

Шаблон отчёта по лабораторной работе №10

Мугари Абдеррахим , 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 :	15
2.4	Работа с данными программы в GDB :	16
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 :	26
4	Выводы, согласованные с целью работы :	27

Список иллюстраций

2.1	Ресунок 1	6
2.2	Ресунок 2	7
2.3	Ресунок 3	8
2.4	Ресунок 4	9
2.5	Ресунок 5	10
2.6	Ресунок 6	11
2.7	Ресунок 7	12
2.8	Ресунок 8	13
2.9	Ресунок 9	13
2.10	Ресунок 10	14
2.11	Ресунок 11	14
2.12	Ресунок 12	15
2.13	Ресунок 13	16
2.14	Ресунок 14	16
2.15	Ресунок 15	17
2.16	Ресунок 16	18
2.17	Ресунок 17	18
2.18	Ресунок 18	19
2.19	Ресунок 19	19
2.20	Ресунок 20	19
2.21	Ресунок 21	20
2.22	Ресунок 22	21
2.23	Ресунок 23	22
2.24	Ресунок 24	23
2.25	Ресунок 25	24
2.26	Ресунок 26	24

Список таблиц

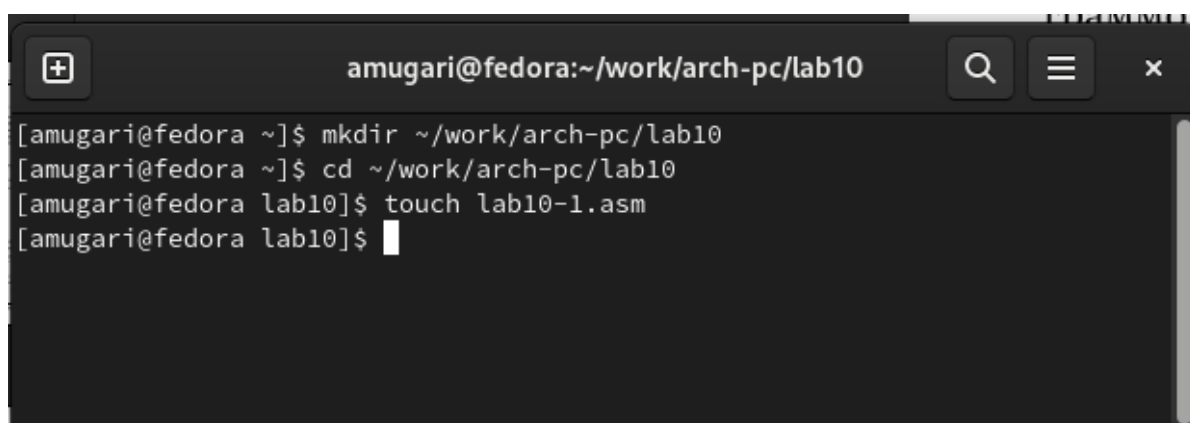
1 Цель работы :

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 :

