Лабораторная работа №9

НКАбд-02-23

Выборнов Дмитрий Валерьевич

1 Цель работы

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

2 Задание

1. Реализация подпрограмм в NASM.
2. Отладка программам с помощью GDB.
3. Добавление точек останова.
4. Обработка аргументов командной строки в GDB.
5. Задание для самостоятельной работы.

3 Теоретическое введение

3.1 Отладка.

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.

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 программаотладчик приостанавливает выполнение программы и ждёт команд.

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.

3.2 Подпрограммы.

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.

Подпрограмма может вызываться как из внешнего файла, так и быть частью основной программы.

4 Выполнение лабораторной работы

4.1 Реализация подпрограмм в NASM.

Создаю новый файл и ввожу в него текст программы.

```
 GNU nano 6.2
x:  RESB 80
res:  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, [res]
 call iprintLF

 call quit

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

 ret
```

Рис. 4.1: Первый шаг.

Создаю исполняемый файл и проверяю его работу.

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ nasm -f elf lab9-1.asm
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1 lab9-1.o
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ ./lab9-1
Введите x: 6
2x + 7 = 19
```

Рис. 4.2: Второй шаг.

Изменяю текст программы.

```
%include 'in_out.asm'
SECTION .data
 msg:  DB 'Введите x: ', 0
 result:  DB 'f(g(x)) = ', 0

SECTION .bss
 x:  RESB 80
 res:  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, [res]
  call iprintLF

  call quit

 _calcul:

  call _subcalcul
```

Рис. 4.3: Третий шаг.

Запускаю изменённую программу и проверяю её работу.

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ nasm -f elf lab9-1-2.asm
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-1-2 lab9-1-2.o
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ ./lab9-1-2
Введите x: 5
f(g(x)) = 35
```

Рис. 4.4: Четвёртый шаг.

4.2 Отладка программам с помощью GDB.

Создаю новый файл и ввожу в него текст программы вывода Hello, world!

```
SECTION .data
 msg1:  db 'Hello, ', 0x0
 msg1Len:  equ $ - msg1

 msg2:  db 'world!', 0xa
 msg2Len:  equ $ - msg2

SECTION .text
GLOBAL _start
 _start:

  mov eax, 4
  mov ebx, 1
  mov ecx, msg1
  mov edx, msg1Len
  int 0x80

  mov eax, 4
  mov ebx, 1
  mov ecx, msg2
  mov edx, msg2Len
  int 0x80

  mov eax, 1
  mov ebx, 0
  int 0x80
```

Рис. 4.5: Пятый шаг.

Получаю исполняемый файл, загружаю его в GDB и проверяю его при помощи команды run.

