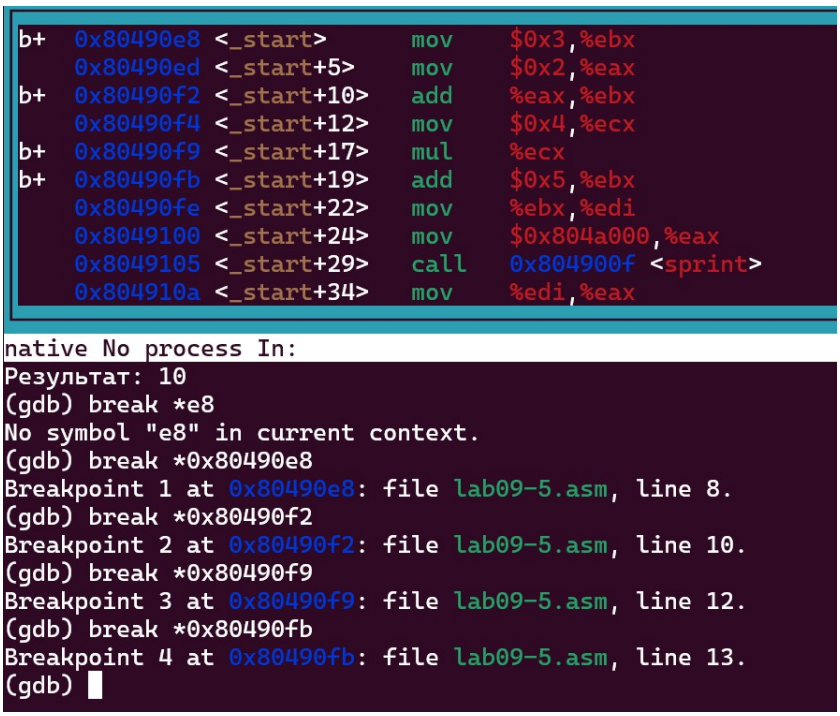

Рис. 5.6: Точки останова

6) Прохожу по программе обращая внимание на несостыковки с логикой. Таким образом нашла 3 ошибки

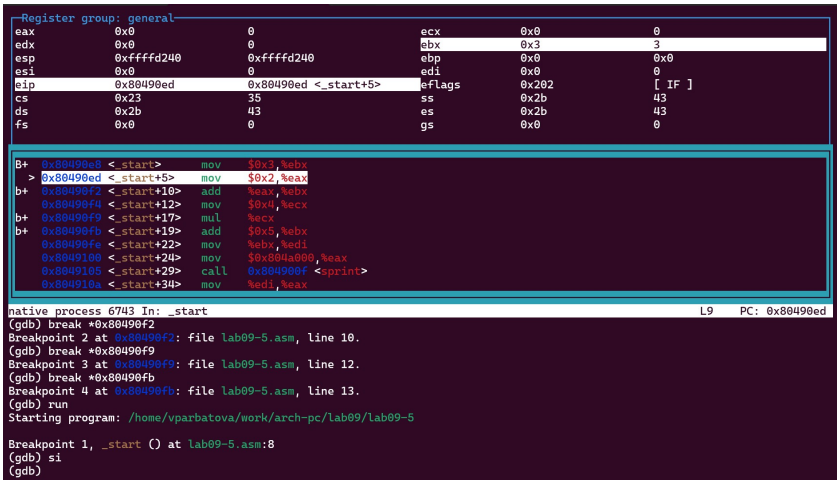

Рис. 5.7: Отладка файла

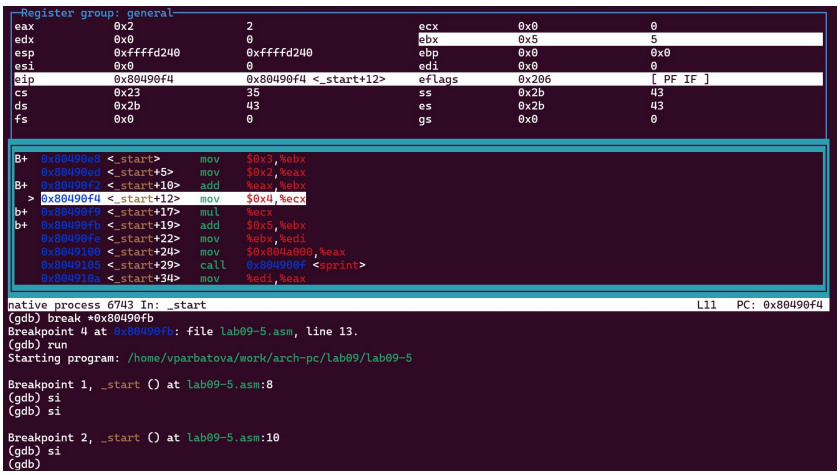

Рис. 5.8: Первая ошибка

7) Исправляю ошибки

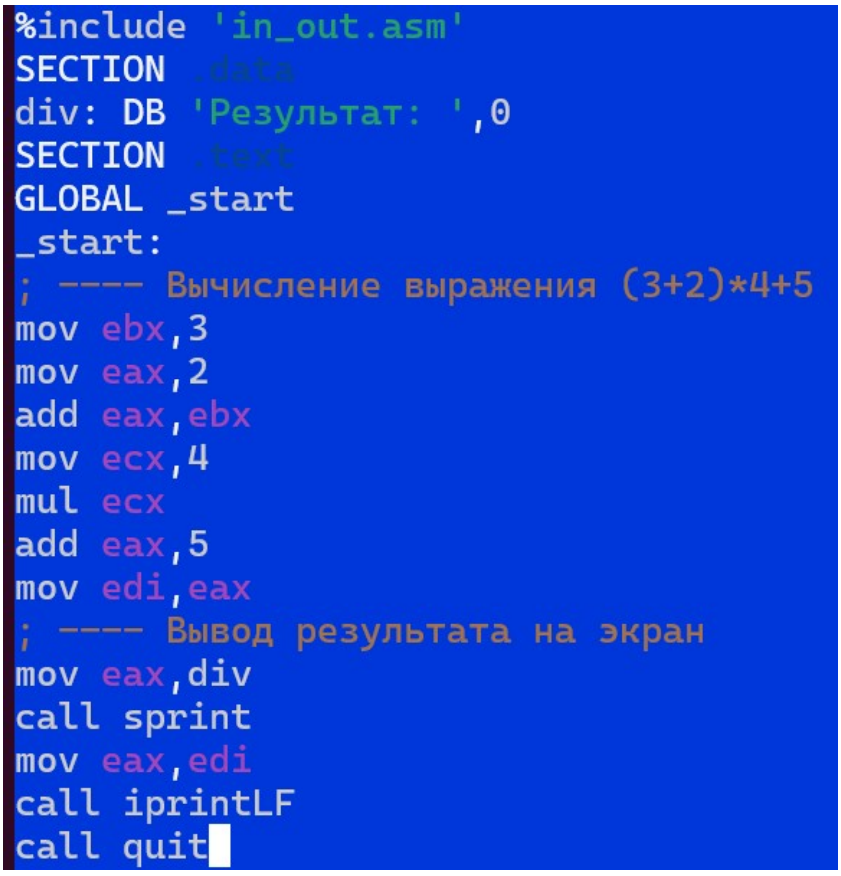

Рис. 5.9: Исправленный текст файла

8) Работа файла

```
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

Рис. 5.10: Работа файла

6 Выводы

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

Список литературы