

Лабораторная работа №10

Понятие подпрограммы. Отладчик GDB

Медникова Екатерина Михайловна

Содержание

1	Цель работы	5
2	Теоретическое введение	6
3	Выполнение лабораторной работы	8
4	Самостоятельная работа	20
5	Выводы	22

Список иллюстраций

3.1	Создание каталога и файла	8
3.2	Ввод текста программы в файл	9
3.3	Создание и проверка работы файла	9
3.4	Изменение текста программы	10
3.5	Изменение текста программы	11
3.6	Создание файла	11
3.7	Ввод текста программы	12
3.8	Создание исполняемого файла и загрузка в отладчик	12
3.9	Проверка работы программы	13
3.10	Анализ программы	13
3.11	disassemble	13
3.12	set disassembly-flavor intel	14
3.13	Режим псевдографики	14
3.14	Точка останова	15
3.15	Установка точки останова	15
3.16	Информация о точках останова	15
3.17	Содержимое регистров	15
3.18	Значение msg1	16
3.19	Значение msg2	16
3.20	Изменение символа в msg1	16
3.21	Изменение символа в msg2	16
3.22	Значение регистра edx в разных форматах	16
3.23	Изменение значения регистра ebx	17
3.24	Копирование lab9-2.asm в lab10-3.asm	17
3.25	Создание исполняемого файла	17
3.26	Загрузка в отладчик	18
3.27	Установка точки останова	18
3.28	Адрес вершины стека	18
3.29	Позиции стека	19
4.1	Преобразование программы	20
4.2	Проверка работы программы	21
4.3	Исправление ошибки	21
4.4	Проверка работы программы	21

Список таблиц

1 Цель работы

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

2 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, отработывает, 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 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 строки, 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.

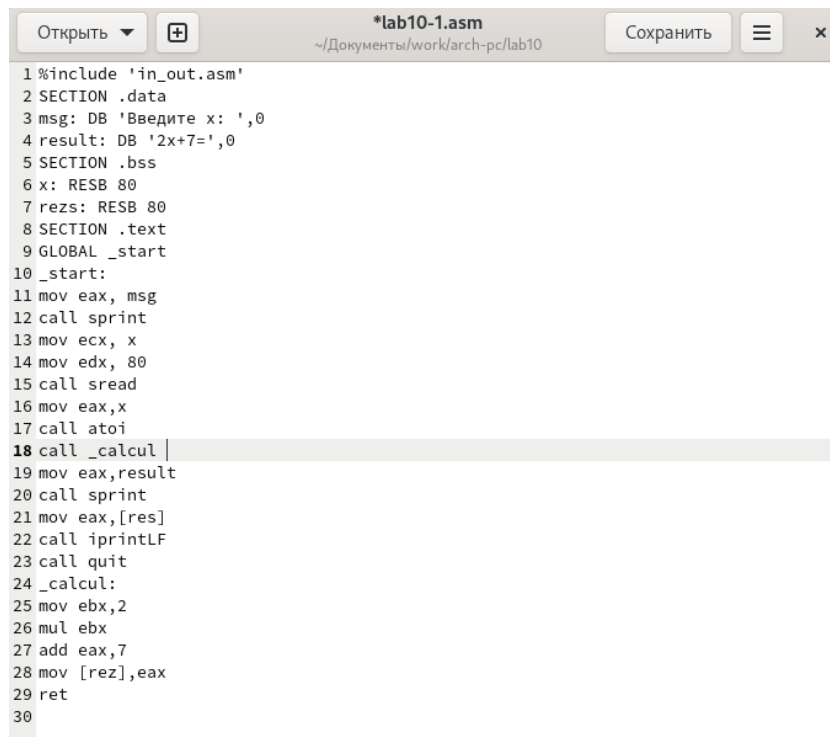
3 Выполнение лабораторной работы

1. Создала каталог для выполнения лабораторной работы No 10, перешла в него и создала файл lab10-1.asm.

```
[emmednikova@fedora ~]$ mc  
  
[emmednikova@fedora arch-pc]$ mkdir lab10  
[emmednikova@fedora arch-pc]$ cd lab10/  
[emmednikova@fedora lab10]$ touch lab10-1.asm  
[emmednikova@fedora lab10]$ ls  
lab10-1.asm  
[emmednikova@fedora lab10]$
```

