Лабораторная работа №9

Понятие подпрограммы. Отладчик GDB

Колонтырский Илья Русланович

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 Выполнение лабораторной работы

Создадим папку и файл lab9-1.asm (Рис. 2.1)

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

Рис. 2.1: Создание папки и файла lab9-1.asm

Вставим в файл lab9-1.asm предложенный код (Рис. 2.2)

```
 GNU nano 6.2   /home/irkolontyrskiy/work/arch-pc/lab09/lab09-1.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:
;---------------------------------------
; Основная программа
;---------------------------------------
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
;---------------------------------------
; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
mov ebx,2
mul ebx
add eax,7
mov [res],eax
ret ; выход из подпрограммы
```

Рис. 2.2: Вставка кода

Скомпилируем программу и посмотрим на результат (Рис. 2.3)

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

Рис. 2.3: Компиляция программы и запуск

Изменим программу, добавив подпрограмму, которая вычисляет g(x), после

чего выводит результат вычисления f(g(x)) (Рис. 2.4)

```
  GNU nano 6.2   /home/irkolontyrskiy/work/arch-pc/lab09/lab09-1.asm *
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
f: DB 'f(x)=2x+7',0
g: DB 'g(x)=3x-1',0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
;---------------------------------------
; Основная программа
;---------------------------------------
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,f
call sprintLF
mov eax,g
call sprintLF
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 ; выход из подпрограммы
_subcalcul:
mov ebx,3
mul ebx
sub eax,1
ret
```

Рис. 2.4: Изменение файла

Скомпилируем программу и посмотрим на результат (Рис. 2.5)

```
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$ ld -m elf_i386 lab09-1.o -o
lab09-1
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
f(x)=2x+7
g(x)=3x-1
35
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$
```

Рис. 2.5: Компиляция программы и повторный запуск

Создадим ещё один файл (Рис. 2.6)

```
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$ touch lab09-2.asm
```

Рис. 2.6: Создание файла lab9-2.asm

Вставим файл следующий код (Рис. 2.7)

```
 GNU nano 6.2   /home/irkolontyrskiy/work/arch-pc/lab09/lab09-2.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
```

Рис. 2.7: Вставка кода

Скомпилируем программу для отладки (с аргументом -g) и загрузим в gdb

(Рис. 2.8)

```
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.ls
t lab09-2.asm
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 la
b09-2.o
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$ gdb lab09-2
```

Рис. 2.8: Компиляция файла для отладки

В gdb запустим её с помощью run (Рис. 2.9)

```
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
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-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 lab09-2...
(gdb) run
Starting program: /home/irkolontyrskiy/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 9157) exited normally]
(gdb)
```

Рис. 2.9: Запуск программы

На метке _start создадим брейкпоинт (Рис. 2.10)

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

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

Рис. 2.10: Создание брейкпоинта

Теперь дизассемблируем программу (Рис. 2.11)

```
(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. 2.11: Дизассемблирование

Теперь сделаем вывод на синтаксисе intel (Рис. 2.12)

```
(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. 2.12: Включение синтаксиса intel

Включим отображения кода и регистров (Рис. 2.13)

```
(gdb) layout asm
(gdb) layout regs
```

Рис. 2.13: Включение отображения кода и регистров

Выведем инормацию о брейкпоинтах (Рис. 2.14)

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

Рис. 2.14: Вывод информации о брейкпоинтах

Теперь создадим брейкпоинт не по имени, а по адресу, а также выведем информацию обо всех брейкпоинтах (Рис. 2.15)

```
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
b+  0x8049031 <_start+49>   mov    ebx,0x0
    0x8049036 <_start+54>   int    0x80
    0x8049038               add    BYTE PTR [eax],al
    0x804903a               add    BYTE PTR [eax],al
native process 9347 In: _start                          L9    PC: 0x8049000
(gdb) layout regs
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 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
(gdb)
```

Рис. 2.15: Создание брейкпоинта по адресу и вывод информации о всех брейкпоинтах

Теперь выполним команду si 5 раз, выполнив тем самым последовательно 5 строчек кода (Рис. 2.16)

