**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

**Факультет физико-математических и естественных наук**

**Кафедра прикладной информатики и теории вероятностей**

# ОТЧЕТ

## по лабораторной работе № 9

*дисциплина: Архитектура компьютера*

Студент: Бессонов Андрей Максимович

Группа: НКАбд - 01 - 25

**МОСКВА**

2025 г.

# Содержание

# Список иллюстраций

# 1. Цель работы

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

# 2. Теоретическое введение

2.1. Понятие об отладке

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, отрабатывает, 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.

2.2. Методы отладки

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 строчки, 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.

2.3 Понятие подпрограммы

Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом.

Если в программе встречается одинаковый участок кода, его можно оформить в виде подпрограммы, а во всех нужных местах поставить её вызов. При этом подпрограмма будет содержаться в коде в одном экземпляре, что позволит уменьшить размер кода всей программы.

# 3. Выполнение лабораторной работы

Создадим каталог для выполнения лабораторной работы № 9, перейдем в него и создадим файл lab09-1.asm.

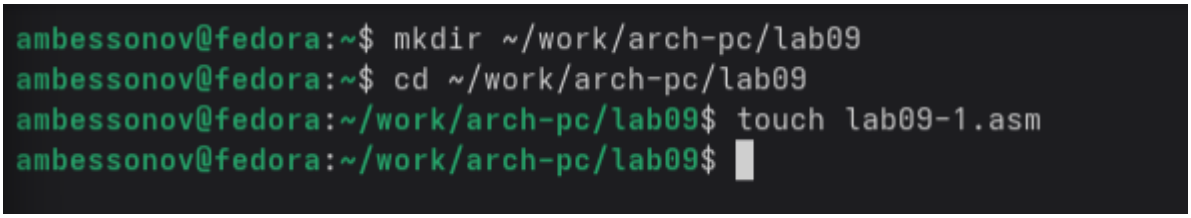


Рис 3.1: 31

Введем в файл lab09-1.asm текст программы из листинга 9.1. Создадим исполняемый файл и проверим его работу.

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

Рис 3.2: 32

Изменим текст программы, добавив подпрограмму _subcalcul в подпрограмму _calcul, для вычисления выражения $f(g(x))$, где $x$ вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$. Т.е. $x$ передается в подпрограмму _calcul из нее в подпрограмму _subcalcul, где вычисляется выражение $g(x)$, результат возвращается в _calcul и вычисляется выражение $f(g(x))$.

```
    call quit

;---------------------------------------
; Подпрограмма вычисления f(g(x))
; Вход:  eax = x
; Выход: eax = 2*(3x-1)+7
;---------------------------------------
_calcul:
    push ebx            ; Сохраняем ebx в стеке

    ; Вычисляем g(x) = 3x-1
    call _subcalcul     ; Вход: eax = x, Выход: eax = g(x) = 3x-1

    ; Вычисляем f(g(x)) = 2*g(x) + 7
    mov ebx, 2
    mul ebx             ; eax = 2*g(x)
    add eax, 7          ; eax = 2*g(x) + 7

    mov [res], eax      ; Сохраняем результат

    pop ebx             ; Восстанавливаем ebx
    ret

;---------------------------------------
; Подпрограмма вычисления g(x) = 3x-1
; Вход:  eax = x
; Выход: eax = 3x-1
;---------------------------------------
_subcalcul:
    mov ebx, 3
    mul ebx             ; eax = 3*x
    sub eax, 1          ; eax = 3*x - 1
    ret                 ; Возвращаем результат в _calcul
```

Рис 3.3: 33

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

Рис 3.4: 34

Создадим файл lab09-2.asm с текстом программы из Листинга 9.2

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

Открыть

```
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
```

Рис 3.5: 35

Получим исполняемый файл.

```
ambessonov@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
ambessonov@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
```

Рис 3.6: 36

Загрузим исполняемый файл в отладчик gdb.