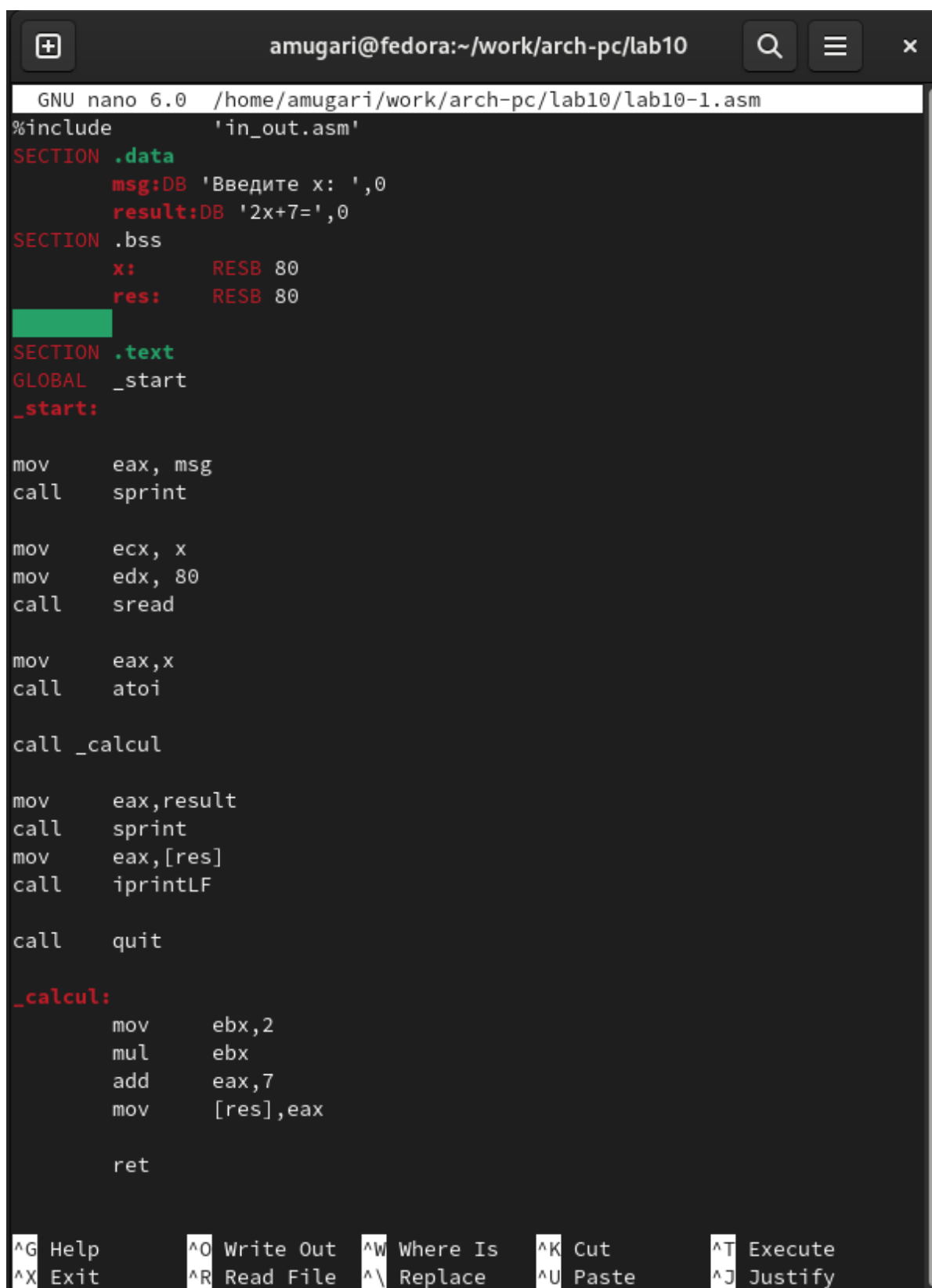
1. Здесь мы начали с создания каталога для программы лабораторной работы №10, а затем переместились в десятый каталог лаборатории “~/work/arch-pc/lab10”, после чего мы создали файл “**lab10-1.asm**”. (рис. 2.1)



```
amugari@fedora:~/work/arch-pc/lab10
[amugari@fedora ~]$ mkdir ~/work/arch-pc/lab10
[amugari@fedora ~]$ cd ~/work/arch-pc/lab10
[amugari@fedora lab10]$ touch lab10-1.asm
[amugari@fedora lab10]$
```