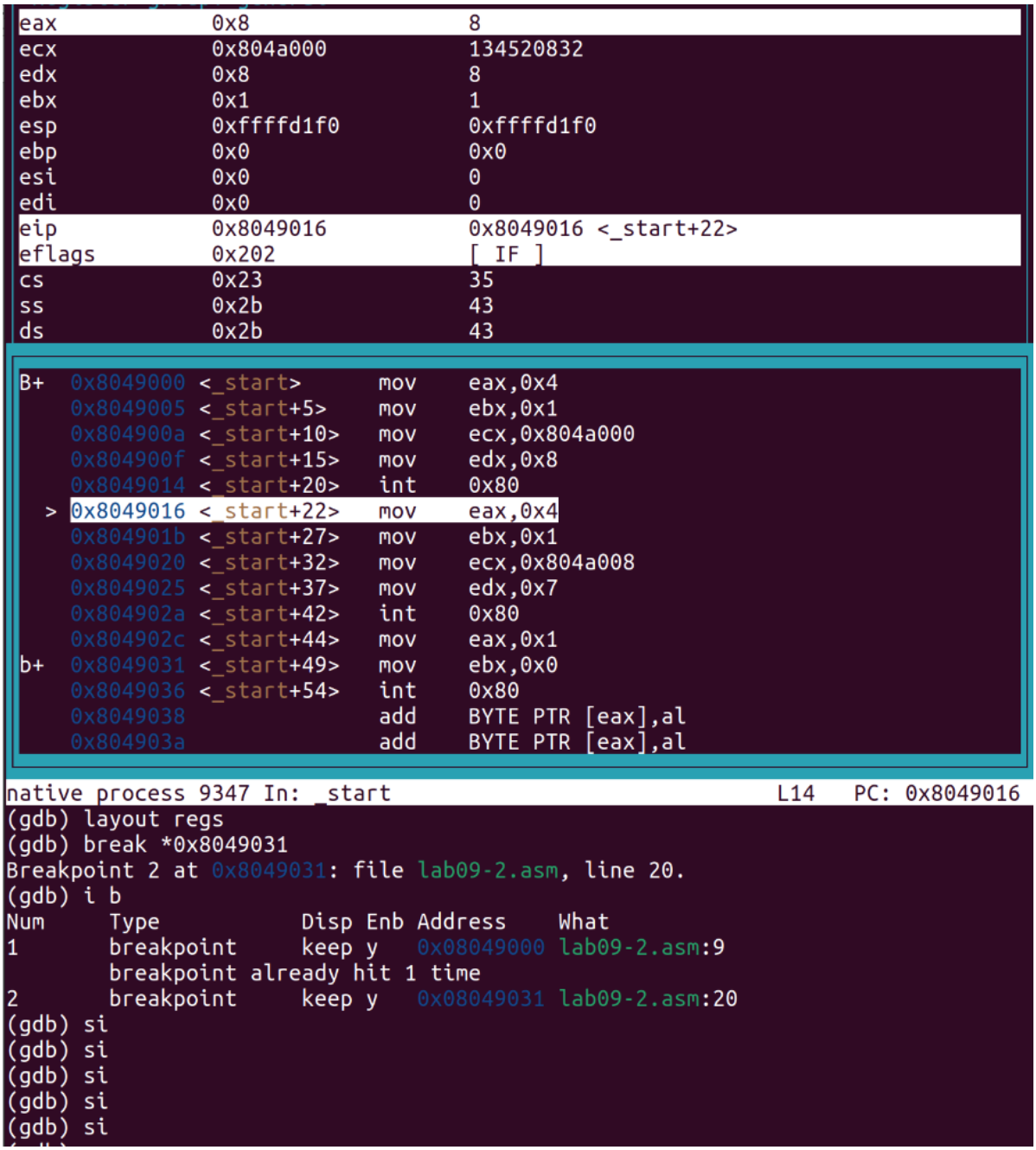

Рис. 2.16: Использование si

За 5 строк поменялись значения eax, ecx, edx и ebx регистров. Выведем инфор-

мацию о значениях регистров (Рис. 2.17)