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

```
ambessonov@fedora:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Fedora Linux) 16.2-3.fc42
Copyright (C) 2024 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 lab09-2...
(gdb) run
Starting program: /home/ambessonov/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]) n
Debuginfod has been disabled.
To make this setting permanent, add 'set debuginfod enabled off' to .gdbinit.
Hello, world!
[Inferior 1 (process 14053) exited normally]
(gdb)
```

Рис 3.7: 37

Для более подробного анализа программы установим брейкпоинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запустим её.

```
(gdb) break _start
Breakpoint 1 at 0x8048080: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/ambessonov/work/arch-pc/lab09/lab09-2

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

Рис 3.8: 38

Посмотрим дисассимилированный код программы с помощью команды disassemble начиная с метки _start.


Рис 3.9: 39

Переключимся на отображение команд с Intel'овским синтаксисом, введя команду set disassembly-flavor intel.


Рис 3.10: 310

Различия синтаксиса AT&T и Intel:

1. Порядок операндов

AT&T: `назначение ← источник` (обратный порядок)
Intel: `назначение ← источник` (прямой порядок)

2. Префиксы регистров и констант

AT&T: Регистры с `%`, константы с `$`, размер операнда в суффиксе инструкции
mov $0x4,%eax ; $ - константа, % - регистр
Intel: Без префиксов, размер операнда определяется по регистру
mov eax,0x4 ; без префиксов

Включим режим псевдографики для более удобного анализа программы.

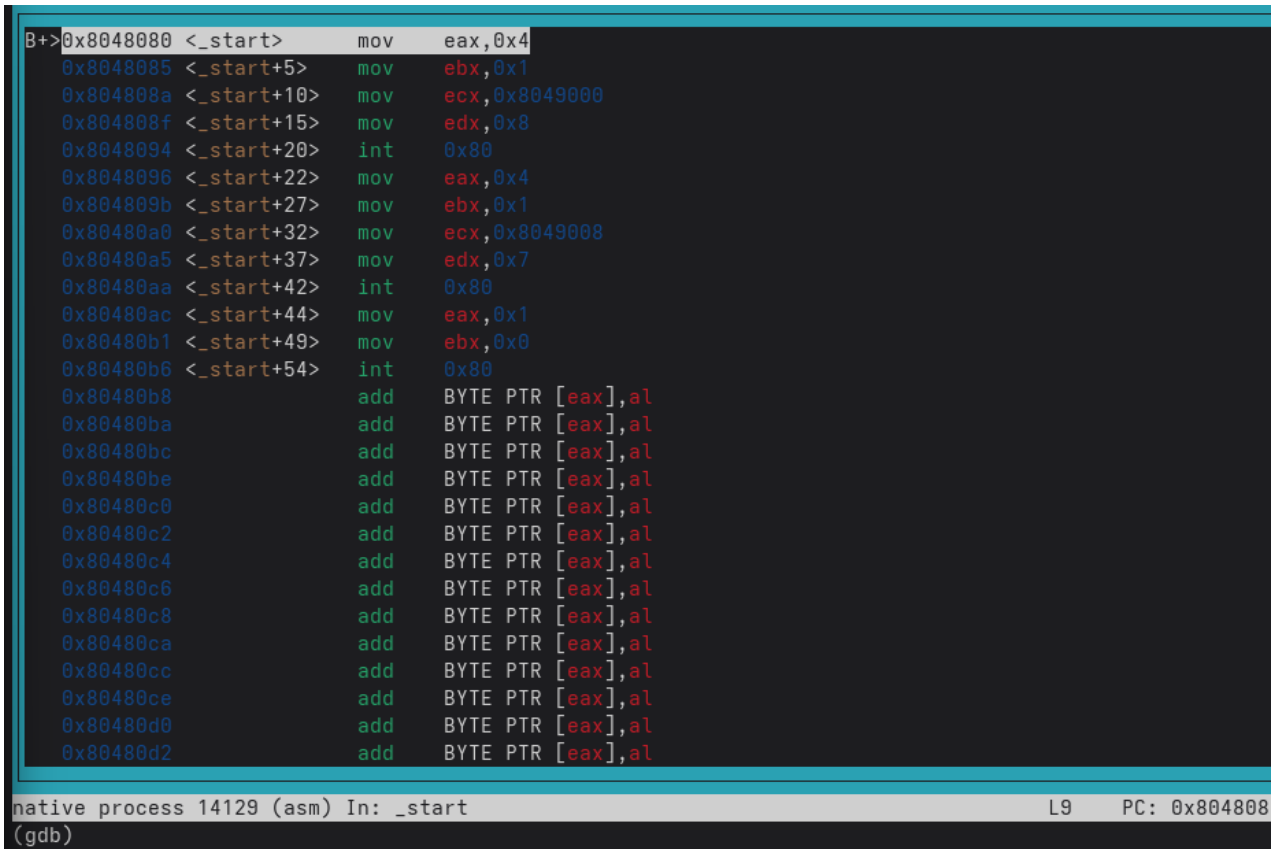


Рис 3.11: 311

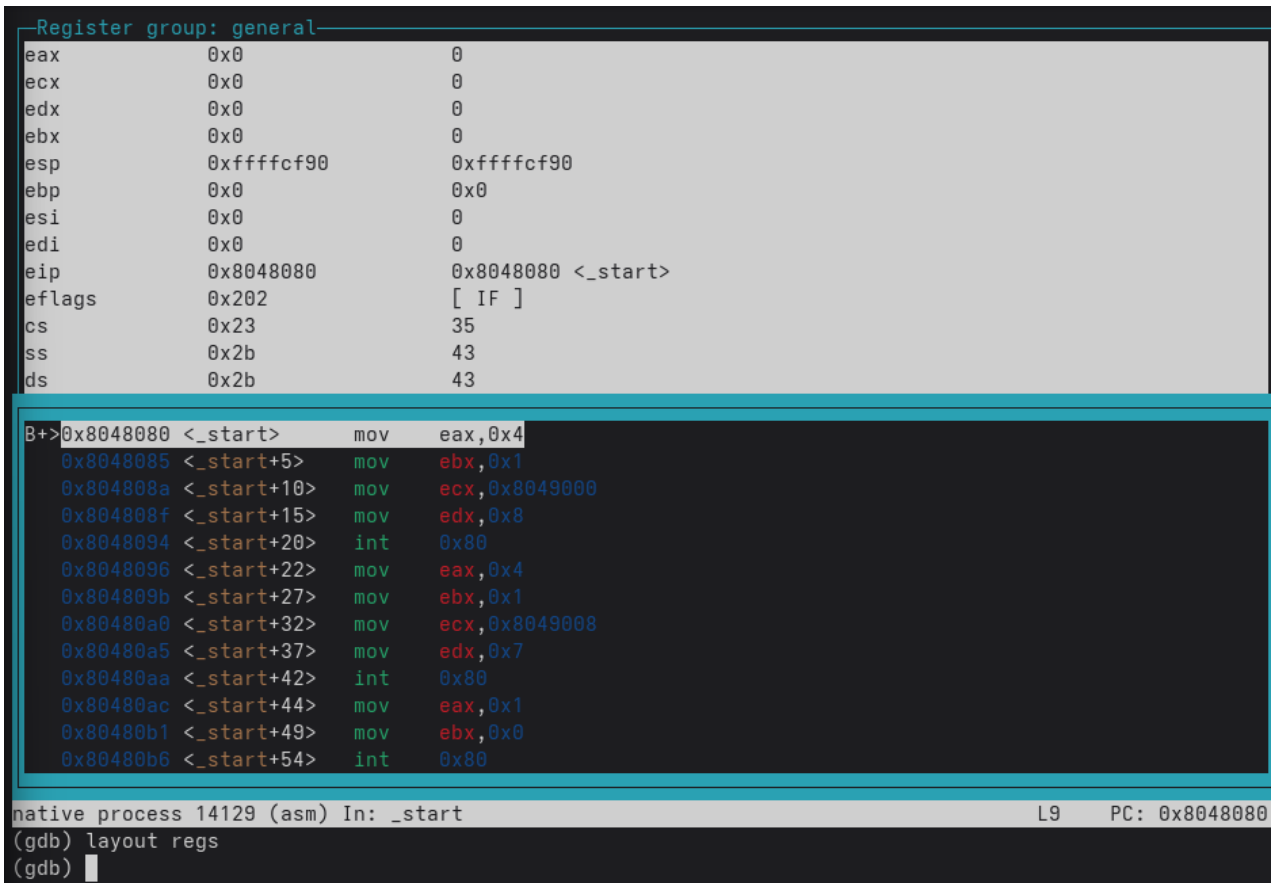


Рис 3.12: 312

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

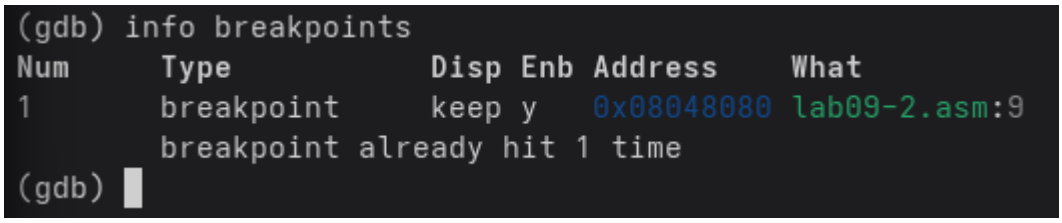


Рис 3.13: 313

Установим еще одну точку останова по адресу инструкции.

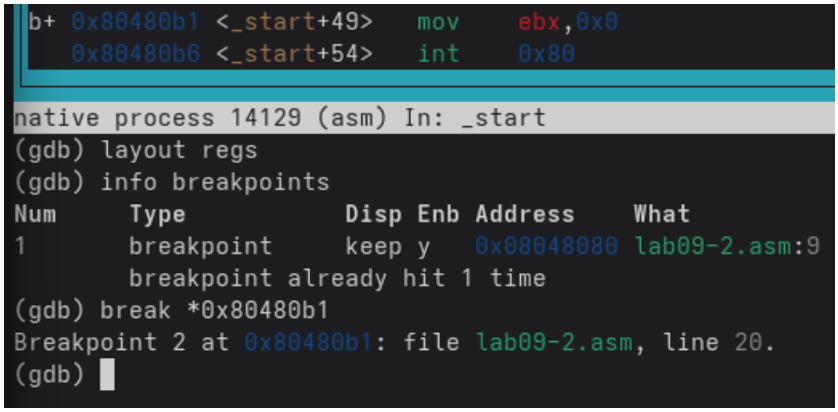


Рис 3.14: 314

Посмотрим информацию о всех установленных точках останова.

