

Шаблон отчёта по лабораторной работе

№10

Сильвен Макс Грегор Филс , НКАбд-03-22

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы :	6
2.1	Реализация циклов в NASM :	6
2.2	Отладка программ с помощью GDB :	10
2.3	Добавление точек останова :	17
2.4	Работа с данными программы в GDB :	18
2.5	Обработка аргументов командной строки в GDB :	23
2.6	Выводы по результатам выполнения заданий :	25
3	Задание для самостоятельной работы :	26
3.1	Выводы по результатам выполнения заданий :	28
4	Выводы, согласованные с целью работы :	29
	Список литературы	30

Список иллюстраций

2.1	Рисунок	6
2.2	Рисунок	7
2.3	Рисунок	8
2.4	Рисунок	9
2.5	Рисунок	10
2.6	Рисунок	11
2.7	Рисунок	12
2.8	Рисунок	13
2.9	Рисунок	13
2.10	Рисунок	14
2.11	Рисунок	15
2.12	Рисунок	16
2.13	Рисунок	17
2.14	Рисунок	17
2.15	Рисунок	18
2.16	Рисунок	19
2.17	Рисунок	20
2.18	Рисунок	20
2.19	Рисунок	21
2.20	Рисунок	21
2.21	Рисунок	22
2.22	Рисунок	23
2.23	Рисунок	24
2.24	Рисунок	24
3.1	Файл lab10-4.asm	26
3.2	Работа программы lab10-4.asm	26
3.3	код с ошибкой	27
3.4	отладка	27
3.5	код исправлен	28

Список таблиц

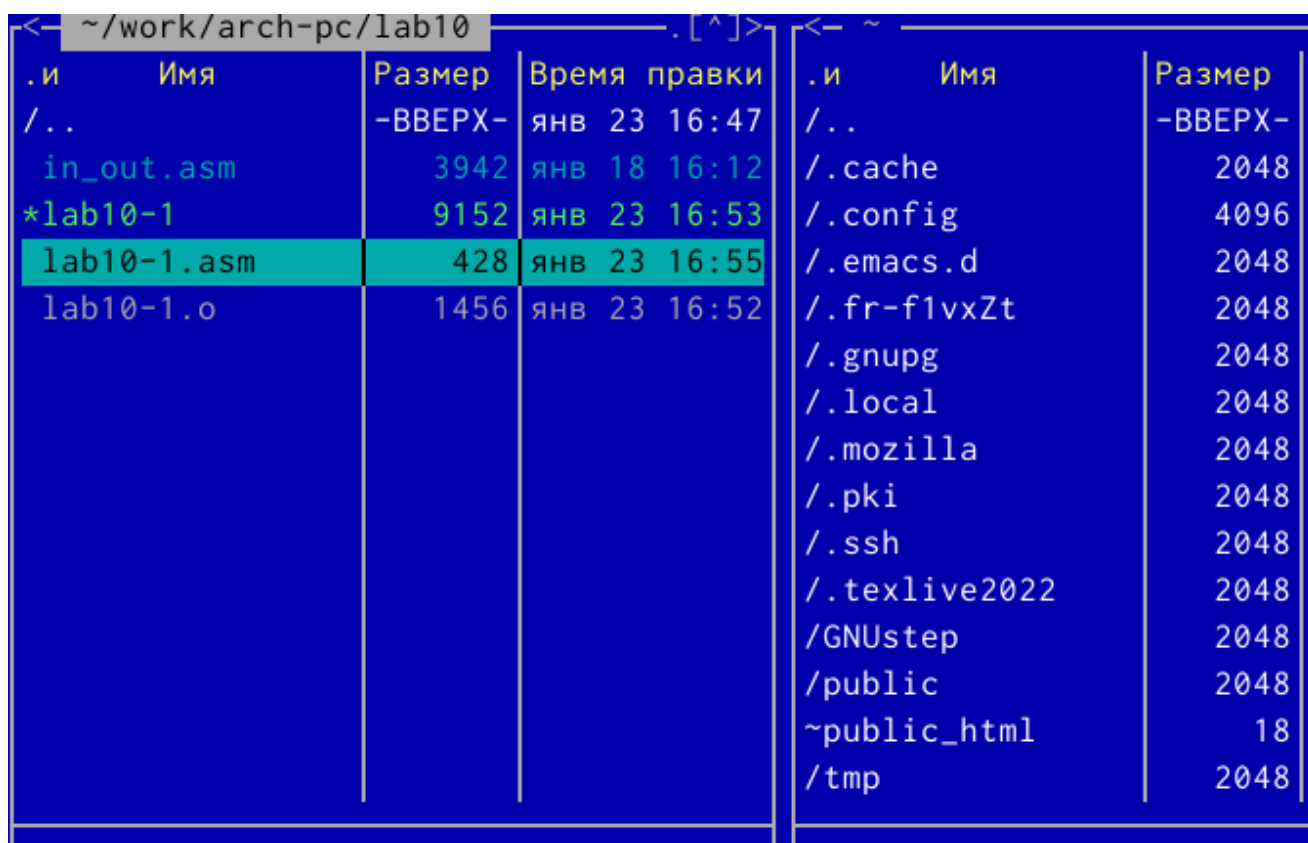
1 Цель работы

- В этой лабораторной работе мы научимся писать программы с использованием подпрограмм и познакомимся со способами отладки с использованием GDB и его основными функциями