Рис. 3.1: Создание каталога и файла

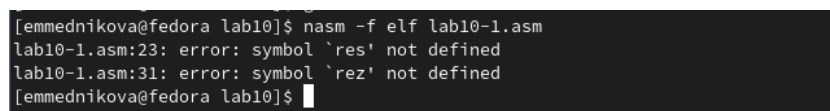
2. Ввела в файл lab10-1.asm текст программы из листинга 10.1.



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

Рис. 3.2: Ввод текста программы в файл

Создала исполняемый файл и проверила его работу. В результате программа не заработала, так как не распознаёт 'res' и 'rez'.



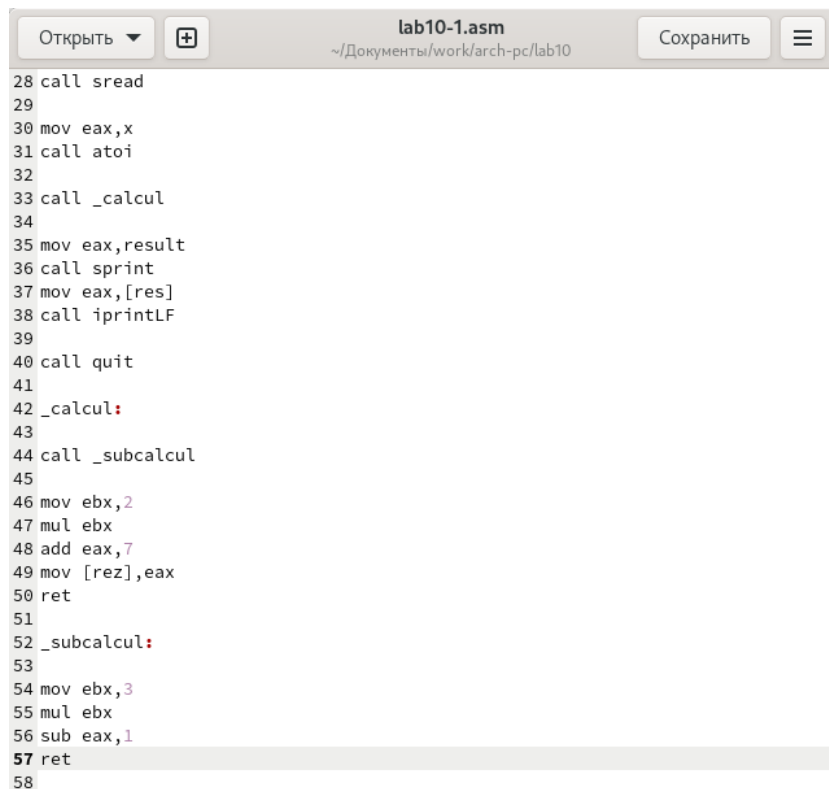
```
[emmednikova@fedora lab10]$ nasm -f elf lab10-1.asm
lab10-1.asm:23: error: symbol `res' not defined
lab10-1.asm:31: error: symbol `rez' not defined
[emmednikova@fedora lab10]$
```

Рис. 3.3: Создание и проверка работы файла

Изменила текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`.

```
1 %include 'in_out.asm'
2
3 SECTION .data
4 msg: DB 'Введите x: ',0
5 prim1: DB 'f(x) = 2x+7',0
6 prim2: DB 'g(x) = 3x-1',0
7 result: DB 'f(g(x))= ',0
8
9 SECTION .bss
10 x: RESB 80
11 rezs: RESB 80
12
13 SECTION .text
14 GLOBAL _start
15 _start:
16
17 mov eax, prim1
18 call sprintf
19
20 mov eax, prim2
21 call sprintf
22
23 mov ecx,msg
24 call sprintf
25
26 mov ecx, x
27 mov edx, 80
28 call sread
29
30 mov eax,x
31 call atoi
```