Рис. 2.1: Ресунок 1

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



```
GNU nano 6.0 /home/amugari/work/arch-pc/lab10/lab10-1.asm
%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

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

call    quit

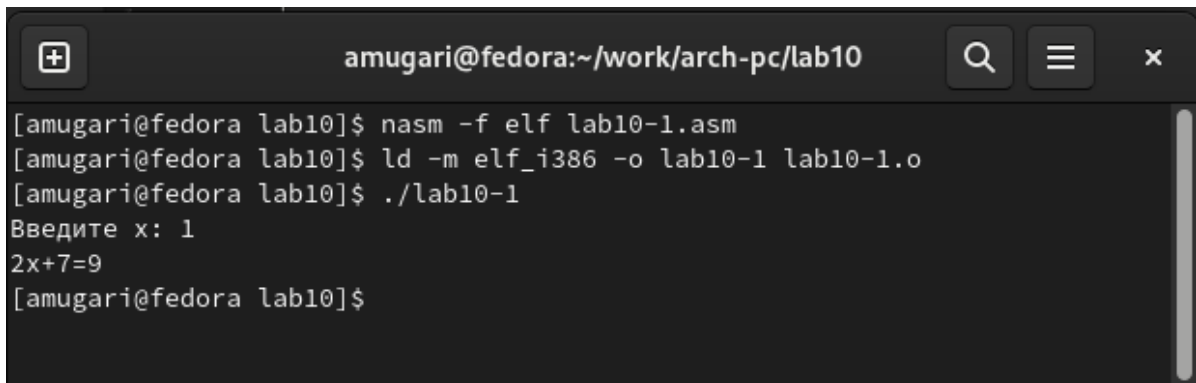
_calcul:
    mov     ebx, 2
    mul     ebx
    add     eax, 7
    mov     [res], eax

    ret

^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify
```

Рис. 2.2: Ресунок 2

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



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

Рис. 2.3: Ресунок 3

- 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)


```
amugari@fedora:~/work/arch-pc/lab10
GNU nano 6.0 /home/amugari/work/arch-pc/lab10/lab10-1.asm
#include 'in_out.asm'
SECTION .data
    msg:DB 'Введите x: ',0
    result:DB 'f(x)=3(2x+7)-1=',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, [res]

call    _subcalcul

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

call    quit

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

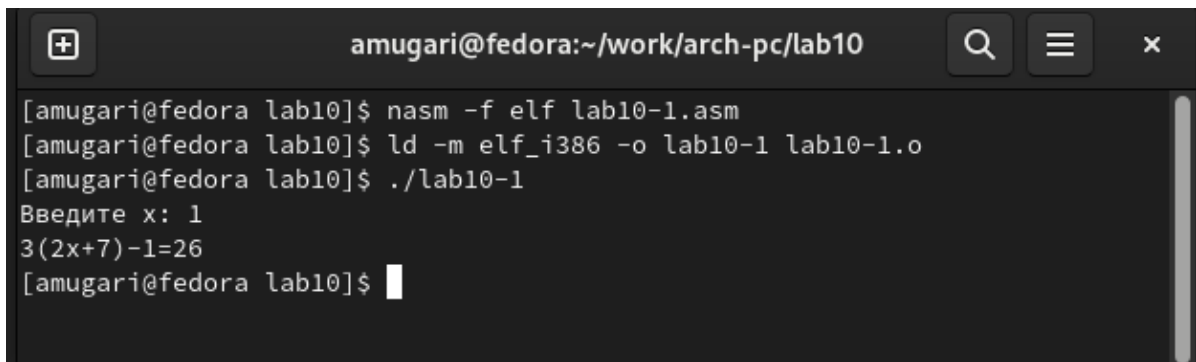
    ret

_subcalcul:
    mov     ebx, 3
    mul     ebx
    add     eax, -1
    mov     [res], eax

    ret

^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify
```

Рис. 2.4: Ресунок 4



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

Рис. 2.5: Ресунок 5

2.2 Отладка программ с помощью GDB :

1. На этом шаге мы создали файл **lab10-2.asm** с текстом программы из **листинга 10.2.** (*Программа печати сообщений Hello world!*). (рис. 2.6)



