Отчётпо лабораторной работе №9

Дисциплина: Архитектура компьютера

Сандан Алина Эдуардовна НКАбд-04-24

Содержание

Список иллюстраций

Список таблиц

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. Работа с данными программы в GDB
5. Обработка аргументов командной строки в GDB
6. Задание для самостоятельной работы

3 Выполнение лабораторной работы

3.1 Реализация подпрограмм в NASM

По аналогии с предыдущими лабораторными работами, файлы asm я буду создавать в каталоге “work/arch-pc”. В файле lab9-1.asm вписываю программу из листинга 9.1, которую я внимательно изучила. Создаю исполняемый файл и запускаю его в работу. (рис. 3.1).

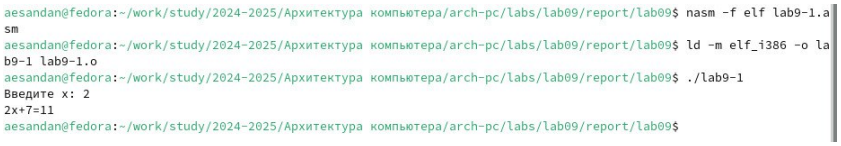
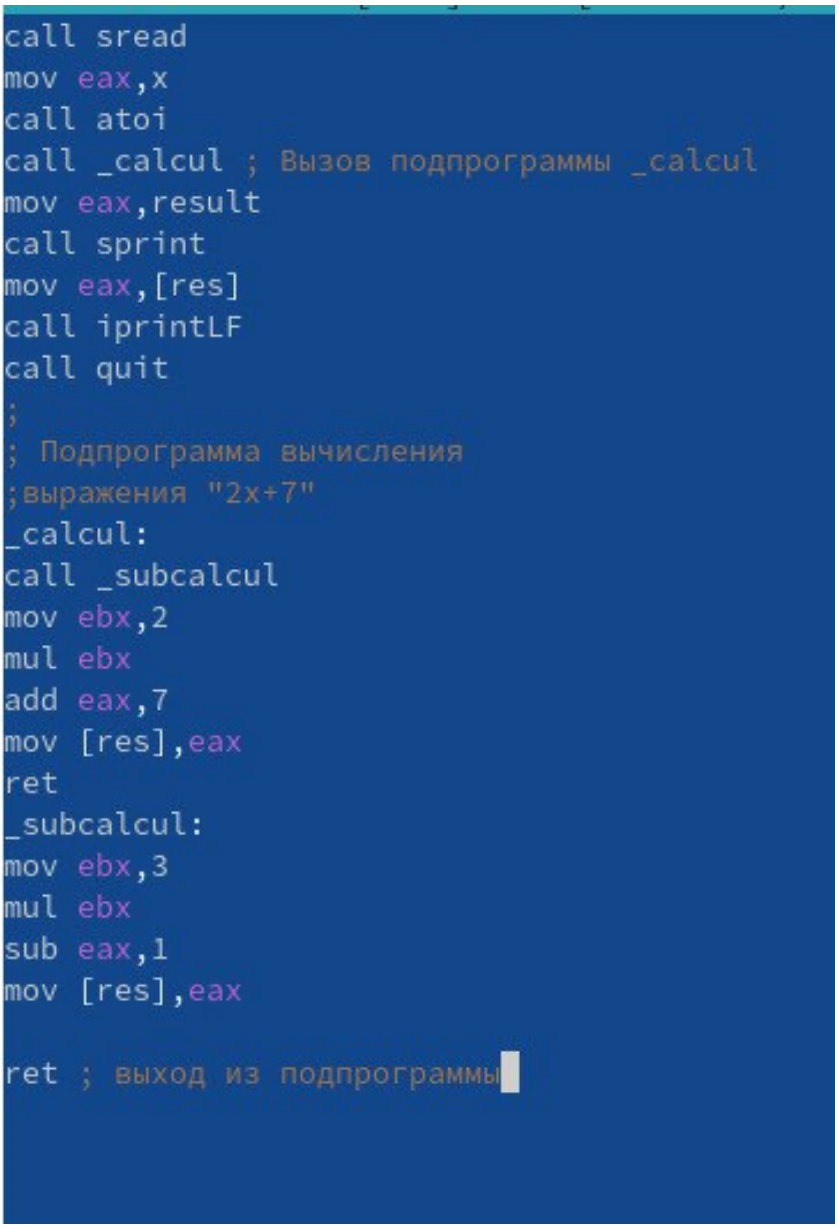


Рис. 3.1: Первые действия в лабораторной работе и результат программы

После этого я переписываю программу согласно методическому материалу (рис. 3.2).



```
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
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret
_subcalcul:
mov ebx,3
mul ebx
sub eax,1
mov [res],eax

ret ; выход из подпрограммы
```

Рис. 3.2: Текст программы

Создаю исполняемый файл и убеждаюсь в том, что все работает успешно.(рис. 3.3).

