Лабораторная работа №9

Понятие подпрограммы. Отладчик GDB

Коровкин Никита Михайлович

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Ознакомиться с понятием подпрограмм в Ассемблере и научиться использовать подпрограммы на практике. Ознакомиться с отладчиком gdb и научиться использовать его

2 Выполнение лабораторной работы

Сперва создадим рабочую директорию и первый файл: с которым мы будем работать.(рис.1)

```
liveuser@localhost-live:~$ mkdir ~/work/arch-pc/lab09
liveuser@localhost-live:~$ cd ~/work/arch-pc/lab09
liveuser@localhost-live:~/work/arch-pc/lab09$ touch lab09-1.asm
```

Рис. 2.1: Создание рабочей директории и файла lab9-1.asm

Далее подключим in_out.asm, перенеся его из папки прошлой лабораторной работы.(рис.2)

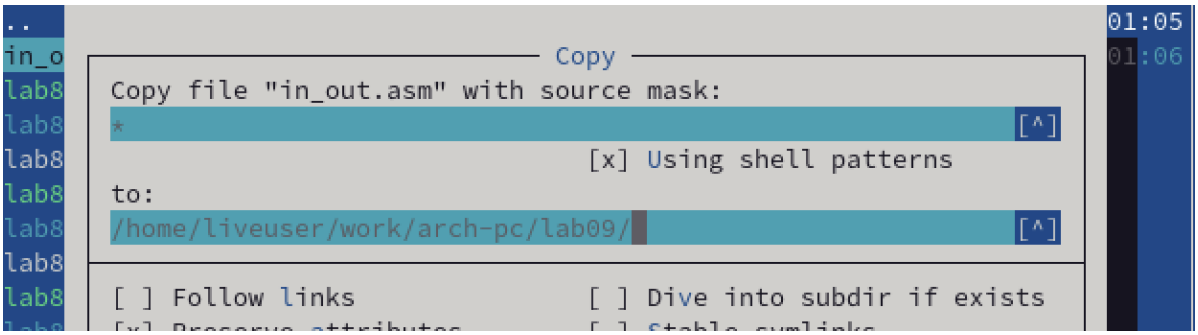

Рис. 2.2: Копирование файла in_out.asm в рабочую папку

Откроем файл и вставим код из первого листинга.(рис.3)

```
%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
```

Рис. 2.3: Вставляем код из первого листинга

Теперь соберем файл и запустим его.(рис.4)

```
liveuser@localhost-live:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
liveuser@localhost-live:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.
o
liveuser@localhost-live:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=17
```

Рис. 2.4: Запуск файла

Затем изменим файл, чтобы в подпрограмме была ещё одна подпрограмма, вычисляющая значение g(x), которая будет передавать значение в первую подпрограмму, которая бы уже вычислила значение f(g(x)).(рис.5)

```
_subcalcul:
mov ebx,3
mul ebx
sub ebx,1
ret
```

Рис. 2.5: Изменение кода файла lab9-1.asm

Запустим программу еще раз.(рис.6)

```
liveuser@localhost-live:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
liveuser@localhost-live:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.
o
liveuser@localhost-live:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 2
2x+7=19
liveuser@localhost-live:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 3
2x+7=25
```

Рис. 2.6: Повторный запуск файла lab9-1.asm

Все работает верно.

Теперь создадим второй файл.(рис.7)

```
liveuser@localhost-live:~/work/arch-pc/lab09$ touch lab09-2.asm
```

Рис. 2.7: Создание второго файла

Вставим туда код из листинга.(рис.8)

```
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
```

Рис. 2.8: Изменение файла lab9-2.asm

Запустим файл.(рис.9)

```
liveuser@localhost-live:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab0
9-2.asm
liveuser@localhost-live:~/work/arch-pc/lab09$ ld - m elf_i386 -o lab09-2 lab09-2
.o
```

Рис. 2.9: Запуск файла lab9-2.asm

Теперь загрузим программу в gdb.(рис.10)

```
liveuser@localhost-live:~/work/arch-pc/lab09$ gdb lab09-2
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/>.
```