```
(gdb) i b
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x08048080 lab09-2.asm:9
        breakpoint already hit 1 time
2       breakpoint     keep y   0x080480b1 lab09-2.asm:20
(gdb)
```

Рис 3.15: 315

Выполним 5 инструкций с помощью команды stepi, отметим что изменяются значения регистров.

```
Register group: general
eax            0x8                8
ecx            0x8049000          134516736
edx            0x8                8
ebx            0x1                1
esp            0xffffcf90         0xffffcf90
ebp            0x0                0x0
esi            0x0                0
edi            0x0                0
eip            0x8048096          0x8048096 <_start+22>
eflags         0x202              [ IF ]
cs             0x23               35
ss             0x2b               43
ds             0x2b               43

B+ 0x8048080 <_start>      mov    eax,0x4
   0x8048085 <_start+5>    mov    ebx,0x1
   0x804808a <_start+10>   mov    ecx,0x8049000
   0x804808f <_start+15>   mov    edx,0x8
   0x8048094 <_start+20>   int    0x80
  >0x8048096 <_start+22>   mov    eax,0x4
   0x804809b <_start+27>   mov    ebx,0x1
   0x80480a0 <_start+32>   mov    ecx,0x8049008
   0x80480a5 <_start+37>   mov    edx,0x7
   0x80480aa <_start+42>   int    0x80
   0x80480ac <_start+44>   mov    eax,0x1
b+ 0x80480b1 <_start+49>   mov    ebx,0x0
   0x80480b6 <_start+54>   int    0x80

native process 14129 (asm) In: _start
1       breakpoint     keep y   0x08048080 lab09-2.asm:9
        breakpoint already hit 1 time
(gdb) break *0x80480b1
Breakpoint 2 at 0x80480b1: file lab09-2.asm, line 20.
(gdb) i b
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x08048080 lab09-2.asm:9
        breakpoint already hit 1 time
2       breakpoint     keep y   0x080480b1 lab09-2.asm:20
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) siHello,
```

Рис 3.16: 316

Посмотрим содержимое регистров с помощью команды info registers.