The screenshot shows a terminal window with the title bar "amugari@fedora:~/work/arch-pc/lab10". The address bar shows the file path "/home/amugari/work/arch-pc/lab10/lab10-2.asm". The terminal content is as follows:

```
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

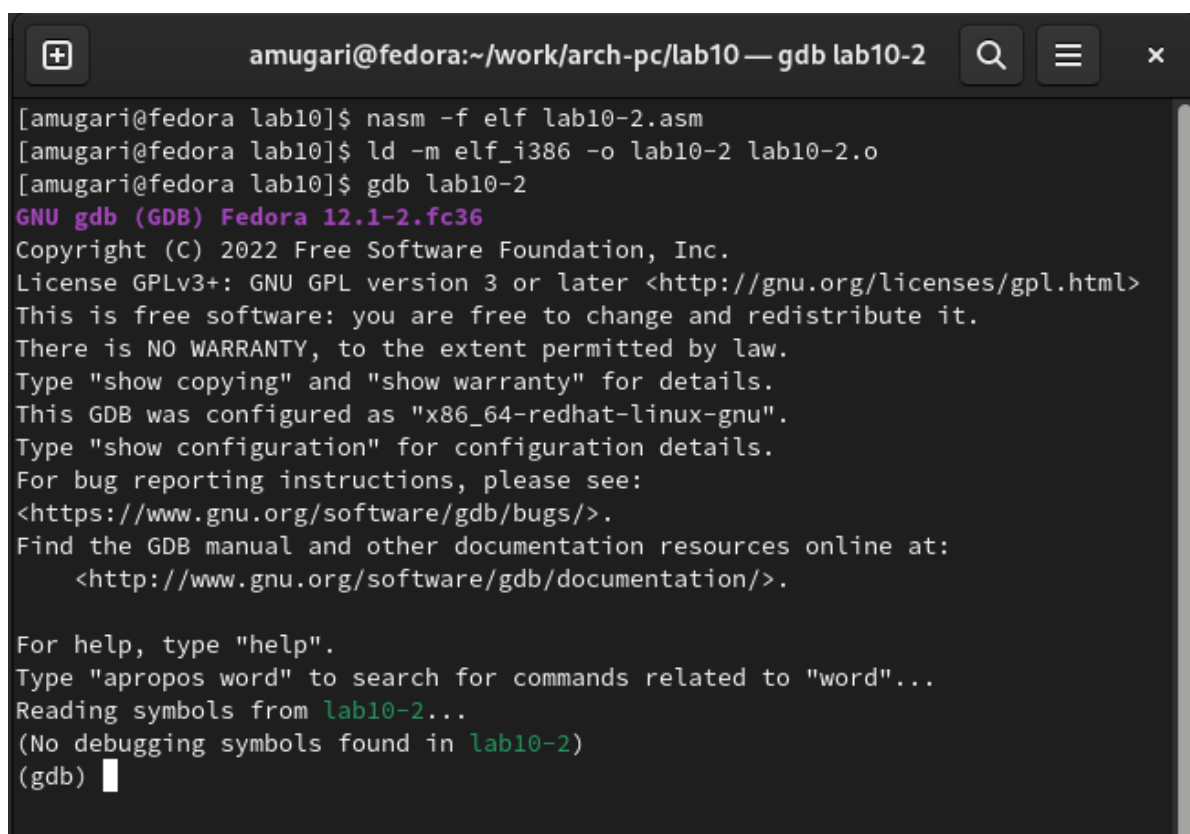
    mov     eax, 1
    mov     ebx, 0
    int     0x80
```

At the bottom of the terminal, there is a menu of keyboard shortcuts:

^G Help	^O Write Out	^W Where Is	^K Cut	^T Execute
^X Exit	^R Read File	^\ Replace	^U Paste	^J Justify