Рис. 2.10: Загрузка программы в gdb

Запустим ее в отладчике.(рис.11)

```
(gdb) run
Starting program: /home/liveuser/work/arch-pc/lab09/lab09-2

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 21862) exited normally]
(gdb)
```

Рис. 2.11: Запуск программы в отладчике

После запуска создадим брейкпоинт на метке _start с помощью команды break.(рис.12)

```
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/liveuser/work/arch-pc/lab09/lab09-2

Breakpoint 1, _start () at lab09-2.asm:9
9       mov eax, 4
(gdb)
```

Рис. 2.12: Создание брейкпоинта

С помощью команды disassemble дизассемблируем метку.(рис.13)

```
(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. 2.13: Дизассемблирование

При помощи следующей команды переключаем синтаксис вывода на intel(рис.14)

```
(gdb) set disassembly-flavor intel
(gdb)
```

Рис. 2.14: Переключение синтаксиса

Дизассемблиурем программу еще раз(рис.15)

```
(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. 2.15: Повторное дизассемблирование программы

Теперь включаем графическое отображение кода.(рис.16)

```
B+>0x8049000 <_start>       mov    eax,0x4
   0x8049005 <_start+5>     mov    ebx,0x1
   0x804900a <_start+10>    mov    ecx,0x804a000
   0x804900f <_start+15>    mov    edx,0x8
   0x8049014 <_start+20>    int    0x80
   0x8049016 <_start+22>    mov    eax,0x4
   0x804901b <_start+27>    mov    ebx,0x1
   0x8049020 <_start+32>    mov    ecx,0x804a008
   0x8049025 <_start+37>    mov    edx,0x7
   0x804902a <_start+42>    int    0x80
   0x804902c <_start+44>    mov    eax,0x1
   0x8049031 <_start+49>    mov    ebx,0x0
   0x8049036 <_start+54>    int    0x80
native process 21873 In: _start                                L9    PC:
(gdb)
```

Рис. 2.16: Графическое отображение кода

Включаем графическое отображение значений регистров.(рис.17)

Рис. 2.17: Графическое отображение регистров

Выведем отображение информации о имеющихся брейкпоинтах.(рис.18)

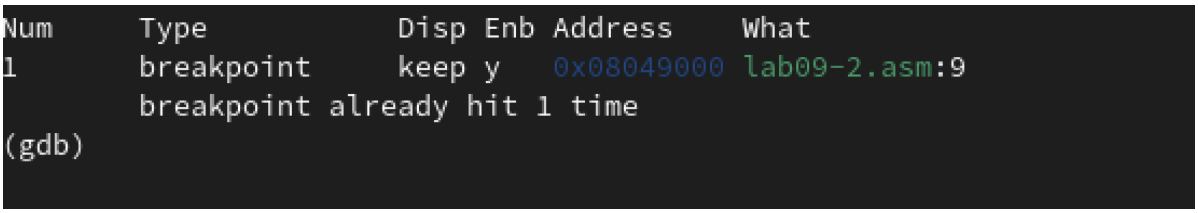

Рис. 2.18: Отображение информации о брейкпоинтах

А теперь создадим брейкпоинт самостоятельно.(рис.19)

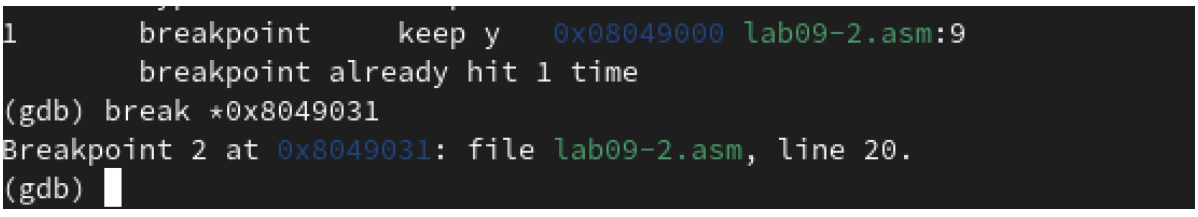

Рис. 2.19: Создание брейкпоинта по адресу

Выведем информацию о брейкпоинтах еще раз.(рис.20)

```
(gdb) i b
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x08049000 lab09-2.asm:9
        breakpoint already hit 1 time
2       breakpoint     keep y   0x08049031 lab09-2.asm:20
```