2 Выполнение лабораторной работы :

2.1 Реализация циклов в NASM :

- Здесь мы начали с создания каталога для программы лабораторной работы No10, а затем переместились в десятый каталог лаборатории “~/work/arch-pc/lab10”, после чего мы создали файл “lab10-1.asm”. (рис. 2.1)

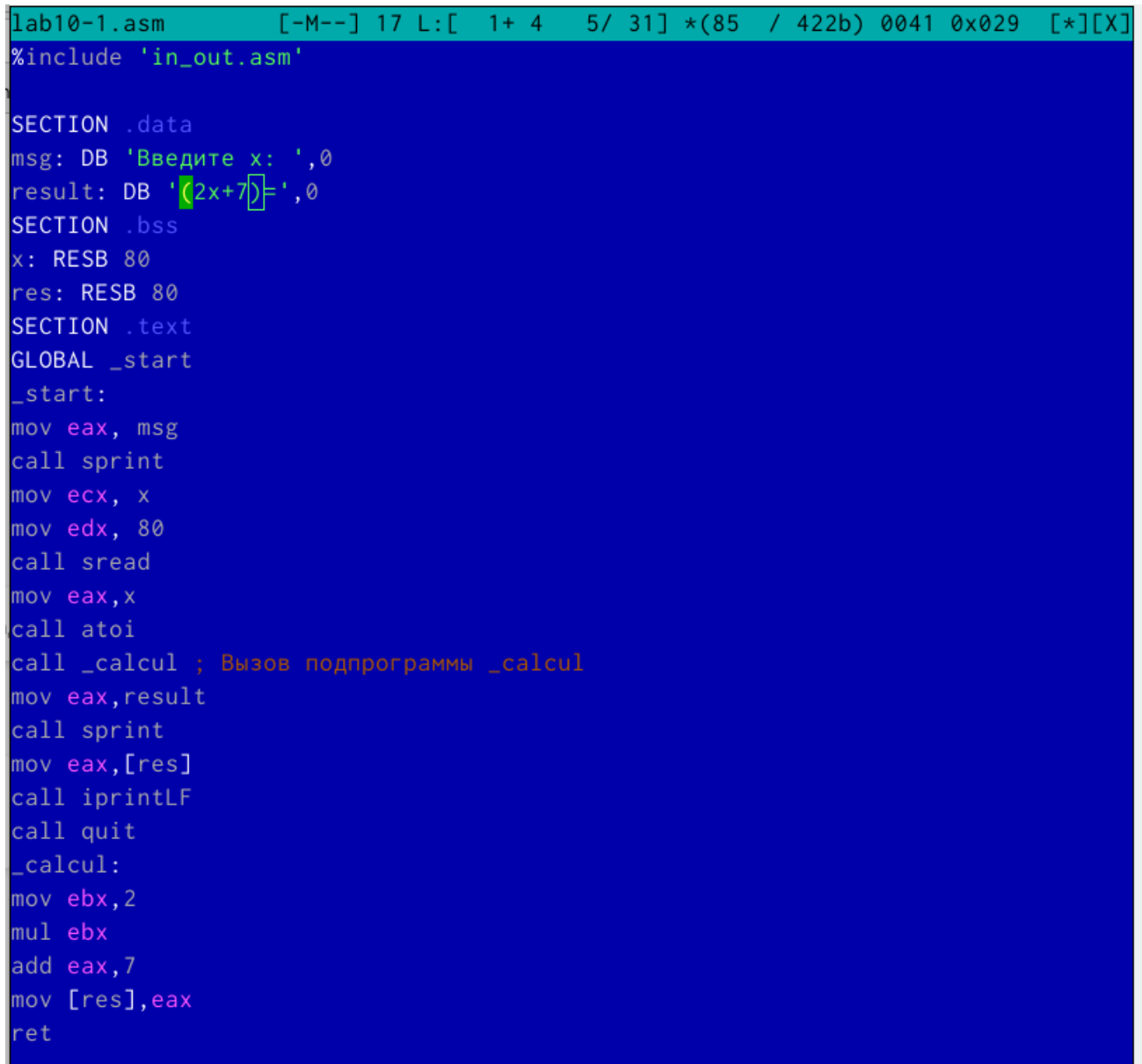


и	Имя	Размер	Время правки
/..	-ВВЕРХ-		янв 23 16:47
	in_out.asm	3942	янв 18 16:12
	*lab10-1	9152	янв 23 16:53
	lab10-1.asm	428	янв 23 16:55
	lab10-1.o	1456	янв 23 16:52