Рис. 2.6: Ресунок 6

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



```
amugari@fedora:~/work/arch-pc/lab10 — gdb lab10-2
[amugari@fedora lab10]$ nasm -f elf lab10-2.asm
[amugari@fedora lab10]$ ld -m elf_i386 -o lab10-2 lab10-2.o
[amugari@fedora lab10]$ gdb lab10-2
GNU gdb (GDB) Fedora 12.1-2.fc36
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-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...
(no debugging symbols found in lab10-2)
(gdb) █
```

Рис. 2.7: Ресунок 7

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

```

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...
(No debugging symbols found in lab10-2)
(gdb) run
Starting program: /home/amugari/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.
Hello, world!
[Inferior 1 (process 7621) exited normally]
(gdb)

```

Рис. 2.8: Ресунок 8

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

```

Hello, world!
[Inferior 1 (process 7621) exited normally]
(gdb) break _start
Breakpoint 1 at 0x08049000
(gdb) r
Starting program: /home/amugari/work/arch-pc/lab10/lab10-2

Breakpoint 1, 0x08049000 in _start ()
(gdb)

```

Рис. 2.9: Ресунок 9

5. 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.
(gdb)
```

Рис. 2.10: Ресунок 10

6. 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 11

- 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**.
7. Затем мы включили псевдографический режим для более удобного анализа программы. (рис. 2.12)

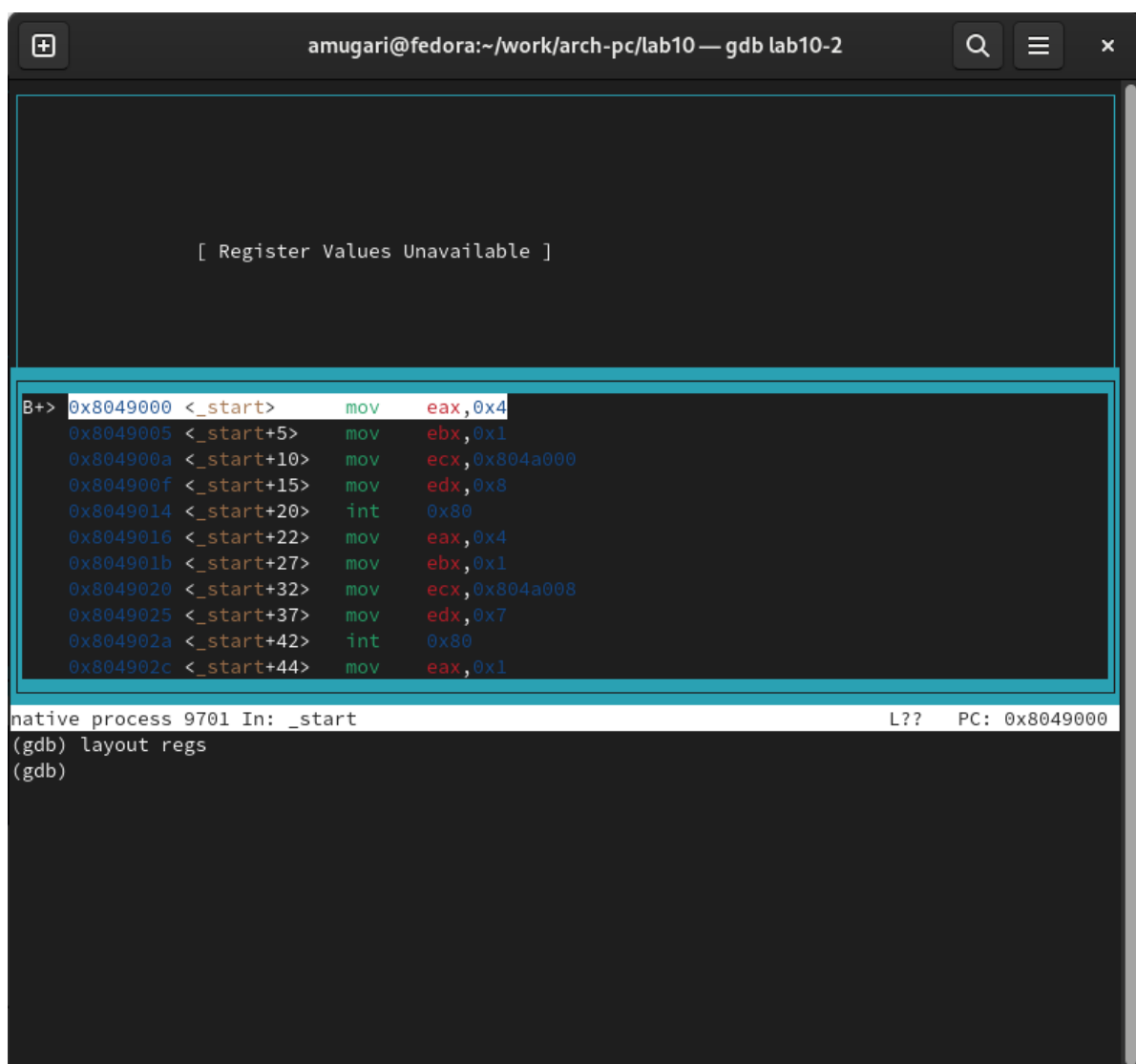


Рис. 2.12: Ресунок 12

2.3 Добавление точек останова :

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