Рис. 3.4: Изменение текста программы

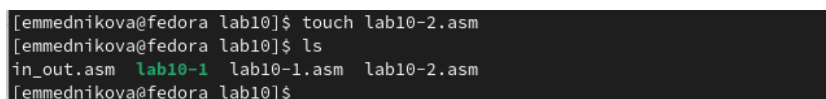


```
28 call sread
29
30 mov eax,x
31 call atoi
32
33 call _calcul
34
35 mov eax,result
36 call sprint
37 mov eax,[res]
38 call iprintLF
39
40 call quit
41
42 _calcul:
43
44 call _subcalcul
45
46 mov ebx,2
47 mul ebx
48 add eax,7
49 mov [rez],eax
50 ret
51
52 _subcalcul:
53
54 mov ebx,3
55 mul ebx
56 sub eax,1
57 ret
58
```

Рис. 3.5: Изменение текста программы

При попытке создать исполняемый файл и проверить его работу, выдавалась та же самая ошибка, что и на рис. 3.3.

3. Создала файл lab10-2.asm с текстом программы из Листинга 10.2. (Программа печати сообщения Hello world!)



```
[emmednikova@fedora lab10]$ touch lab10-2.asm
[emmednikova@fedora lab10]$ ls
in_out.asm  lab10-1  lab10-1.asm  lab10-2.asm
[emmednikova@fedora lab10]$
```

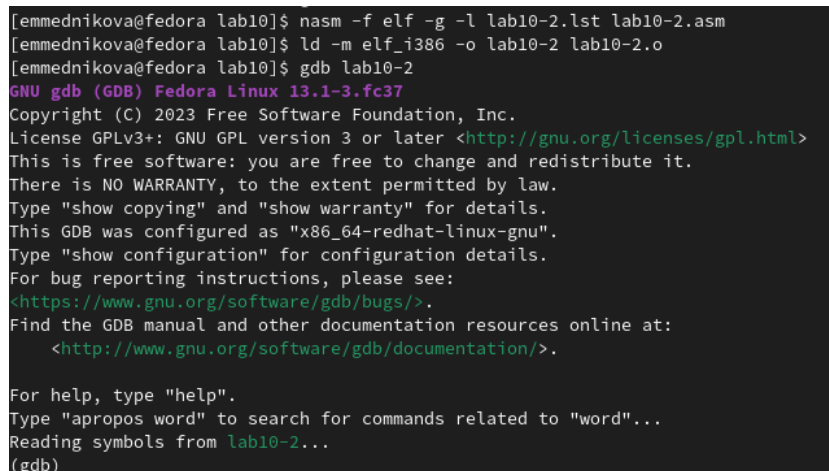
Рис. 3.6: Создание файла



```
1 SECTION .data
2 msg1: db "Hello, ",0x0
3 msg1Len: equ $ - msg1
4 msg2: db "world!",0xa
5 msg2Len: equ $ - msg2
6 SECTION .text
7 global _start
8 _start:
9 mov eax, 4
10 mov ebx, 1
11 mov ecx, msg1
12 mov edx, msg1Len
13 int 0x80
14 mov eax, 4
15 mov ebx, 1
16 mov ecx, msg2
17 mov edx, msg2Len
18 int 0x80
19 mov eax, 1
20 mov ebx, 0
21 int 0x80
```

Рис. 3.7: Ввод текста программы

Получила исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом '-g'. Загрузила исполняемый файл в отладчик gdb.



```
[emmednikova@fedora lab10]$ nasm -f elf -g -l lab10-2.lst lab10-2.asm
[emmednikova@fedora lab10]$ ld -m elf_i386 -o lab10-2 lab10-2.o
[emmednikova@fedora lab10]$ gdb lab10-2
GNU gdb (GDB) Fedora Linux 13.1-3.fc37
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"...
Reading symbols from lab10-2...
(gdb)
```