и	Имя	Размер	Время правки
/..	-ВВЕРХ-		янв 23 16:47
/.cache		2048	янв 23 16:47
/.config		4096	янв 23 16:47
/.emacs.d		2048	янв 23 16:47
/.fr-f1vxZt		2048	янв 23 16:47
/.gnupg		2048	янв 23 16:47
/.local		2048	янв 23 16:47
/.mozilla		2048	янв 23 16:47
/.pki		2048	янв 23 16:47
/.ssh		2048	янв 23 16:47
/.texlive2022		2048	янв 23 16:47
/GNUstep		2048	янв 23 16:47
/public		2048	янв 23 16:47
~public_html		18	янв 23 16:47
/tmp		2048	янв 23 16:47

Рис. 2.1: Ресунок

- Затем мы заполнили код нашей программы в файле lab10-1.asm.(рис. 2.2)



```
lab10-1.asm      [-M--] 17 L:[ 1+ 4 5/ 31] *(85 / 422b) 0041 0x029 [*][X]
#include 'in_out.asm'

SECTION .data
msg: DB 'Введите x: ',0
result: DB '([2x+7])=',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 ; Вызов подпрограммы _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. 2.2: Рисунок

- После этого мы скомпилировали файл, создали исполняемый файл и проверили его работу.(рис. 2.3)

```
fsmaksgregor@dk6n50:~/work/... x mc [fsmaksgregor@dk6n50]:~/... x fsmaksgregor@dk6n50:~/work/... x
fsmaksgregor@dk6n50 ~/work/arch-pc/lab10 $ nasm -f elf lab10-1.asm
fsmaksgregor@dk6n50 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-1 lab10-1.o
fsmaksgregor@dk6n50 ~/work/arch-pc/lab10 $ ./lab10-1
Введите x: 1
2x+7=9
fsmaksgregor@dk6n50 ~/work/arch-pc/lab10 $
```

Рис. 2.3: Рисунок

- Мы внесли изменения в наш код, чтобы она вычислила это уравнение $f(g(x))$, где x вводится с клавиатуры и $f(x) = 2x + 7$, $g(x) = 3x - 1$ а затем создали исполняемый файл.(рис. 2.4)(рис. 2.5)


```
mc [fsmaksgregor@dk5n55]:~/work/arch-pc/lab10  fsmaksgregor@dk5n55:~/work/arch-pc/lab10
lab10-1.asm  [----]  3 L:[ 1+ 5  6/ 50] *(100 / 529b) 0084 0x054  [*][X]
#include 'in_out.asm'

SECTION .data
msg: DB 'Введите x: ',0
result: DB 'f(x)=3(2x+7)=',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 ; Вызов подпрограммы _calcul

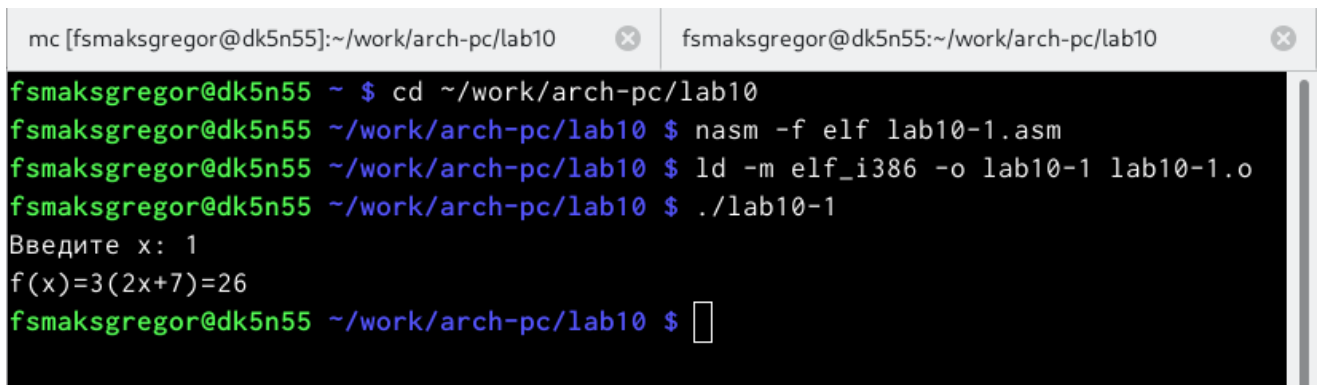
mov eax, [res]