```
eax            0x8                 8
ecx            0x8049000           134516736
edx            0x8                 8
ebx            0x1                 1
esp            0xffffcf90          0xffffcf90
ebp            0x0                 0x0
esi            0x0                 0
edi            0x0                 0
eip            0x8048096           0x8048096 <_start+22>
eflags         0x202               [ IF ]
cs             0x23                35
ss             0x2b                43
ds             0x2b                43
es       Hello, 0x2b               43
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис 3.17: 317

Посмотрим значение переменной msg1 по имени.

```
(gdb) x/1sb &msg1
0x8049000 <msg1>:       "Hello, "
(gdb)
```

Рис 3.18: 318

Посмотрим значение переменной msg2 по адресу.

```
(gdb) x/1sb 0x8049008
0x8049008 <msg2>:       "world!\n\034"
(gdb)
```

Рис 3.19: 319

Изменим значение для регистра или ячейки памяти с помощью команды set. Изменим первый символ переменной msg1.

```
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x8049000 <msg1>:       "hello, "
(gdb)
```

Рис 3.20: 320

Заменим любой символ во второй переменной msg2.

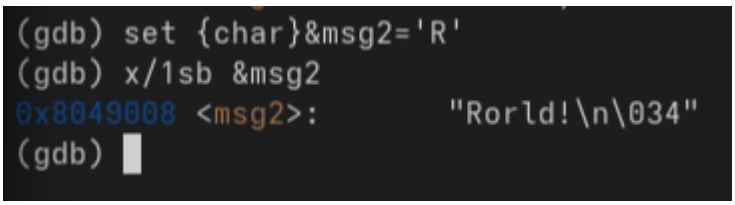


Рис 3.21: 321

Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx.

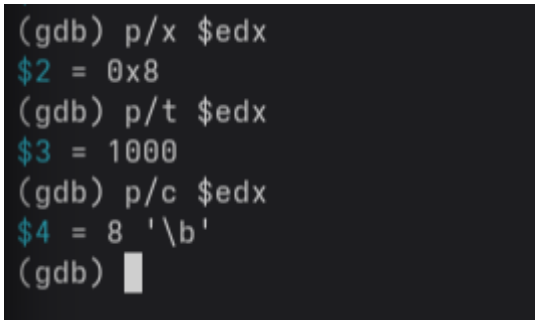


Рис 3.22: 322

С помощью команды set изменим значение регистра ebx:

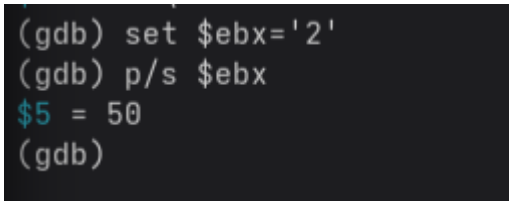


Рис 3.23: 323

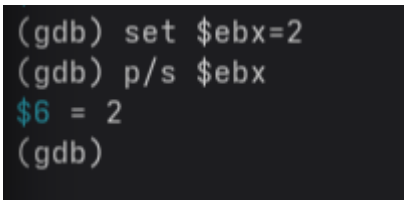


Рис 3.24: 324

p/s интерпретирует значение регистра как адрес и читает строку из памяти, в то время как p просто показывает числовое значение регистра.

Завершим выполнение программы с помощью команды continue или stepi (сокращенно si) и выйдем из GDB с помощью команды quit.