Рис. 3.8: Создание исполняемого файла и загрузка в отладчик

Проверила работу программы, запустив ее в оболочке GDB с помощью команды run.

```
(gdb) run
Starting program: /home/emmednikova/Документы/work/arch-pc/lab10/lab10-2

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 4489) exited normally]
(gdb)
```

Рис. 3.9: Проверка работы программы

Для более подробного анализа программы установила брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустила её.

```
(gdb) break _start
Breakpoint 1 at 0x08049000: file lab10-2.asm, line 9.
(gdb) run
Starting program: /home/emmednikova/Документы/work/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:9
9      mov eax, 4
(gdb)
```

Рис. 3.10: Анализ программы

Посмотрела 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. 3.11: `disassemble`

Переключилась 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. 3.12: `set disassembly-flavor intel`

Различия отображения синтаксиса машинных команд в режимах АТТ и Intel: В коде Intel отсутствуют суффиксы обозначения размера; опускается символ `%` перед именами регистров; имеет другой способ описания местоположений в памяти.

Включила режим псевдографики для более удобного анализа программы.

```
[ Register Values Unavailable ]

B+> 0x08049000 <_start>  mov     eax,0x4
      0x08049005 <_start+5>  mov     ebx,0x1
      0x0804900a <_start+10> mov     ecx,0x804a000
      0x0804900f <_start+15> mov     edx,0x8
      0x08049014 <_start+20> int     0x80
      0x08049016 <_start+22> mov     eax,0x4

native process 4557 In: _start          L9    PC: 0x08049000
(gdb) layout regs
(gdb) █
```

Рис. 3.13: Режим псевдографики

4. На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверила это с помощью команды `info breakpoints`.

```
(gdb) info breakpoints
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000 lab10-2.asm:9
breakpoint already hit 1 time
(gdb)
```

Рис. 3.14: Точка останова

Определила адрес предпоследней инструкции (`mov ebx,0x0`) и установила точку останова.

```
(gdb) b *0x8049000
Note: breakpoint 1 also set at pc 0x8049000.
Breakpoint 2 at 0x8049000: file lab10-2.asm, line 9.
(gdb)
```

Рис. 3.15: Установка точки останова

Посмотрела информацию о всех установленных точках останова.

```
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000 lab10-2.asm:9
breakpoint already hit 1 time
2        breakpoint     keep y   0x08049000 lab10-2.asm:9
(gdb)
```

Рис. 3.16: Информация о точках останова

5. Посмотрела содержимое регистров с помощью команды `info registers`.

```
native process 4557 In: _start                                L9    PC: 0x8049000
eax      0x0                                           0
ecx      0x0                                           0
edx      0x0                                           0
ebx      0x0                                           0
esp      0xfffffd180                                0xfffffd180
ebp      0x0                                           0x0
esi      0x0                                           0
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 3.17: Содержимое регистров

Посмотрела значение переменной `msg1` по имени.

```
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) █
```

Рис. 3.18: Значение msg1

Посмотрела значение переменной msg2 по адресу.

```
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb) █
```

Рис. 3.19: Значение msg2

Изменила первый символ переменной msg1.

```
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) █
```

Рис. 3.20: Изменение символа в msg1

Заменяла символ во второй переменной msg2.

```
(gdb) set {char}&msg2='k'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "korld!\n\034"
(gdb) █
```

Рис. 3.21: Изменение символа в msg2

Вывела в шестнадцатеричном, в двоичном формате и в символьном виде значение регистра edx.

```
(gdb) p/x $edx
$1 = 0x0
(gdb) p/t $edx
$2 = 0
(gdb) p/s $edx
$3 = 0
(gdb)
```

Рис. 3.22: Значение регистра edx в разных форматах

С помощью команды set изменила значение регистра ebx.

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb)
```

Рис. 3.23: Изменение значения регистра ebx

6. Скопировала файл lab9-2.asm, созданный при выполнении лабораторной работы No9, с программой, выводящей на экран аргументы командной строки (Листинг 9.2) в файл с именем lab10-3.asm.