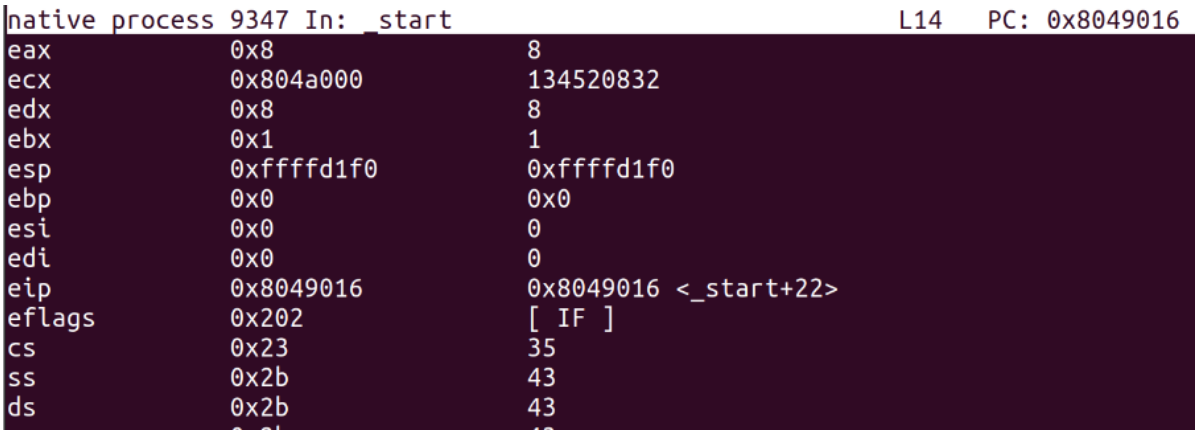

Рис. 2.17: Вывод значений регистров

Выведем значение переменной (Рис. 2.18)

Рис. 2.18: Вывод переменной

Выведем теперь значение переменной по адресу (Рис. 2.19)

Рис. 2.19: Вывод переменной по адресу

Изменим значение переменной и выведем его (Рис. 2.20)

Рис. 2.20: Изменение переменной и вывод значения

Изменим вторую переменную и выведем её (Рис. 2.21)

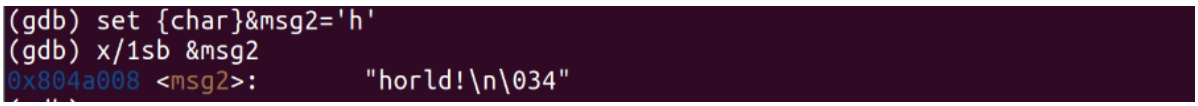

Рис. 2.21: Изменение второй переменной и вывод значения

Выведем значение регистра edx в строковом, двоичном, шестнадцатиричном видах (Рис. 2.22)

Рис. 2.22: Вывод значения регистра

Изменим теперь значение регистра (Рис. 2.23)

Рис. 2.23: Изменение регистра

Мы записали в регистр 2 разных значения. В первом случае это строка, а во втором - число. Выйдем из отладчика и скопируем файл из прошлой работы, соберм его и вгрузим в отладчик (Рис. 2.24)

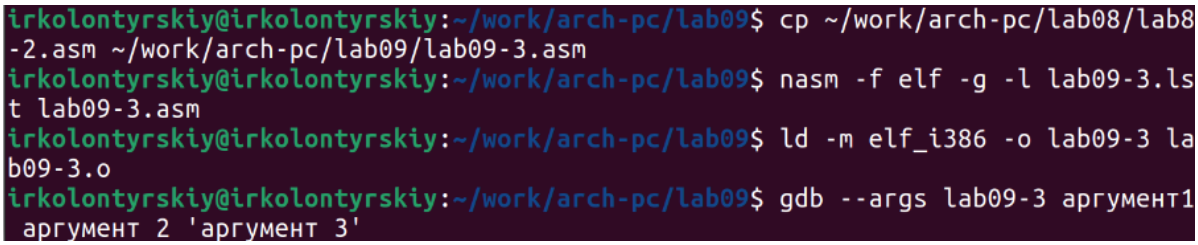

Рис. 2.24: Копирование файла из прошлой работы и выгрузка в gdb

Создадим брейкпоинт и запустим его, после чего выведем значение регистра esp (Рис. 2.25)