Рис. 2.20: Повторных вывод информации о брейкпоинтах

Воспользуемся командой si для построчного выполнения кода. Задействуем ее 5 раз.(рис.21-22)

```
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/liveuser/work/arch-pc/lab09/lab09-2


Breakpoint 1, _start () at lab09-2.asm:9
(gdb) si
(gdb)
```

Рис. 2.21: Построчное выполнение кода

```


Breakpoint 1, _start () at lab09-2.asm:9
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)
```

Рис. 2.22: Построчное выполнение кода

Как видим, поменялись значения регистров eax, ecx, edx и ebx. Теперь выведем информацию о значениях регистров(рис.23)

```
eax            0x8                 8
ecx            0x804a000           134520832
edx            0x8                 8
ebx            0x1                 1
esp            0xffffd0b0          0xffffd0b0
ebp            0x0                 0x0
esi            0x0                 0
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 2.23: Изменение значений регистров

Теперь выведем значение переменной по имени и по адресу.(рис.24-25)

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

Рис. 2.24: Вывод значения переменной по имени

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

Рис. 2.25: Вывод значения переменной по адресу

После этого изменим первый символ переменной.(рис.26)

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

Рис. 2.26: Изменение первого символа

А затем изменим несколько символов переменной, обращаясь по адресу.(рис.27)

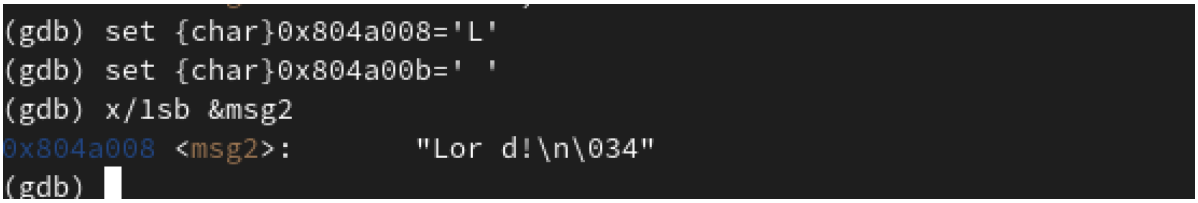

Рис. 2.27: Изменение нескольких символов

Теперь предстоит вывести значение регистра в изначальном, двоичном и шестнадцатиричном виде(рис.28)

Рис. 2.28: Вывод значения регистра в разном виде

Изменим значение регистра.(рис.29)

Рис. 2.29: Изменение значения регистра

Нетрудно заметить, что в регистр записались разные значения, так как в одном случае мы записываем туда число - в другом строку.

Завершаем работу программы с помощью continue и выходим из отладчика.(рис.30)

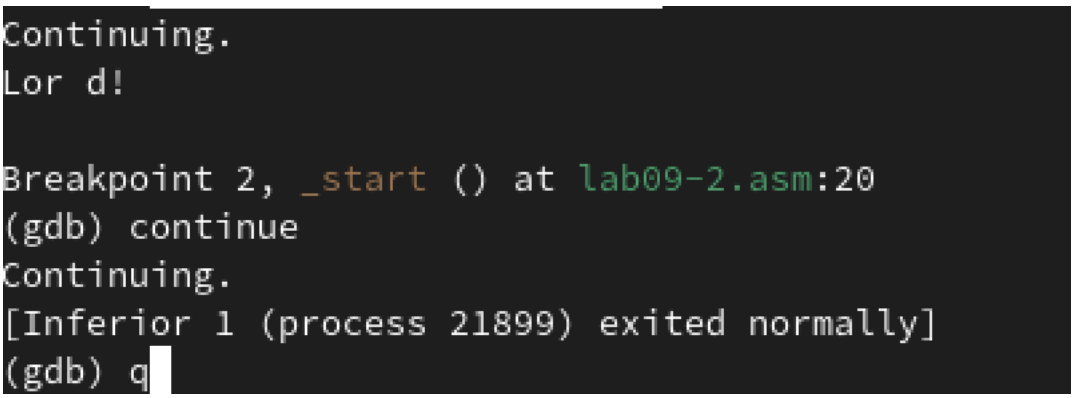

Рис. 2.30: Завершение работы

Теперь скопируем файл из предыдущей лабораторной работы.(рис.21)

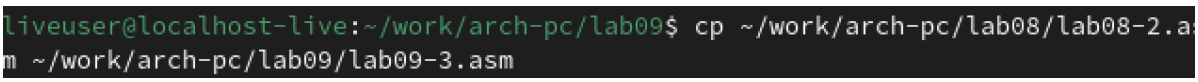

Рис. 2.31: Копирование предыдущей лабораторной работы

Мы так же соберем его и загрузим в gdb.(рис.32)

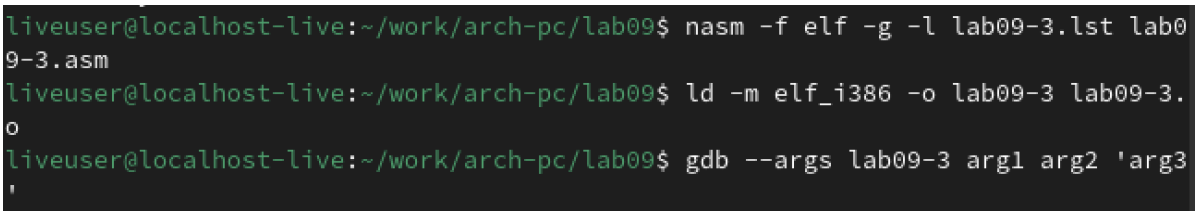

Рис. 2.32: Загрузка файла предыдущей работы в gdb

Теперь создаем брейкпоинт и запускаем программу.(рис.33)

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/liveuser/work/arch-pc/lab09/lab09-3 arg1 arg2 arg3

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 lab09-3.asm:5
5       pop ecx ; Извлекаем из стека в `ecx` количество
(gdb)
```