call _subcalcul

mov eax, result
call sprint
mov eax, [res]
call iprintLF

call quit
```

Рис. 2.4: Ресунок

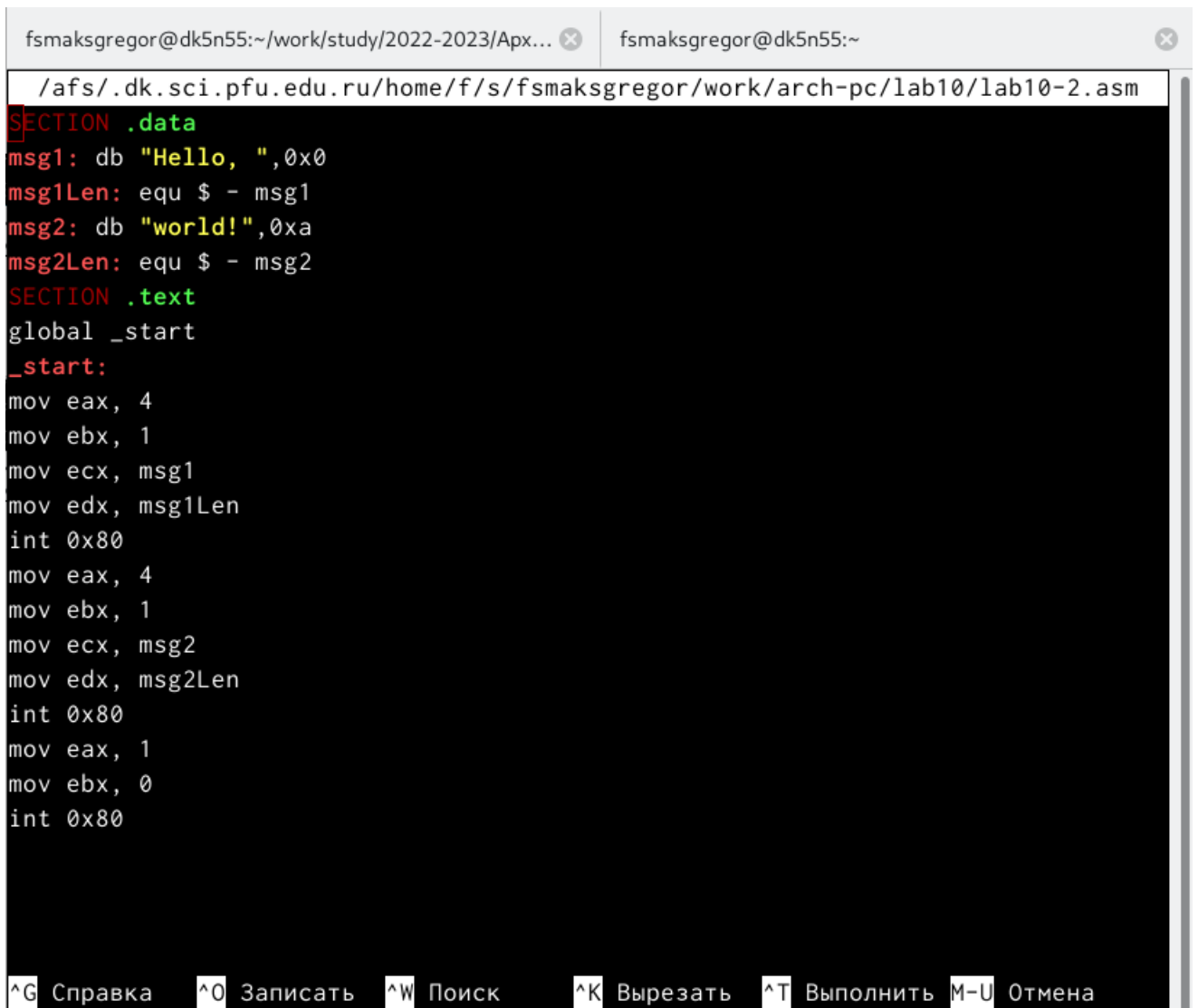


```
mc [fsmaksgregor@dk5n55]:~/work/arch-pc/lab10 x fsmaksgregor@dk5n55:~/work/arch-pc/lab10 x
fsmaksgregor@dk5n55 ~ $ cd ~/work/arch-pc/lab10
fsmaksgregor@dk5n55 ~/work/arch-pc/lab10 $ nasm -f elf lab10-1.asm
fsmaksgregor@dk5n55 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-1 lab10-1.o
fsmaksgregor@dk5n55 ~/work/arch-pc/lab10 $ ./lab10-1
Введите x: 1
f(x)=3(2x+7)=26
fsmaksgregor@dk5n55 ~/work/arch-pc/lab10 $
```

Рис. 2.5: Ресунок

2.2 Отладка программ с помощью GDB :

- На этом шаге мы создали файл lab10-2.asm с текстом программы из листинга 10.2.(рис. 2.6)



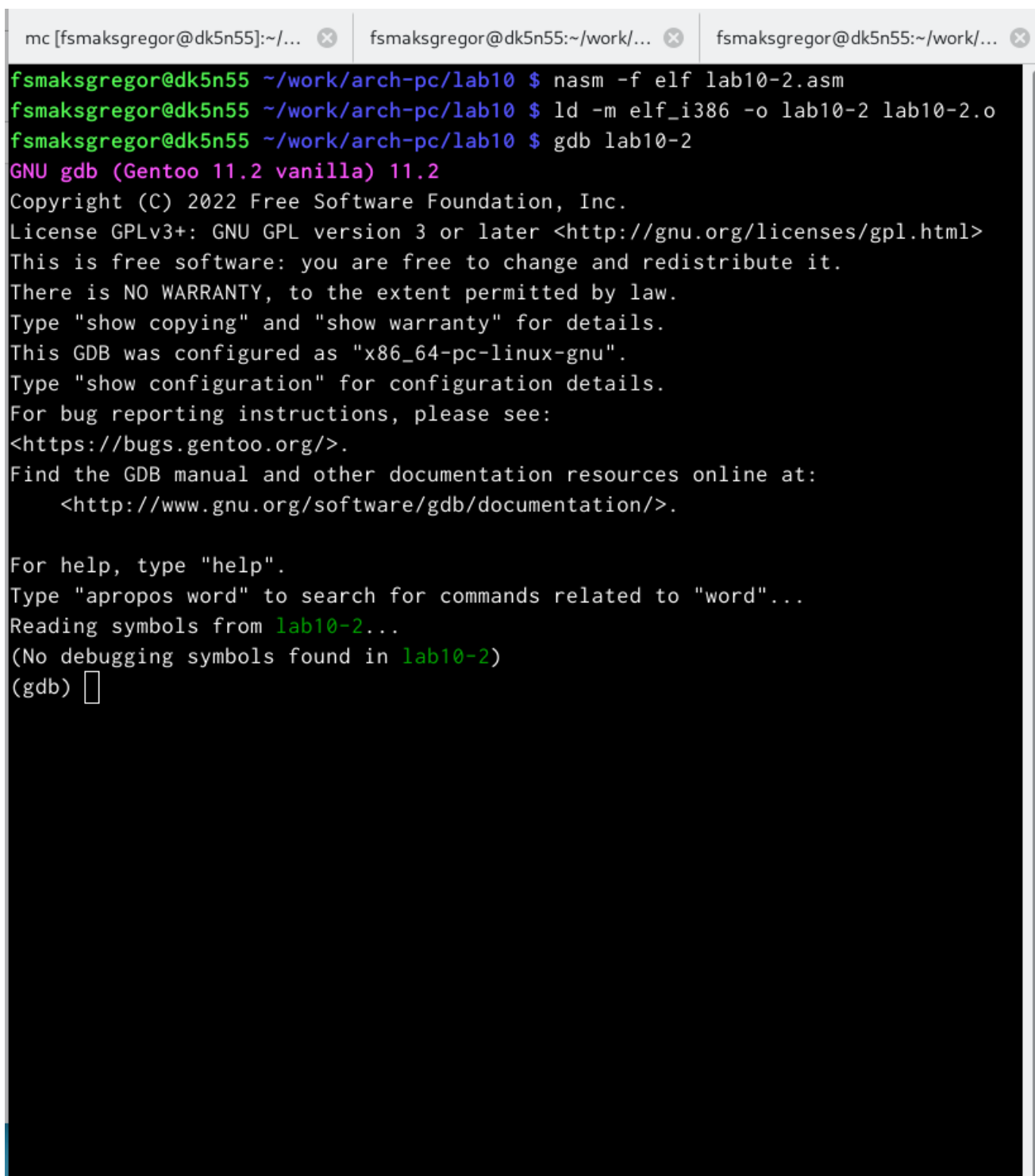
The screenshot shows a terminal window with two tabs. The active tab is titled 'fsmaksgregor@dk5n55:~/work/study/2022-2023/Apx...'. The terminal displays the following assembly code:

```
/afs/.dk.sci.pfu.edu.ru/home/f/s/fsmaksgregor/work/arch-pc/lab10/lab10-2.asm
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
```

At the bottom of the terminal window, there is a menu bar with the following items: ^G Справка, ^O Записать, ^W Поиск, ^K Вырезать, ^T Выполнить, M-U Отмена.

Рис. 2.6: Рисунок

- После этого мы скомпилировали файл, создали исполняемый файл.Затем мы загрузили исполняемый файл в отладчик GDM. (рис. 2.7)

A terminal window with three tabs. The first tab is titled 'mc [fsmaksgregor@dk5n55]:~/...' and is inactive. The second and third tabs are titled 'fsmaksgregor@dk5n55:~/work/...' and are active. The terminal content shows the user compiling a program with nasm and ld, then launching it in GDB. GDB displays its version (11.2), copyright, license (GPLv3+), and configuration details. It then attempts to load symbols from 'lab10-2' but reports that no debugging symbols were found. The prompt '(gdb) ' is shown at the bottom.