Левая панель		Файл	Команда	Настройки	Правая панель		
<- ...ументы/work/arch-pc/lab10 -.[^]>				<- ...ументы/work/arch-pc/lab09 -.[^]>			
.и	Имя	Размер	Время правки	.и	Имя	Размер	Время правки
./..		-ВВЕРХ-	апр 30 20:13	./..		-ВВЕРХ-	апр 30 20:13
	in_out.asm	3942	апр 12 16:37		in_out.asm	3942	апр 12 16:37
*	lab10-1	9192	мая 1 12:31	*	lab9-1	9100	апр 30 19:11
	lab10-1.asm	566	мая 1 12:54		lab9-1.asm	334	апр 30 19:11
*	lab10-2	9112	мая 1 13:00		lab9-1.o	1424	апр 30 19:11
	lab10-2.asm	294	мая 1 12:59	*	lab9-2	5124	апр 30 19:14
	lab10-2.lst	1134	мая 1 12:59		lab9-2.asm	944	апр 30 19:13
	lab10-2.o	1616	мая 1 12:59		lab9-2.o	1088	апр 30 19:13
	lab10-3.asm	944	апр 30 19:13	*	lab9-3	9052	апр 30 19:24
					lab9-3.asm	332	апр 30 19:23
					lab9-3.o	1312	апр 30 19:24
				*	sm	9080	апр 30 19:35
					sm.asm	356	апр 30 19:35

Рис. 3.24: Копирование lab9-2.asm в lab10-3.asm

Создала исполняемый файл.

```
[emmednikova@fedora lab10]$ nasm -f elf -g -l lab10-3.lst lab10-3.asm
[emmednikova@fedora lab10]$ ld -m elf_i386 -o lab10-3 lab10-3.o
```

Рис. 3.25: Создание исполняемого файла

Загрузила исполняемый файл в отладчик, указав аргументы.

```
[emmednikova@fedora lab10]$ gdb --args lab10-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) Fedora Linux 13.1-3.fc37
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"...
Reading symbols from lab10-3...
(gdb)
```

Рис. 3.26: Загрузка в отладчик

Установила точку останова перед первой инструкцией в программе и запустила ее.

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab10-3.asm, line 5.
(gdb) run
Starting program: /home/emmednikova/Документы/work/arch-pc/lab10/lab10-3 аргумент1 аргумент 2 аргумент\ 3

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.fedoraproject.org/>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.

Breakpoint 1, _start () at lab10-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb)
```

Рис. 3.27: Установка точки останова

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

```
(gdb) x/x $esp
0xffffd130: 0x00000005
(gdb)
```

Рис. 3.28: Адрес вершины стека

Посмотрела остальные позиции стека – по адресу [esp+4] располагается адрес в

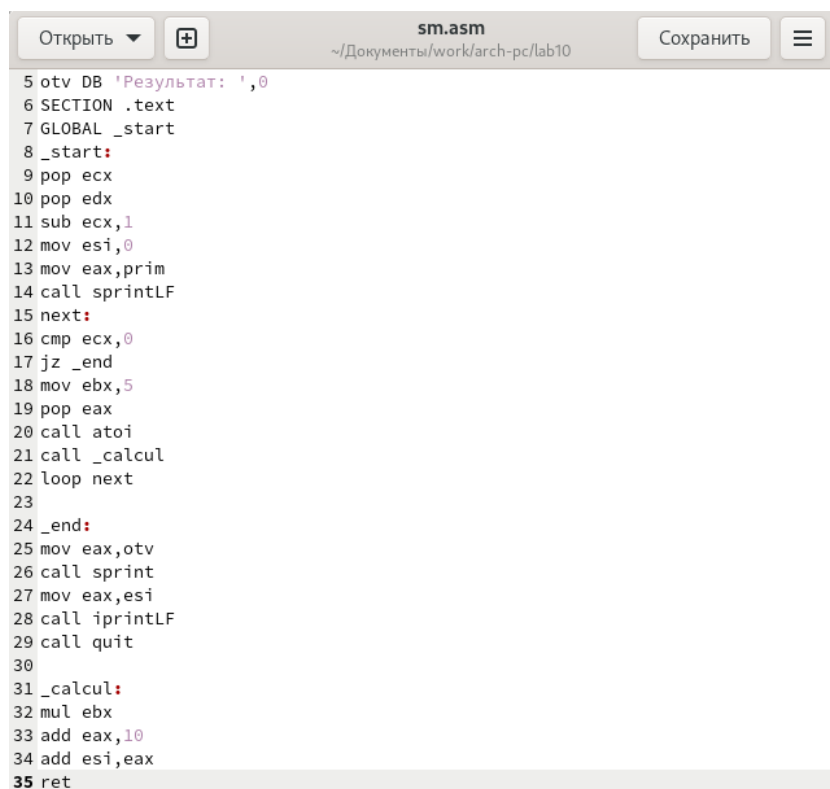
памяти, где находится имя программы, по адресу [esp+8] хранится адрес первого аргумента, по адресу [esp+12] – второго и т.д.