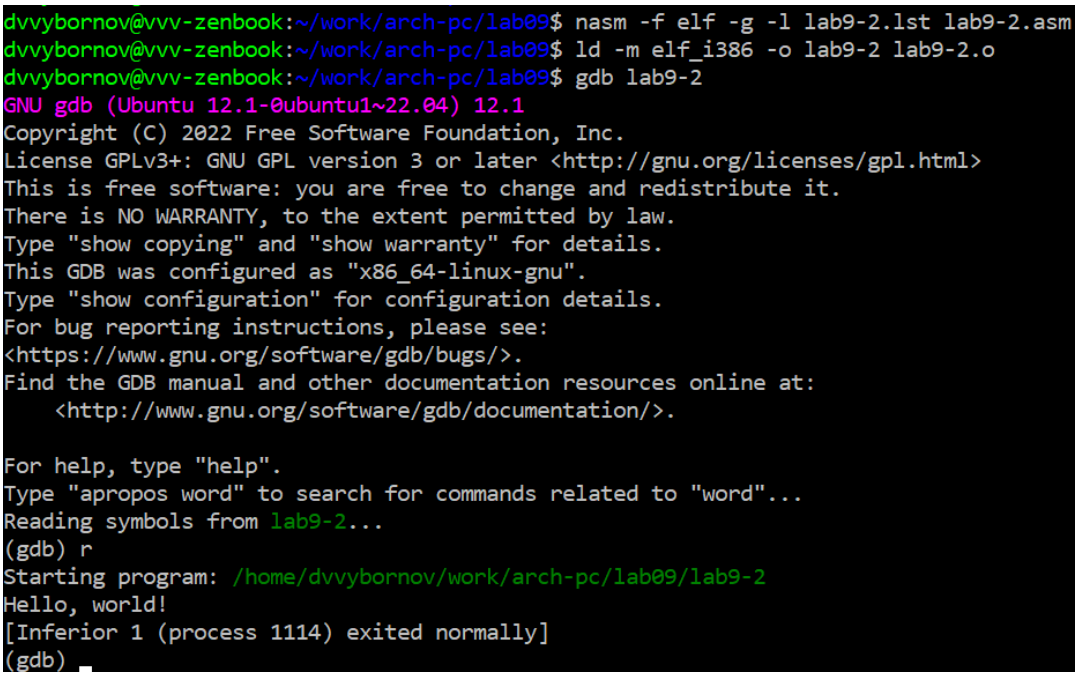


Рис. 4.6: Шестой шаг.

Запускаю прогамму с брейкпоинтом.

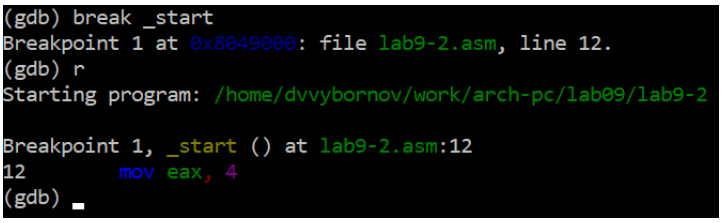


Рис. 4.7: Седьмой шаг.

Изучаю диссимилированный код программы с двумя разными синтаксисами.

Рис. 4.8: Восьмой шаг.

Основные различия отображения синтаксиса машинных команд в режимах ATT и Intel заключаются в отображении третьего столбца данных.

Включаю режим псевдографики.