```
mc [fsmaksgregor@dk5n55]:~/... x fsmaksgregor@dk5n55:~/work/... x fsmaksgregor@dk5n55:~/work/... x
fsmaksgregor@dk5n55 ~/work/arch-pc/lab10 $ nasm -f elf lab10-2.asm
fsmaksgregor@dk5n55 ~/work/arch-pc/lab10 $ ld -m elf_i386 -o lab10-2 lab10-2.o
fsmaksgregor@dk5n55 ~/work/arch-pc/lab10 $ gdb lab10-2
GNU gdb (Gentoo 11.2 vanilla) 11.2
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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...
(No debugging symbols found in lab10-2)
(gdb) 
```

Рис. 2.7: Ресунок

- затем мы проверили работу программы, запустив ее в оболочке GDB с помощью команды run.(рис. 2.8)

```

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab10-2...
(No debugging symbols found in lab10-2)
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/f/s/fsmaksgregor/work/arch-pc/lab
10/lab10-2
Hello, world!
[Inferior 1 (process 4400) exited normally]
(gdb) 

```

Рис. 2.8: Ресунок

- затем мы установили точку останова на метке `**_start**`, которая запускает выполнение любой программы на ассемблере, и запустили ее.(рис. 2.9)

<- ~/work/arch-pc/lab10 .[^]>				<- ~			
.и	Имя	Размер	Время правки	.и	Имя	Размер	В
/..		-ВВЕРХ-	январь 23 16:47	/..		-ВВЕРХ-	с
	in_out.asm	3942	январь 18 16:12	/.cache		2048	с
*lab10-1		9152	январь 23 16:53	/.config		4096	с
	lab10-1.asm	428	январь 23 16:55	/.emacs.d		2048	с
	lab10-1.o	1456	январь 23 16:52	/.fr-f1vxZt		2048	с
				/.gnupg		2048	с
				/.local		2048	с
				/.mozilla		2048	с
				/.pki		2048	с
				/.ssh		2048	с
				/.texlive2022		2048	с
				/GNUstep		2048	с
				/public		2048	с
				~public_html		18	с
				/tmp		2048	с

Рис. 2.9: Ресунок

- Затем мы просмотрели разобранный программный код, используя команду `disassemble`, начинающуюся с метки `**_start**`. (рис. 2.10)

```
(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.
```

Рис. 2.10: Ресунок

- после этого мы переключились на отображение команд с синтаксисом Intel, введя команду `set disassembly-flavor intel`. (рис. 2.11)

```

(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. 2.11: Ресунок

- Разница в синтаксисе между AT&T и INTEL заключается в том, что AT&T использует синтаксис `mov $0x4,%eax`, который популярен среди пользователей Linux, с другой стороны, INTEL использует синтаксис `mov eax,0x4`, который является популярен среди пользователей Windows.
- Затем мы включили псевдографический режим для более удобного анализа программы. (рис. 2.12)