Рис. 2.33: Запуск программы

Затем выведем значение регистра esp, где хранятся данные о стеке.(рис.34)

```
(gdb) x/x $esp
0xffffd090:     0x00000004
(gdb)
```

Рис. 2.34: Вывод значения регистра

Теперь нужно вывести значение всех элементов стека.(рис.35)

```
(gdb) x/s *(void**)($esp + 4)
0xffffd25d:     "/home/liveuser/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd287:     "arg1"
(gdb) x/s *(void**)($esp + 12)
0xffffd28c:     "arg2"
(gdb) x/s *(void**)($esp + 16)
0xffffd291:     "arg3"
(gdb) x/s *(void**)($esp + 20)
0x0:    <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 2.35: Вывод значений всех элементов стека

Можно заметить, что для вывода каждого элемента нужно менять значения адреса с шагом на 4. Это связано с тем, что под каждый элемент выделяется 4 байта.

#Выполнение самостоятельной работы

Первым делом копируем файл задания из прошлой лабораторной работы.(рис.36)

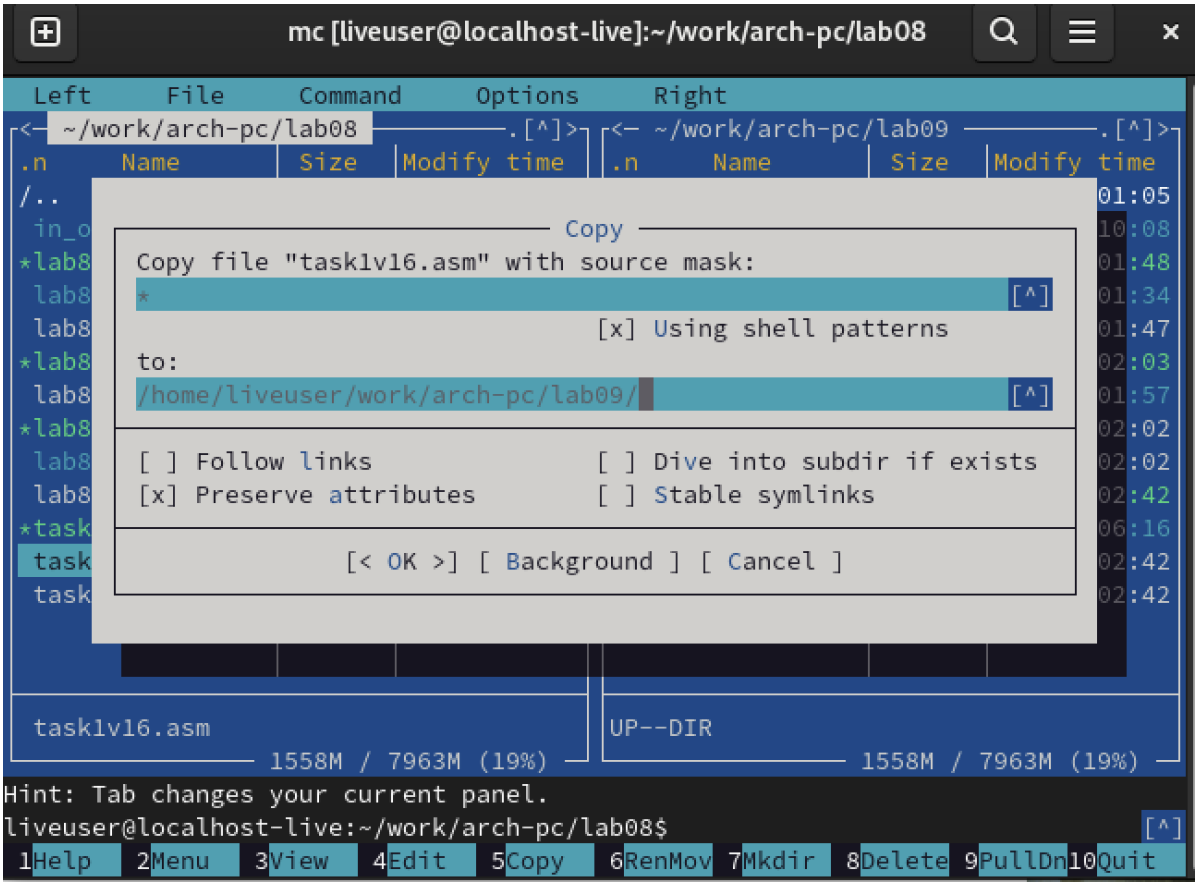

Рис. 2.36: Копируем файл из предыдущей работы

Его код выглядит так.(рис.37)

```
SECTION .data
msg db "Результат: ",0
msg2 db "Функция: f(x)=30x-11"
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
add esi,eax
loop next
_end:
mov eax, msg2
call sprintLF
mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit
_calcul:
mov ebx, 30
mul ebx
sub eax, 11
ret
```

Рис. 2.37: Код в файле

Теперь соберем код и проверим.(рис.38)

```
liveuser@localhost-live:~/work/arch-pc/lab09$ nasm -f elf task1v16.asm
liveuser@localhost-live:~/work/arch-pc/lab09$ ld -m elf_i386 -o  task1v16 task1v
16.o
liveuser@localhost-live:~/work/arch-pc/lab09$ ./task1v16 1 2 3 4
Функция: f(x)=30x-11
Результат: 256
liveuser@localhost-live:~/work/arch-pc/lab09$
```

Рис. 2.38: Запуск кода

Создадим второй файл.(рис.39)

```
liveuser@localhost-live:~/work/arch-pc/lab09$ touch task2.asm
liveuser@localhost-live:~/work/arch-pc/lab09$
```

Рис. 2.39: Второй файл

Вставляем в него код из листинга.(рис.40)