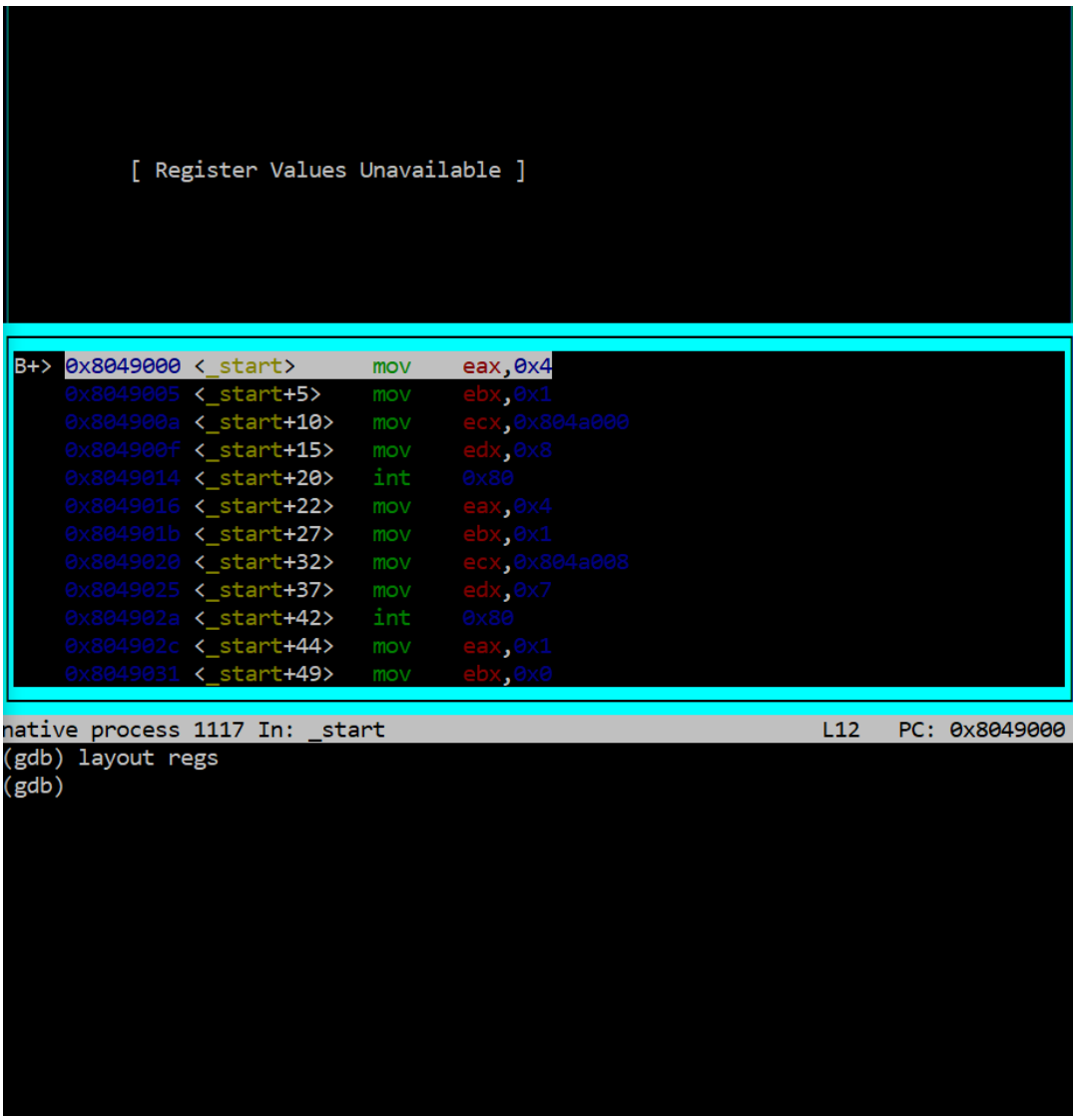


Рис. 4.9: Девятый шаг.

4.3 Добавление точек останова.

Создаю ещё одну точку останова и проверяю информацию о них.

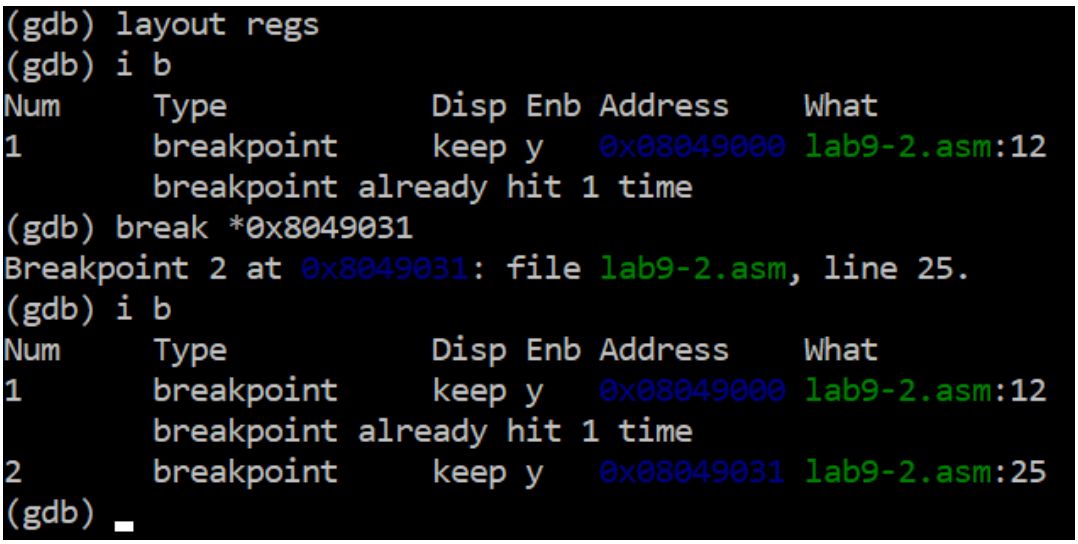


Рис. 4.10: Десятый шаг.

4.4 Работа с данными программы в GDB.

Во время работы программы изменяются значения тех регистров, с которыми она взаимодействует, а также значение регистра eip.

Проверяю информацию о регистрах.