```
Breakpoint 1, 0x08049000 in _start ()
(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 13

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

```
(gdb) b *0x08049031
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 14

2.4 Работа с данными программы в GDB :

1. 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. 2.16)


```
(gdb) info registers
eax            0x0            0
ecx            0x0            0
edx            0x0            0
ebx            0x0            0
esp            0xffffd140      0xffffd140
ebp            0x0            0x0
esi            0x0            0
edi            0x0            0
eip            0x8049000      0x8049000 <_start>
eflags         0x202          [ IF ]
cs             0x23           35
ss             0x2b           43
ds             0x2b           43
es             0x2b           43
fs             0x0            0
gs             0x0            0
```

Рис. 2.15: Ресунок 15

```

(gdb) stepi
0x08049005 in _start ()
(gdb) stepi
0x0804900a in _start ()
(gdb) stepi
0x0804900f in _start ()
(gdb) stepi
0x08049014 in _start ()
(gdb) stepi
Hello, 0x08049016 in _start ()
(gdb) i r
eax            0x8                8
ecx            0x804a000        134520832
edx            0x8                8
ebx            0x1                1
esp            0xffffd140       0xffffd140
ebp            0x0                0x0
esi            0x0                0
edi            0x0                0
eip            0x8049016        0x8049016 <_start+22>
eflags        0x202            [ IF ]
cs             0x23            35
ss             0x2b            43
ds             0x2b            43
es             0x2b            43
fs             0x0                0
gs             0x0                0
(gdb)

```

Рис. 2.16: Ресунок 16

- После проверки мы видим, что регистры : **eax,ecx,edx,ebx,esp** изменили свое значение.
2. Мы рассмотрели значение переменной `msg1` по имени, используя команду **x/1sb**(рис. 2.17)

```

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

```

Рис. 2.17: Ресунок 17

3. Здесь мы рассмотрели значение переменной `msg2`, используя адрес.(рис. 2.18)

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

Рис. 2.18: Ресунок 18

4. Здесь мы изменили первую букву переменной `msg1`, которая имеет тип `char`.(рис. 2.19)

```
(gdb) set {char}&msg1='h'

(gdb) x/1sb &msg1

0x804a000:      "hello, "
(gdb)
```

Рис. 2.19: Ресунок 19

5. После этого мы изменили первую букву переменной `msg2`.(рис. 2.20)

```
(gdb) set {char}&msg2='F'