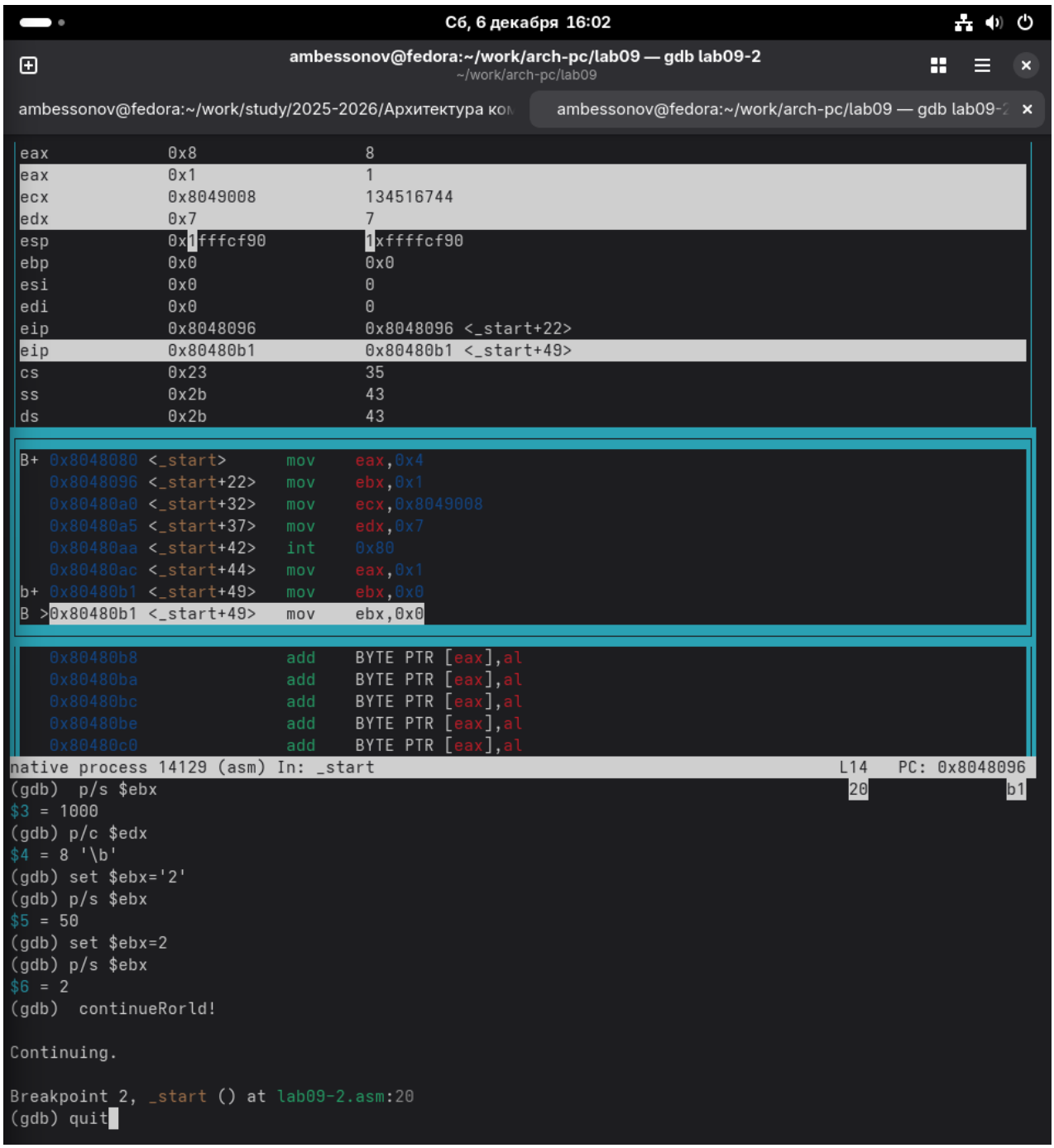


Рис 3.25: 325

Скопируем файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки (Листинг 8.2) в файл с именем lab09-3.asm.

Создадим исполняемый файл.

Загрузим исполняемый файл в отладчик, указав аргументы.

```
ambessonov@fedora:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.
asm
ambessonov@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
ambessonov@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
ambessonov@fedora:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Fedora Linux) 16.2-3.fc42
Copyright (C) 2024 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 lab09-3...
(gdb)
```

Рис 3.26: 326

Установим точку останова перед первой инструкцией в программе и запустим ее.

```
(gdb)  b _start
Breakpoint 1 at 0x8048148: file lab09-3.asm, line 8.
(gdb) run
Starting program: /home/ambessonov/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\

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

Breakpoint 1, _start () at lab09-3.asm:8
8       pop ecx ; Извлекаем из стека в `ecx` количество
(gdb)
```

Рис 3.27: 327

Посмотрим остальные позиции стека.