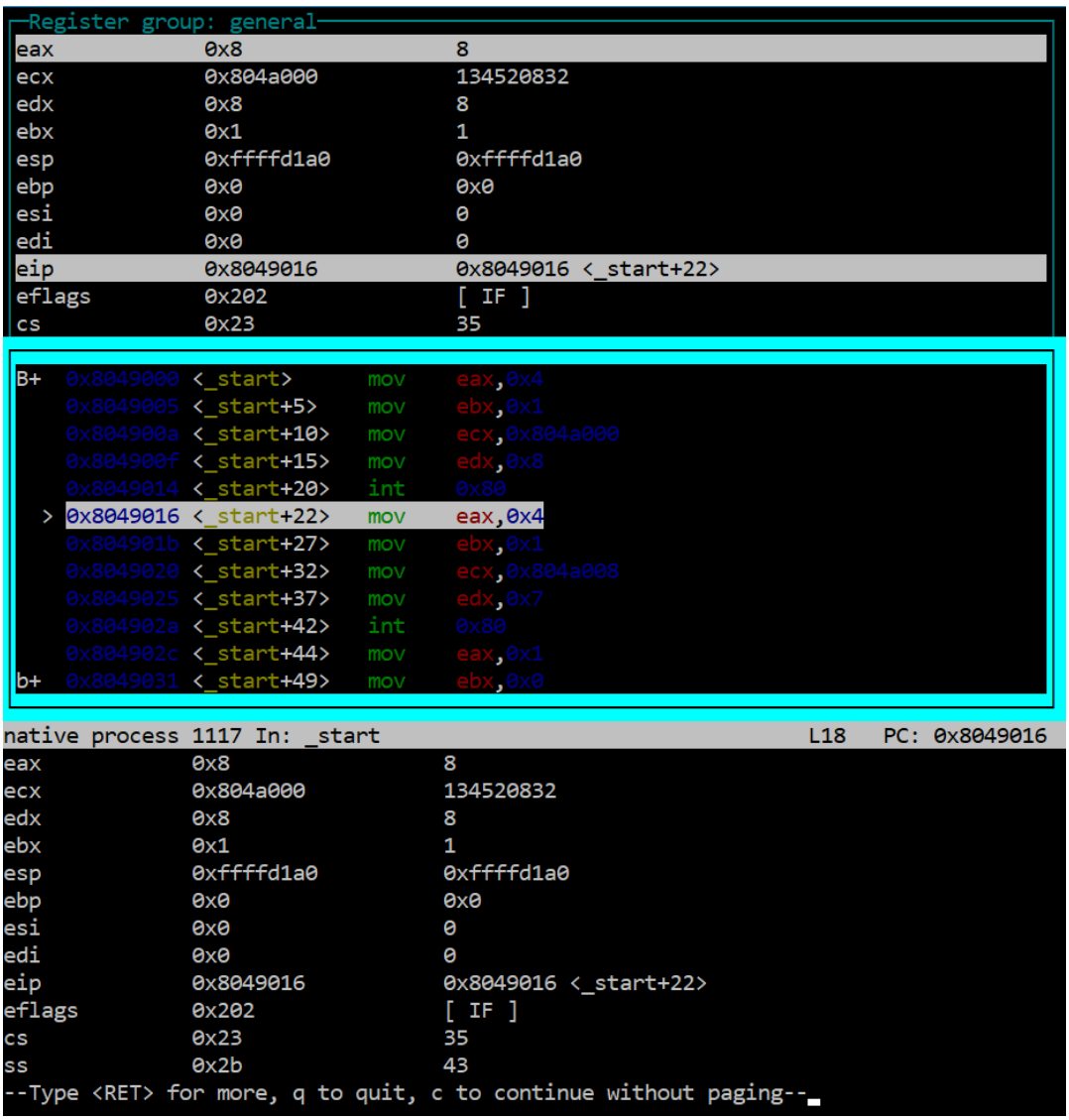


Рис. 4.11: Одиннадцатый шаг.

Просматриваю значение регистров msg1 и msg2.

```
(gdb) x/1sb &msg1
0x804a000 <msg1>:       "Hello, "
(gdb) x/1sb 0x804a008
0x804a008 <msg2>:       "world!\n\034"
(gdb)
```

Рис. 4.12: Двенадцатый шаг.

Изменяю несколько символов в msg1 и msg2.

```
(gdb) set {char} &msg1 = 'h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:       "hello, "
(gdb) set {char} &msg2 = 'b'
(gdb) x/1sb &msg2
0x804a008 <msg2>:       "borld!\n\034"
(gdb)
```

Рис. 4.13: Тринадцатый шаг.

Вывожу значение регистра eax в разных форматах.

```
(gdb) p /x $edx
$2 = 0x8
(gdb) p /t $edx
$3 = 1000
(gdb) p /c $edx
$4 = 8 '\b'
(gdb)
```

Рис. 4.14: Четырнадцатый шаг.

Изменяю значение регистра ebx.

```
(gdb) set $ebx = 2
(gdb) p /s $ebx
$6 = 2
(gdb)
```

Рис. 4.15: Пятнадцатый шаг.

В первый раз изменение не сработало, так как в ebx хранилось численное значение.

4.5 Обработка аргументов командной строки в GDB.

Копирую нужный файл из восьмой лабораторной работы, создаю исполняемый файл и загружаю его в GDB.