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/irkolontyrskiy/work/arch-pc/lab09/lab09-3 аргумент1 аргу
мент 2 аргумент\ 3

Breakpoint 1, _start () at lab09-3.asm:5
5       pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd1a0:     0x00000005
(gdb)
```

Рис. 2.25: Создание брейкпоинта и запуск, вывод значения esp

Выведем значение элементов стека (Рис. 2.26)

```
(gdb) x/s *(void**)($esp + 4)
0xffffd35e:     "/home/irkolontyrskiy/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd38e:     "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffd3a0:     "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffd3b1:     "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd3b3:     "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0:    <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 2.26: Вывод всех значений в стеке

Поскольку под каждый элемент стека выделяется 4 байта, мы должны выводить элементы с шагом в 4

Самостоятельная работа

Создадим файлы для самостоятельной работы (Рис. 2.27)

```
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$ touch v19-1.asm
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$ touch v19-2.asm
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$
```

Рис. 2.27: Создание файлов

В первом файле нам нужно переписать файл первой самостоятельной работы прошлой лабораторной работы с использованием подпрограмм (Рис. 2.28)

```
 GNU nano 6.2    /home/irkolontyrskiy/work/arch-pc/lab09/v19-1.asm *
%include 'in_out.asm'
SECTION .data
msg db "Результат: ",0
msg2 db "Функция: f(x)=8x-3",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 _calcul
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, 8
mul ebx
sub eax, 3
ret
```

Рис. 2.28: Редактирование программы

Скомпилируем файл и проверим его работу (Рис. 2.29)

```
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$ nasm -f elf v19-1.asm
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$ ld -m elf_i386 -o v19-1 v19-
1.o
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$ ./v19-1 1 2 3 4
Функция: f(x)=8x-3
Результат: 68
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$
```

Рис. 2.29: Сборка и проверка работы программы

Вставим во второй код предложенный код (Рис. 2.30)

```
 GNU nano 6.2    /home/irkolontyrskiy/work/arch-pc/lab09/v19-2.asm *
%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. 2.30: Вставка кода

Соберём его и вставим в gdb (Рис. 2.31)