```
%include 'in_out.asm'
SECTION .data
div: DB 'Результат: ',0
SECTION .text
GLOBAL _start
_start:
; ---- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

Рис. 2.40: Вставляем код из листинга

Запускаем файл.(рис.41)

```
liveuser@localhost-live:~/work/arch-pc/lab09$ nasm -f elf -g -l task2.lst task2.
asm
liveuser@localhost-live:~/work/arch-pc/lab09$ ld -m elf_i386 -o task2 task2.o
liveuser@localhost-live:~/work/arch-pc/lab09$ ./task2
Результат: 10
```

Рис. 2.41: Запуск второго файла

Вставим наш файл в gdb(рис.42)

```
liveuser@localhost-live:~/work/arch-pc/lab09$ gdb task2
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/>.
```

Рис. 2.42: Вставляем файл в gdb

Переключаем на синтаксис и включаем графическое отображение.(рис.43)

```
   0x80490e8 <_start>      mov    ebx,0x3
   0x80490ed <_start+5>    mov    eax,0x2
   0x80490f2 <_start+10>   add    ebx,eax
   0x80490f4 <_start+12>   mov    ecx,0x4
   0x80490f9 <_start+17>   mul    ecx
   0x80490fb <_start+19>   add    ebx,0x5
exec No process In:                                          L??   PC: ??
(gdb) lasyout regs
Undefined command: "lasyout".  Try "help".
(gdb) layour regs
Undefined command: "layour".  Try "help".
(gdb) layout regs
(gdb)
```

Рис. 2.43: Переключение синтаксиса и включение графического отображения

Затем устанавливаем брейкпоинт(рис.44)

```
(gdb) break _start
Breakpoint 1 at 0x80490e8: file task2.asm, line 8.
(gdb)
```

Начинаем выполнение кода.(рис.45-46)