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ nasm -f elf -g -l lab9-3.lst lab9-3.asm
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-3 lab9-3.o
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ gdb --args lab9-3 аргумент1 аргумент 2
'аргумент 3'
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab9-3...
```

Рис. 4.16: Шестнадцатый шаг.

Запускаю программу.

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 6.
(gdb) run
Starting program: /home/dvvybornov/work/arch-pc/lab09/lab9-3 аргумент1 аргумент 2 ар
гумент\ 3

Breakpoint 1, _start () at lab9-3.asm:6
6          pop ecx
```

Рис. 4.17: Семнадцатый шаг.

Просматриваю остальные позиции стека.

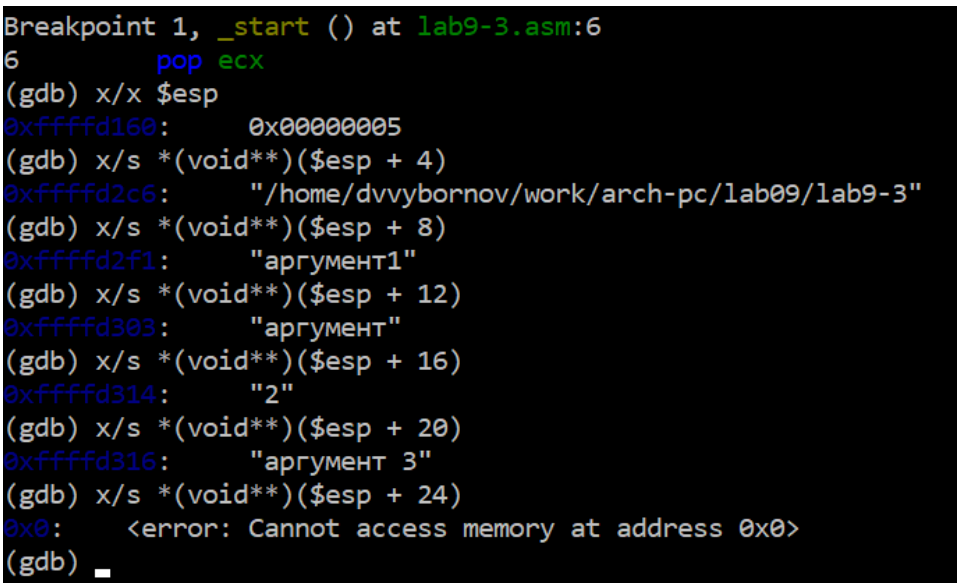


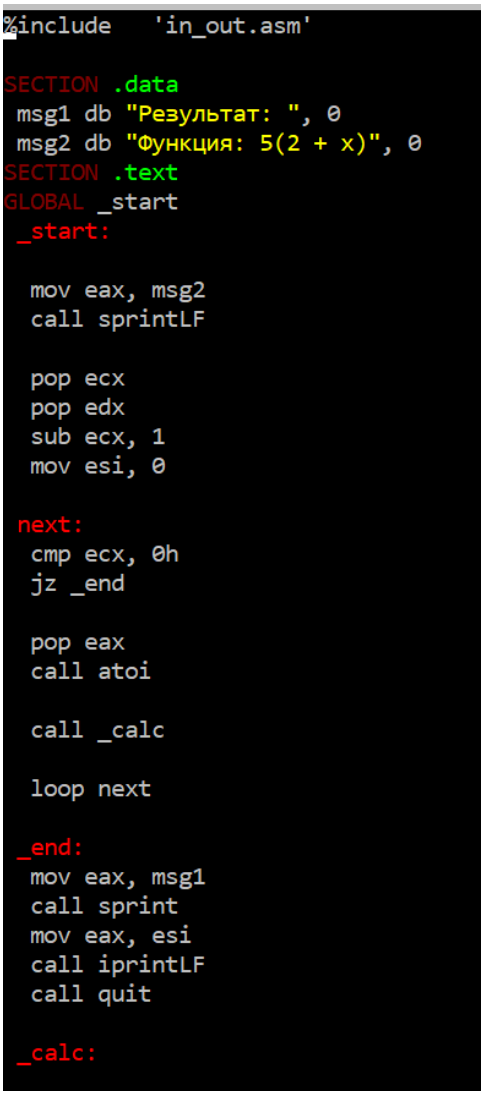
Рис. 4.18: Восемнадцатый шаг.

Шаг изменения адреса равен 4, так как для каждого обьекта в стеке отводится 4 бита.

4.6 Задание для самостоятельной работы.