```
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$ nasm -f elf v19-2.asm
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$ ld -m elf_i386 -o v19-2 v19-
2.o
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$ gdb v19-2
```

Рис. 2.31: Сборка программы и вгрузка в gdb

Начнём построчно выполнять код (Рис. 2.32):

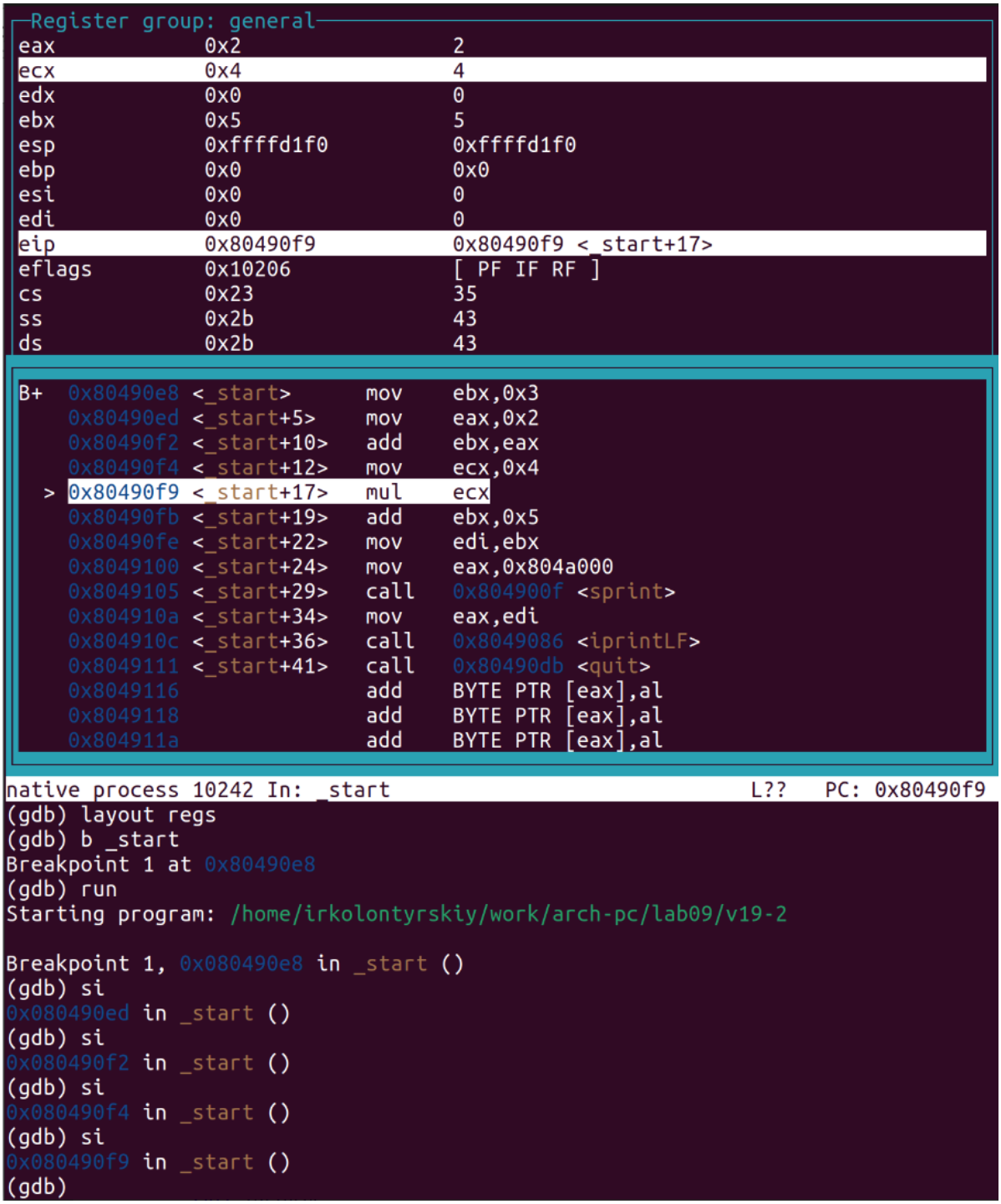

Рис. 2.32: Построчное выполнение кода

Видно, что при умножении результат сохраняется в eax, хотя кодом подразумевается сохранение в ebx. Запустим код и убедимся, что он работает неверно

(Рис. 3.33)

```
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$ ./v19-2
Результат: 10
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$
```

Рис. 2.33: Проверка программы

Изменим его так, чтобы умножалось нужное нам значение, а не то, что хранится в регистре eax изначально (Рис. 2.24)

```
 GNU nano 6.2    /home/irkolontyrskiy/work/arch-pc/lab09/v19-2.asm *
%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 eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit
```

Рис. 2.34: Изменение кода

Проверим правильность его работы, скомпилировав его и запустив (Рис. 2.35)

```
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$ nasm -f elf v19-2.asm
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$ ld -m elf_i386 -o v19-2 v19-
2.o
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$ ./v19-2
Результат: 25
irkolontyrskiy@irkolontyrskiy:~/work/arch-pc/lab09$
```

Рис. 2.35: Компиляция кода и проверка работы

Теперь код работает правильно

3 Выводы

Были получены навыки работы с adb, а также получены навыки работы с подпрограммами