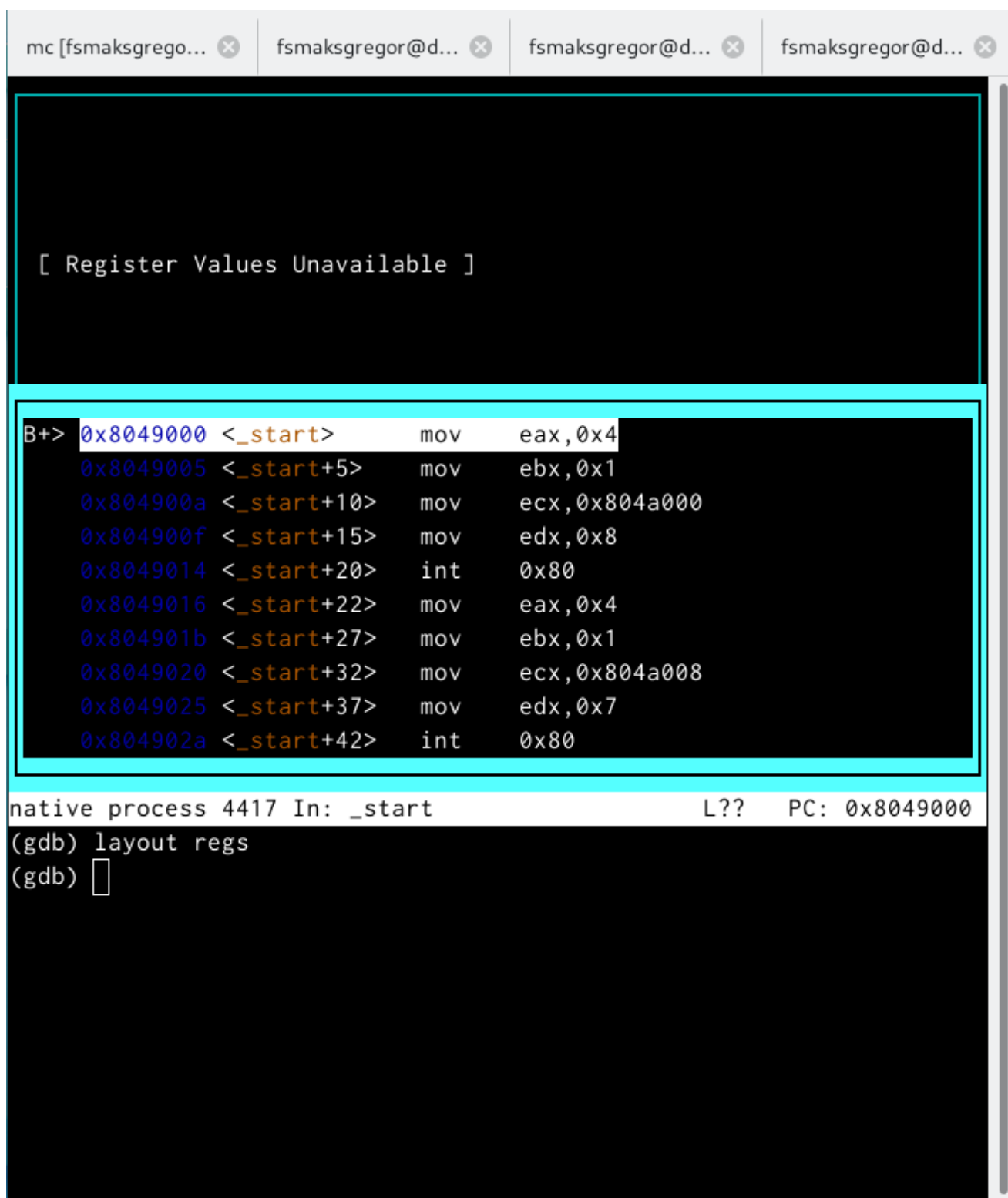


Рис. 2.12: Ресунок

2.3 Добавление точек останова :

- Мы проверили точку останова с помощью информационных точек останова.(рис. 2.13)

```
native process 4417 In: _start L?? PC: 0x8049000
(gdb) layout regs
(gdb) info breakpoints
Num      Type          Disp Enb Address      What
1        breakpoint    keep y  0x08049000 <_start>
          breakpoint already hit 1 time
(gdb) █
```

Рис. 2.13: Ресунок

- Мы определили адрес предпоследней инструкции (mov ebx,0x0) и установили точку останова.(рис. 2.14)

```
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031
(gdb) i b
Num      Type          Disp Enb Address      What
1        breakpoint    keep y  0x08049000 <_start>
          breakpoint already hit 1 time
2        breakpoint    keep y  0x08049031 <_start+49>
(gdb) █
```

Рис. 2.14: Ресунок

2.4 Работа с данными программы в GDB :

- На этом шаге мы следовали 5 инструкциям, используя командный шаг `i`, и отслеживали изменение значений регистров, но перед этим мы проверили предыдущие значения регистров.(рис. 2.15)(рис. ??)

```
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc5a0 0xffffc5a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049000 0x8049000 <_start>
eflags   0x202    [ IF ]
```

Рис. 2.15: Ресунок

```
(gdb) stepi
0x08049005 in _start ()
(gdb) stepi
0x0804900a in _start ()
(gdb) stepi
0x0804900f in _start ()
(gdb) stepi
0x08049014 in _start ()
(gdb) stepi
0x08049016 in _start ()
(gdb) 
```

```

eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc5a0 0xffffc5a0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--

```

- После проверки мы видим, что регистры : eax,ecx,edx,ebx,esp изменили свое значение.
- Мы рассмотрели значение переменной msg1 по имени, используя команду x/1sb.(рис. 2.16)

```

(gdb) x/1sb &msg1
0x804a000: "Hello, "
(gdb) 

```

Рис. 2.16: Ресунок

- Здесь мы рассмотрели значение переменной msg2, используя адрес.(рис. 2.17)

```
(gdb) x/1sb 0x804a008
0x804a008: "world!\n"
(gdb) 
```

Рис. 2.17: Рисунок

- Здесь мы изменили первую букву переменной msg1, которая имеет тип char.(рис. 2.18)

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

Рис. 2.18: Рисунок

- После этого мы изменили первую букву переменной msg2.(рис. 2.19)

```

(gdb) set {char}&msg2='F'
(gdb) x/1sb &msg2
0x804a008:      "For1d!\n"
(gdb) 

```

Рис. 2.19: Ресунок

- Затем мы выводим значение регистра `edx` в различных форматах (шестнадцатеричном, двоичном и символьном). (рис. 2.20)

```

(gdb) p/x $edx
$1 = 0x8
(gdb) p/s $edx
$2 = 8
(gdb) p/t $edx
$3 = 1000
(gdb) p/s $edx
$4 = 8
(gdb) 

```

Рис. 2.20: Ресунок

- Используя команду `set`, мы изменили значение регистра `ebx`, когда раз, введя '2', а в другой раз, введя 2. (рис. 2.21)

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$5 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$6 = 2
(gdb) 
```

Рис. 2.21: Рисунок

- но когда мы напечатали значение регистра, мы получили значение 50 и это потому, что машина интерпретировала введенное значение как символ, и в таблице ASCII символ '2' имеет значение 50 в десятичной системе, но когда мы ввели значение 2 машина интерпретировала 2 как число в десятичной системе.
- Наконец, мы завершили программу с помощью `stepi` и вышли из GDB с помощью команды `quit`. (рис. 2.22)

```
$4 = 8 No process in: L?? PC: ??  
A debugging session is active.  
  