(gdb) x/1sb &msg2

0x804a008:      "Forld!\n"
(gdb) █
```

Рис. 2.20: Ресунок 20

6. Затем мы выводим значение регистра **edx** в различных форматах (шестнадцатеричном, двоичном и символьном).(рис. 2.21)



```
(gdb) p/x $edx
$18 = 0x8
(gdb) p/s $edx
$19 = 8
(gdb) p/t $edx
$20 = 1000
(gdb) p/s $edx
$21 = 8
(gdb)
```

Рис. 2.21: Ресунок 21

7. Используя команду **set**, мы изменили значение регистра **ebx**, когда раз, введя '2', а в другой раз, введя 2.(рис. 2.22)

```
(gdb) set $ebx='2'
```

```
(gdb) p/s $ebx
```

```
$30 = 50
```

```
(gdb)
```

```
$31 = 50
```

```
(gdb) set $ebx=2
```

```
(gdb) p/s $ebx
```

```
$32 = 2
```

```
(gdb) █
```

Рис. 2.22: Рисунок 22

- 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.
8. Наконец, мы завершили программу с помощью **stepi** и вышли из GDB с помощью команды **quit**. (рис. 2.23)

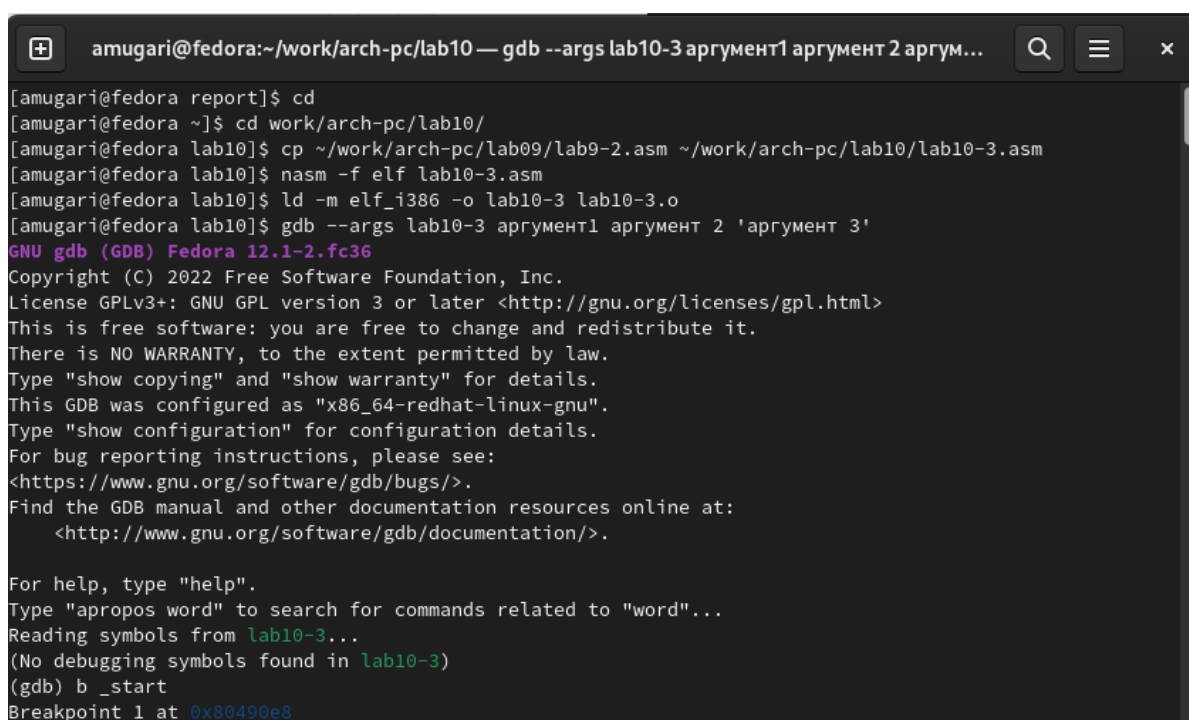
```
0x0804901b in _start ()
(gdb)
0x08049020 in _start ()
(gdb)
0x08049025 in _start ()
(gdb)
0x0804902a in _start ()
(gdb)
Forld!
0x0804902c in _start ()
(gdb)