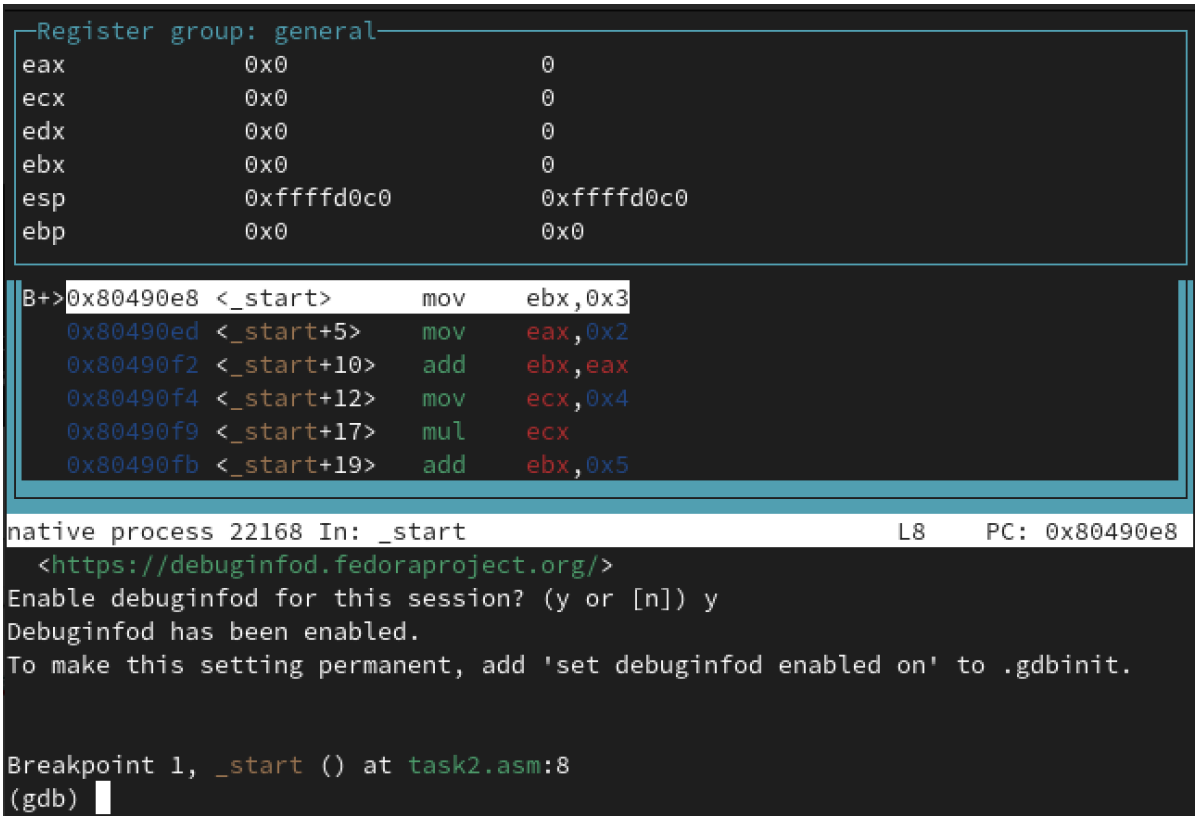

Рис. 2.44: Выполнение кода

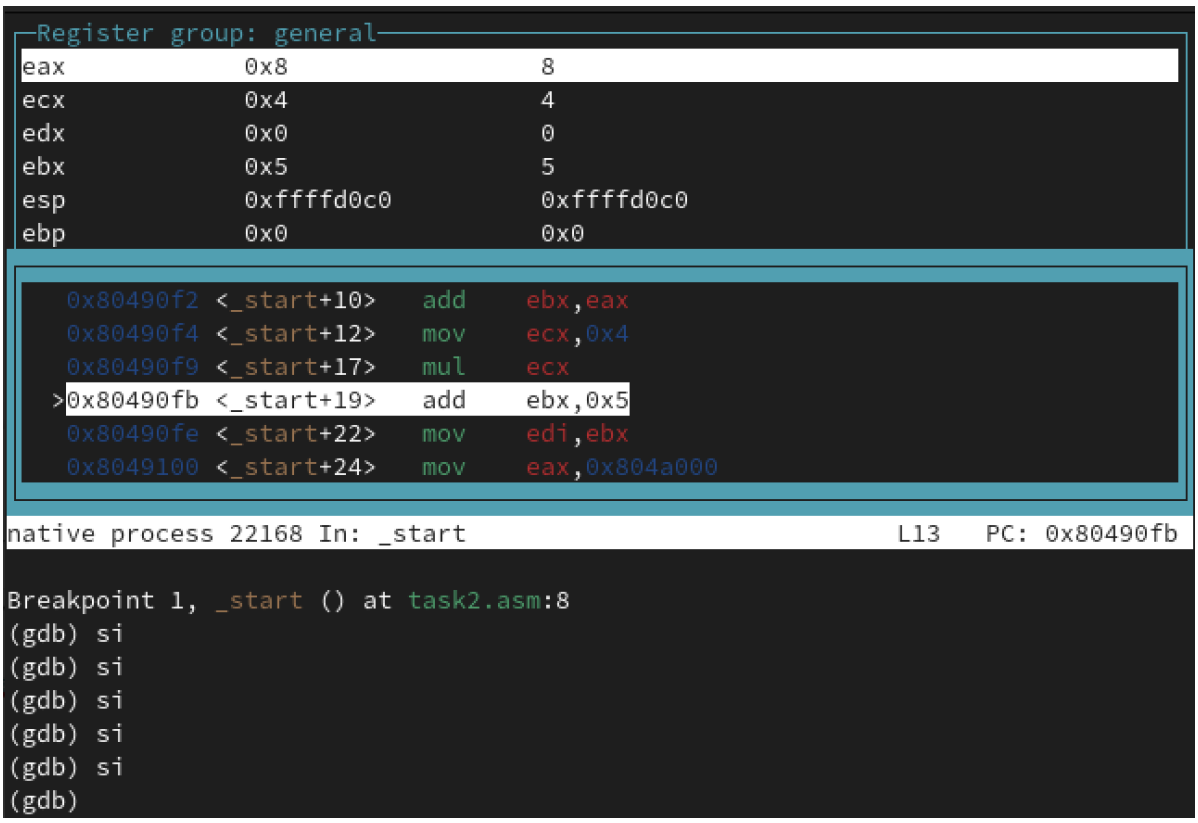

Рис. 2.45: Выполнение кода

Нетрудно заметить,что мы должны были умножить значение регистра ebx, но сделали это с eax.