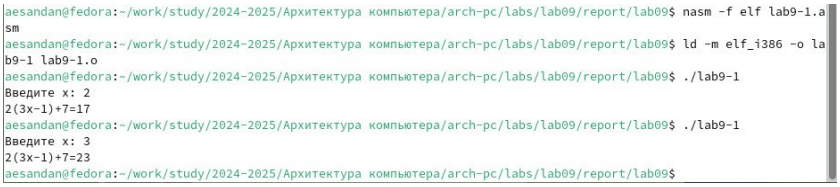


Рис. 3.3: Результат программы

3.2 Отладка программам с помощью GDB

В новом файле lab9-2.asm записываю текст программы из листинга 9.2. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ провожу с ключом ‘-g’. И загружаю файл в откладчик gbd, а затем проверяю выполнение программы с помощью команды run (рис. 3.4).

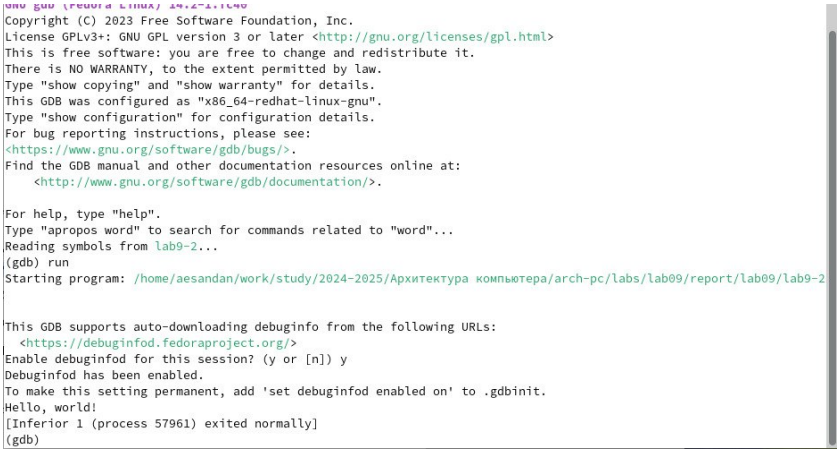


Рис. 3.4: Работа программы

Для более подробного анализа программы устанавливаю брейкпоинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запускаю её. (рис. 3.5).

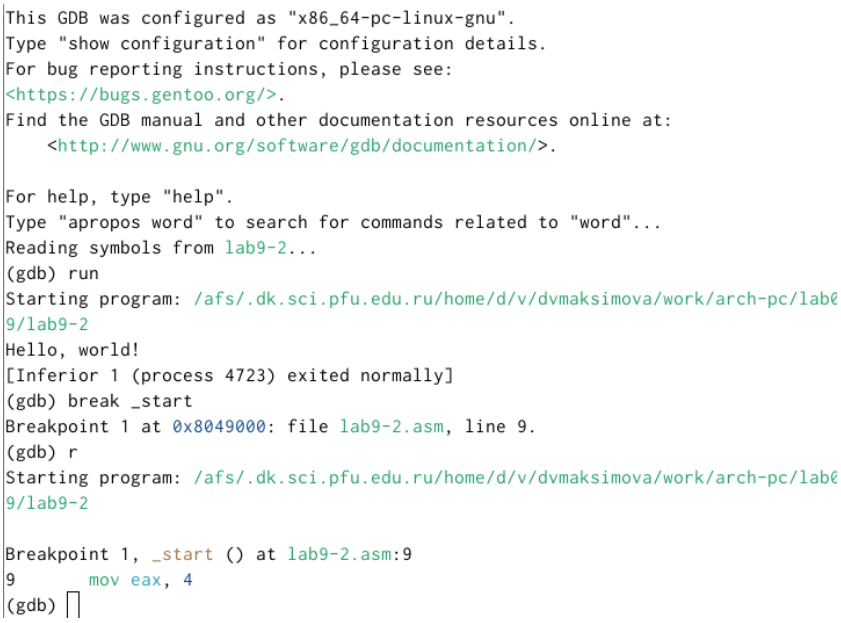


Рис. 3.5: Брейкпоинт

Просматриваю дисассимилированный код программы с помощью команды disassemble начиная с метки _start (рис. 3.6).