Inferior 1 [process 4417] will be killed.  
  
Quit anyway? (y or n) nNot confirmed.  
A debugging session is active.  
  
Inferior 1 [process 4417] will be killed.  
  
Quit anyway? (y or n) nNot confirmed.  
(gdb) stepi  
[Inferior 1 (process 4417) exited normally]  
(gdb) ☐
```

Рис. 2.22: Ресунок

2.5 Обработка аргументов командной строки в GDB :

- На этом этапе мы скопировали файл lab9-2.asm, созданный при выполнении лабораторной работы No9 с программой, отображающей аргументы командной строки на экране (листинг 9.2), в файл с именем lab 10-3.asm, а затем мы скомпилировали этот файл и установил точку останова в ****_start**** и запустил отладчик.(рис. 2.23)

```
Файл  Правка  Вид  Терминал  Вкладки  Справка
fsmaksgregor@dk5n55 ~ $ cd work/arch-pc/lab10
fsmaksgregor@dk5n55 ~/work/arch-pc/lab10 $ cp ~/work/arch-pc/lab09/lab9-2.asm ~/work/arch-pc/lab10/lab10-3.asm
fsmaksgregor@dk5n55 ~/work/arch-pc/lab10 $ gdb --args lab10-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Gentoo 11.2 vanilla) 11.2
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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...
(No debugging symbols found in lab10-3)
(gdb) b _start
Breakpoint 1 at 0x80490e8
(gdb) □
```

Рис. 2.23: Ресунок

- Затем мы посмотрели на остальные позиции стека – адрес в памяти, где находится имя программы, находится в [esp + 4], адрес первого аргумента хранится в [esp + 8], в [esp + 12].(рис. 2.24)

```
Starting program: /afs/.dk.sci.pfu.edu.ru/home/f/s/fsmaksgregor/work/arch-pc/lab10/lab10-3 аргумент1 аргумент 2 аргумент\ 3
Breakpoint 1, 0x080490e8 in _start ()
(gdb) x/x $esp
0xfffffc520: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xfffffc79b: "/afs/.dk.sci.pfu.edu.ru/home/f/s/fsmaksgregor/work/arch-pc/lab10/lab10-3"
(gdb) x/s *(void**)(esp + 8)
0xfffffc7e4: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xfffffc7f6: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xfffffc807: "2"
(gdb) x/s *(void**)(esp + 20)
0xfffffc809: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb) □
```

Рис. 2.24: Ресунок

- Шаг изменения адреса равен 4, потому что размер регистра esp равен 32би-

там = 4 байтам, а количество памяти равно количеству аргументов плюс имя программы, поэтому мы получили 5 шагов с 4 байтами для каждого шага.

2.6 Выводы по результатам выполнения заданий :

- В этой части работы мы узнали, как работать с отладчиком GDB, и получили более близкое представление о том, как работают подпрограммы.

3 Задание для самостоятельной работы :

- Преобразуйте программу из лабораторной работы №9 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму. (рис. 3.1 3.2)

```
(gdb) stepi
0x08049005 in _start ()
(gdb) stepi
0x0804900a in _start ()
(gdb) stepi
0x0804900f in _start ()
(gdb) stepi
0x08049014 in _start ()
(gdb) stepi
0x08049016 in _start ()
(gdb) 
```

Рис. 3.1: Файл lab10-4.asm

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

Рис. 3.2: Работа программы lab10-4.asm

7. В листинге приведена программа вычисления выражения $(3+2)*4+5$. При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.(рис. 3.3 3.4 3.5 ??)

```
(gdb) x/1sb 0x804a008
0x804a008:      "world!\n"
(gdb) 
```

Рис. 3.3: код с ошибкой

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

Рис. 3.4: отладка

Отметим, что перепутан порядок аргументов у инструкции `add` и что по окончании работы в `edi` отправляется `ebx` вместо `eax`

```
(gdb) set {char}&msg2='F'
(gdb) x/1sb &msg2
0x804a008:      "For1d!\n"
(gdb) 
```

Рис. 3.5: код исправлен

3.1 Выводы по результатам выполнения заданий :

- В этой части мы узнали, как превратить программу в подпрограмму, но у нас возникла проблема с подпрограммой `atoi` , поэтому мы не смогли вычислить результат.

4 Выводы, согласованные с целью работы :

- В этой лабораторной работе мы научимся писать программы с использованием подпрограмм и познакомимся со способами отладки с использованием GDB и его основными функциями.

Список литературы