Breakpoint 2, 0x08049031 in _start ()
(gdb)
0x08049036 in _start ()
(gdb)
[Inferior 1 (process 16165) exited normally]
(gdb)
The program is not being run.
(gdb)
The program is not being run.
(gdb) q
[amugari@fedora lab10]$
```

Рис. 2.23: Ресунок 23

2.5 Обработка аргументов командной строки в GDB :

1. На этом этапе мы скопировали файл **lab9-2.asm**, созданный при выполнении лабораторной работы №9 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 установили точку останова в ****_start**** и запустил отладчик.(рис. 2.24)



```
amugari@fedora:~/work/arch-pc/lab10 — gdb --args lab10-3 аргумент1 аргумент 2 аргум...
[amugari@fedora report]$ cd
[amugari@fedora ~]$ cd work/arch-pc/lab10/
[amugari@fedora lab10]$ cp ~/work/arch-pc/lab09/lab9-2.asm ~/work/arch-pc/lab10/lab10-3.asm
[amugari@fedora lab10]$ nasm -f elf lab10-3.asm
[amugari@fedora lab10]$ ld -m elf_i386 -o lab10-3 lab10-3.o
[amugari@fedora lab10]$ gdb --args lab10-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) Fedora 12.1-2.fc36
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-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...
(No debugging symbols found in lab10-3)
(gdb) b _start
Breakpoint 1 at 0x80490e8
```

Рис. 2.24: Ресунок 24

2. 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. 2.25)

```

amugari@fedora:~/work/arch-pc/lab10 — gdb --args lab10-3 аргумент1 аргумент2 аргум...
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, 0x080490e8 in _start ()
(gdb) x/x $esp
0xffffd100: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xffffd2c0: "/home/amugari/work/arch-pc/lab10/lab10-3"
(gdb)
0xffffd2e9: "аргумент1"
(gdb)
0xffffd2fb: "аргумент"
(gdb)
0xffffd30c: "2"
(gdb)
0xffffd30e: "аргумент 3"
(gdb)

```

Рис. 2.25: Ресунок 25

- Шаг изменения адреса равен 4, потому что размер регистра **esp** равен **32битам = 4 байтам**, а количество памяти равно количеству аргументов плюс имя программы, поэтому мы получили 5 шагов с 4 байтами для каждого шага.(рис. 2.26)

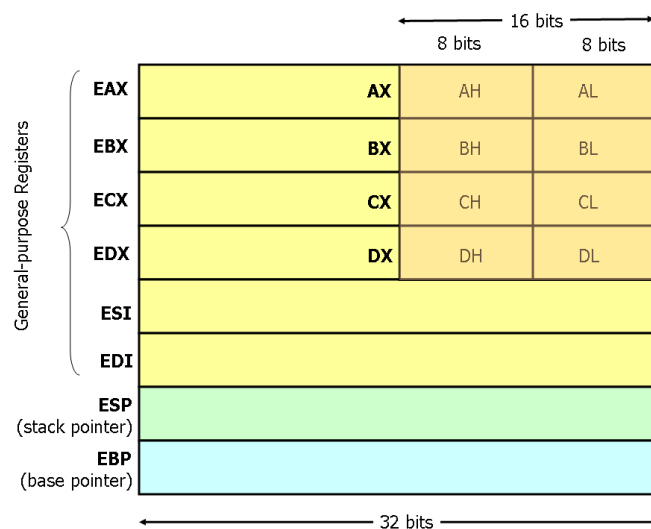


Рис. 2.26: Ресунок 26

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 :

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