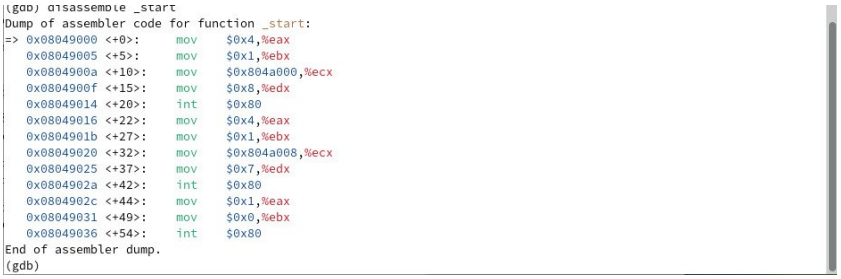


Рис. 3.6: Дисассимилированный код программы

Переключаюсь на отображение команд с Intel'овским синтаксисом, введя команду set disassembly-flavor intel (рис. 3.7).

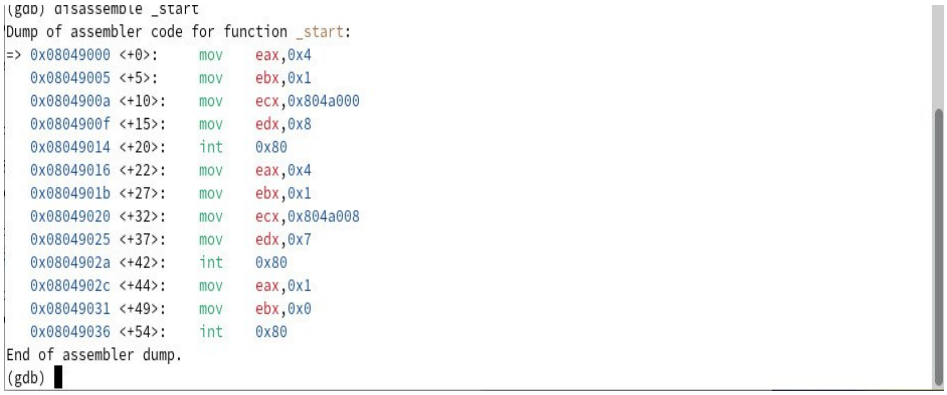


Рис. 3.7: Отображение команд

Есть некоторые различия в отображениях в этих режимах, а именно в виде колонки с текстом программы: в Intel'е она выглядит, как *"$0x{операнд},%{регистер}"*, а в АТТ - *"{регистер},0x{операнд}"*

Включаю режим псевдографики для удобного анализа программ (рис. 3.8).

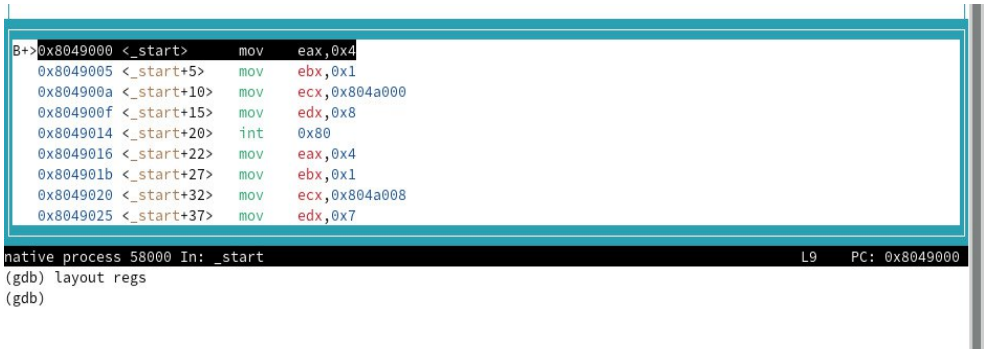


Рис. 3.8: Псевдографика.Режим

3.3 Добавление точек останова

На предыдущих шагах была установлена точка останова по имени метки (_start). Проверяю это с помощью команды info breakpoints (рис. 3.9).

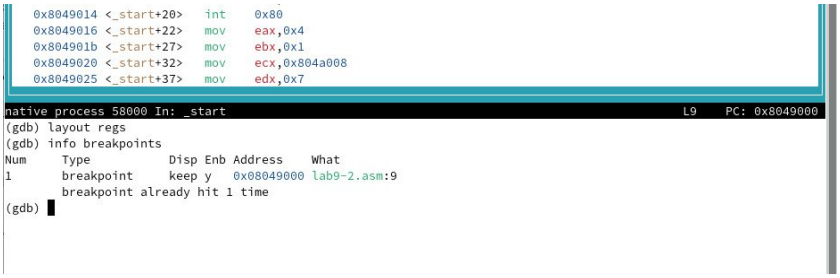


Рис. 3.9: Информация о точке остановы

Я установила еще одну точку останова по адресу инструкции, и теперь когда мы запрашиваем информацию о брейкпоинтах, нам показываются две точки останова: (рис. 3.10).

Рис. 3.10: Точка останова

3.4 Работа с данными программы в GDB

Для того, что бы посмотреть значения регистров можно воспользоваться командой “i r”. Вот какие значения регистров вывелись (рис. 3.11).