Отредактируем код и запустим снова.(рис.47)

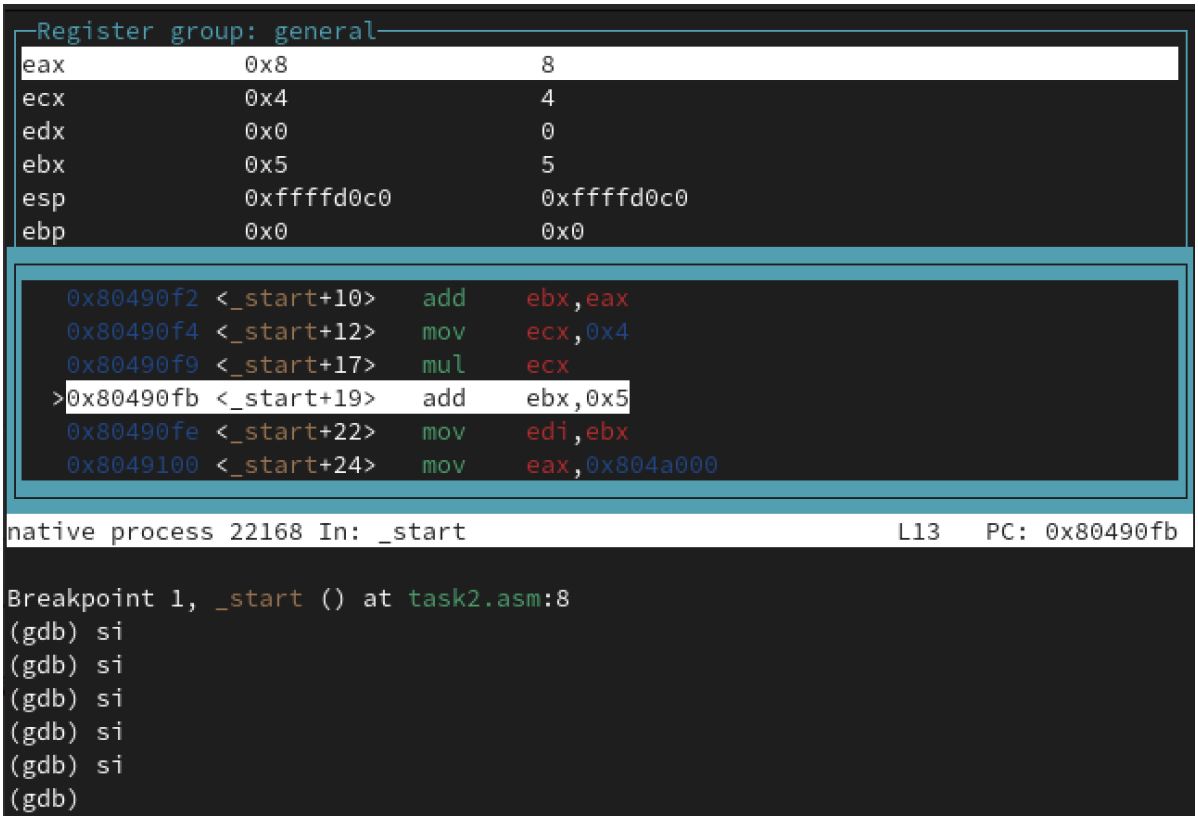

Рис. 2.46: Повторный запуск

Все вывелось верно. Работа сделана правильно.

3 Выводы

В результате выполнения данной лабораторной работы мы научились использовать подпрограммы, узнали как пользоваться отладчиком и изучили его функции.