```
(gdb) x/s *(void*)($esp + 4)
0xffffd2d8:  "/home/emmednikova/Документы/work/arch-pc/lab10/lab10-3"
(gdb) x/s *(void*)($esp + 8)
0xffffd318:  "аргумент1"
(gdb) x/s *(void*)($esp + 12)
0xffffd32a:  "аргумент"
(gdb) x/s *(void*)($esp + 16)
0xffffd33b:  "2"
(gdb) x/s *(void*)($esp + 20)
0xffffd33d:  "аргумент 3"
(gdb) x/s *(void*)($esp + 24)
0x0:  <error: Cannot access memory at address 0x0>
(gdb) █
```

Рис. 3.29: Позиции стека

4 Самостоятельная работа

1. Преобразовала программу из лабораторной работы No9 (Задание No1 для самостоятельной работы), реализовав вычисление значения функции как подпрограмму.



```
5 otv DB 'Результат: ',0
6 SECTION .text
7 GLOBAL _start
8 _start:
9 pop ecx
10 pop edx
11 sub ecx,1
12 mov esi,0
13 mov eax,prim
14 call sprintLF
15 next:
16 cmp ecx,0
17 jz _end
18 mov ebx,5
19 pop eax
20 call atoi
21 call _calcul
22 loop next
23
24 _end:
25 mov eax,otv
26 call sprint
27 mov eax,esi
28 call iprintLF
29 call quit
30
31 _calcul:
32 mul ebx
33 add eax,10
34 add esi,eax
35 ret
```

Рис. 4.1: Преобразование программы


```

[emmednikova@fedora lab10]$ gedit sm.asm
[emmednikova@fedora lab10]$ nasm -f elf sm.asm
[emmednikova@fedora lab10]$ ld -m elf_i386 -o sm sm.o
[emmednikova@fedora lab10]$ ./sm 1 2 3 4
f(x)=10+5x
Результат: 90
[emmednikova@fedora lab10]$

```

Рис. 4.2: Проверка работы программы

2. С помощью отладчика GDB, анализируя изменения значений регистров, определила ошибку программы и исправила ее.



```

1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ---- Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add eax,ebx
11 mov ecx,4
12 mul ecx
13 add eax,5
14 mov edi,eax
15 ; ---- Вывод результата на экран
16 mov eax,div
17 call sprint
18 mov eax,edi
19 call iprintLF
20 call quit

```

Рис. 4.3: Исправление ошибки

```

[emmednikova@fedora lab10]$ gedit sm2.asm
[emmednikova@fedora lab10]$ nasm -f elf sm2.asm
[emmednikova@fedora lab10]$ ld -m elf_i386 -o sm2 sm2.o
[emmednikova@fedora lab10]$ ./sm2
Результат: 25
[emmednikova@fedora lab10]$

```

Рис. 4.4: Проверка работы программы

5 Выводы

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