Рис. 3.11: Значения регистров

С помощью команды x & также можно посмотреть содержимое переменной. Я посмотрела значение переменной msg1 по имени и вот результат. (рис. 3.12).

Рис. 3.12: Просмотр значения переменной

Аналогично можно посмотреть значение переменной msg2 по адресу. (рис. 3.13).

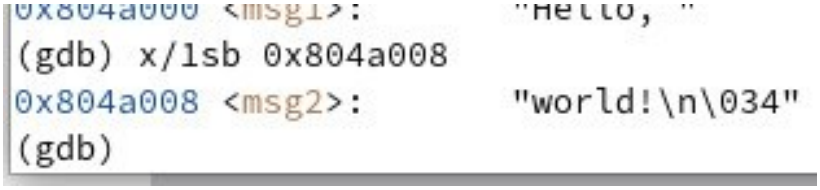


Рис. 3.13: Значение другой переменной

Теперь я изменяю первый символ переменной msg1 (рис. 3.14).

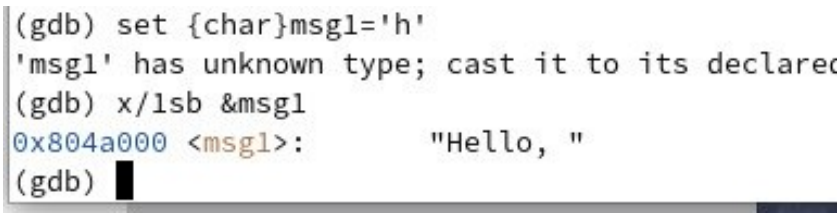


Рис. 3.14: msg1=hello,

А также я изменила разные символы в переменной msg2 (рис. 3.15).

```
0x804a000 <msg1>:       "Hello, "
(gdb) set {char}0x804a00d='a'
(gdb) x/1sb &msg2
0x804a008 <msg2>:       "worlda\n\034"
(gdb)
```

Рис. 3.15: Измененные символы

Теперь изменим значение регистра ebx: (рис. 3.16).

```
0x804a008 <msg2>:       "worlda\n\034"
(gdb) set $ebx='2'
(gdb) p/s $ebx
$1 = 50
(gdb)
```

Рис. 3.16: Изменение ebx

Команда set позволяе изменить значение регистра, проделываем это с регистром ebx, и использую команду p/s, чтоб промотреть значение.

3.5 Обработка аргументов командной строки в GDB

Создаю исполняемый файл из файла, с которым я работала в процессе выполнения лбораторной работы №8.

А затем загружаю данный файл в отладчик, предварительно указав ключ к работе с аргументами, и указываю эти самые аргументы (рис. 3.17).

```
aesandan@fedora:~/work/study/2024-2025/lab09$ ld -m elf_i386 -o lab9-3 lab9-3.o
aesandan@fedora:~/work/study/2024-2025/lab09$ gdb --args lab9-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Fedora Linux) 14.2-1.fc40
Copyright (C) 2023 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-redhat-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"...
```

Рис. 3.17: Переход к отладчику

Для начала устанавливаем первую точку останова, а затем запускаем программу (рис. 3.18).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-3.asm, line 5.
(gdb) r
Starting program: /home/aesandan/work/study/2024-2025/lab09/lab9-3 аргумент1 аргумент 2 аргумент\ 3
```

Рис. 3.18: b-point + run

На данном этапе, мы можем просмотреть позиции стека отдельно по адресам (рис. 3.19).

```
(gdb) x/x $esp
0xffffd050:     0x00000005
(gdb) x/s *(void**)($esp + 4)
0xffffd210:     "/home/aesandan/work/study/2024-2025/lab09/lab9-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd241:     "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffd253:     "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffd264:     "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd266:     "аргумент 3"
```

Рис. 3.19: Позиции стека

3.6 Задание для самостоятельной работы

Переходим к выполнению заданий самостоятельного характера.

№1

Я преобразовываю программу , с которой я работала в процесее 8 лабораторной работы. Я должна реализовать вычисление функции в подпрограмме (до этого вычисления были в основной программе).

Вот таким образом выглядит код программы: (рис. 3.20).

```
%include 'in_out.asm'
SECTION .data
msg1 db "Функция: f(x)=3x-1", 0
msg2 db "Результат: ",0
SECTION .text
global _start
_start:

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

next:
cmp ecx,0h
jz _end
pop eax
call atoi

call mom

add esi,eax
loop next
_end:
mov eax, msg1
call sprintLF
mov eax, msg2
call sprint
mov eax, esi
call iprintLF
call quit

mom:
mov ebx, eax
add eax, ebx
add eax, ebx
sub eax,1
ret

```

Рис. 3.20: Код программы

4 Выводы

Я приобрела навыки написания программ с использованием подпрограмм, познакомилась с методами отладки при помощи GDB и его основными возможностями.