4.6.1 №1

Преобразовываю программу из восьмой лабораторной работы.



```
%include   'in_out.asm'

SECTION .data
 msg1 db "Результат: ", 0
 msg2 db "Функция: 5(2 + x)", 0
SECTION .text
GLOBAL _start
 _start:

  mov eax, msg2
  call sprintLF

  pop ecx
  pop edx
  sub ecx, 1
  mov esi, 0

 next:
  cmp ecx, 0h
  jz _end

  pop eax
  call atoi

  call _calc

  loop next

 _end:
  mov eax, msg1
  call sprint
  mov eax, esi
  call iprintLF
  call quit

 _calc:
```

Рис. 4.19: Первый шаг первого задания.

Проверяю работу изменённой программы.

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ nasm -f elf lab9-4.asm
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-4 lab9-4.o
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ ./lab9-4 8 18 28
Функция: 5(2 + x)
Результат: 300
```

Рис. 4.20: Второй шаг первого задания.

4.6.2 №2

Копирую текст нужной программы в новый файл.

```
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:

mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx

mov eax,div
call sprint
mov eax,edi
call iprintLF

call quit
```

Рис. 4.21: Первый шаг второго задания.

Запускаю эту программу при помощи GDB.

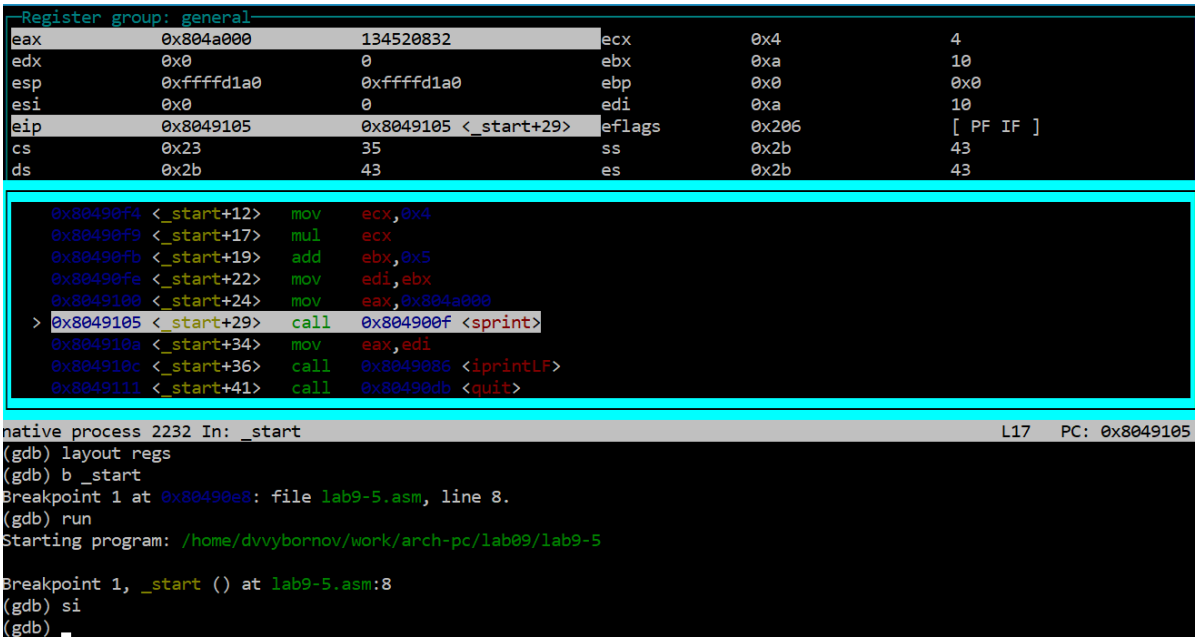


Рис. 4.22: Второй шаг второго задания.

Все проблемы этой программы связаны с тем, как в ней использованы регистры.

Исправляю текст программы.

```
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:

mov ebx,3
mov eax,2
add eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edx, eax

mov eax, div
call sprint
mov eax,edx
call iprintLF

call quit
```

Рис. 4.23: Третий шаг второго задания.

Проверяю работу программы.

```
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ nasm -f elf lab9-5.asm
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab9-5 lab9-5.o
dvvybornov@vvv-zenbook:~/work/arch-pc/lab09$ ./lab9-5
Результат: 25
```

Рис. 4.24: Четвёртый шаг второго задания.

Теперь прогдамма даёт правильный результат.

5 Выводы

Выполнив эту лабораторную работу, я приобрёл навыки написания программ с использованием подпрограмм и познакомился с методами отладки при помощи GDB и его основными возможностями.