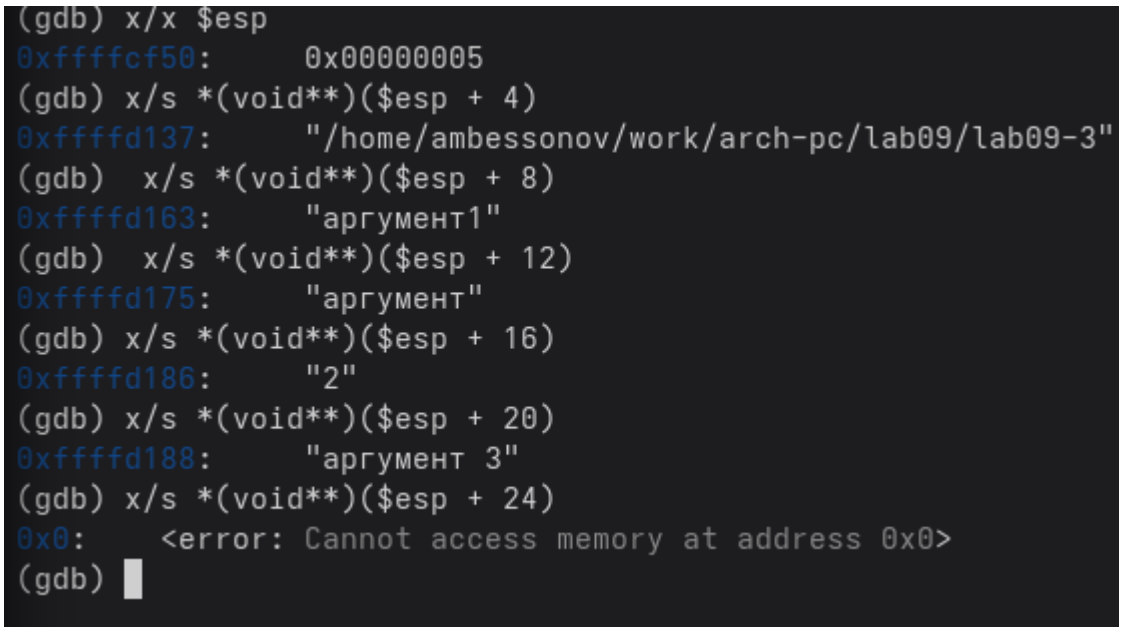


Рис 3.28: 328

Шаг в 4 байта обусловлен архитектурой процессора (32-битной) и размером указателей в ней. Каждый элемент массива argv[] занимает 4 байта в памяти, так как это адрес (указатель) на строку.

Вывод: выполнили задания лабораторной работы.

## 4. Выполнение самостоятельной работы

1. Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму.

Решение:

Преобразуем программу, реализовав вычисление значения функции $f(x)$ как подпрограмму.

```
%include 'in_out.asm'

SECTION .data
msg_func db "Функция: f(x)=6x+13",0
msg_result db "Результат: ",0

SECTION .text
global _start

_start:
    pop ecx             ; Извлекаем количество аргументов
    pop edx             ; Извлекаем имя программы
    sub ecx, 1          ; Уменьшаем количество аргументов на 1
    mov esi, 0          ; Используем esi для хранения суммы

    ; Вывод сообщения о функции
    mov eax, msg_func
    call sprintLF

next:
    cmp ecx, 0          ; Проверяем, есть ли еще аргументы
    jz _end             ; Если аргументов нет - выход

    pop eax             ; Извлекаем аргумент (x) из стека
    call atoi           ; Преобразуем в число

    ; Вычисляем f(x) = 6x + 13 через подпрограмму
    call _calculate_function

    ; Добавляем результат к общей сумме
    add esi, eax        ; esi = esi + f(x)

    dec ecx             ; Уменьшаем счетчик аргументов
    jmp next            ; Переход к следующему аргументу

_end:
    ; Вывод результата
    mov eax, msg_result
    call sprint
    mov eax, esi        ; Записываем сумму в eax
    call iprintLF       ; Печать результата
    call quit           ; Завершение программы

; ------------------------------------------------------------
; Подпрограмма вычисления функции f(x) = 6x + 13
; Вход:  eax = x (аргумент функции)
; Выход: eax = f(x) = 6x + 13
; Сохраняет: ebx (чтобы не нарушать работу основной программы)
; ------------------------------------------------------------
_calculate_function:
    push ebx            ; Сохраняем значение ebx в стеке
    mov ebx, eax        ; Сохраняем x в ebx
    imul eax, 6         ; eax = 6x
    add eax, 13         ; eax = 6x + 13
    pop ebx             ; Восстанавливаем исходное значение ebx
    ret                 ; Возврат в основную программу
```

Рис 4.1: 41

Проверим программу:

```
ambessonov@fedora:~/work/arch-pc/lab09$ nasm -f elf work.asm
ambessonov@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o work work.o
ambessonov@fedora:~/work/arch-pc/lab09$ ./work
Функция: f(x)=6x+13
Результат: 0
ambessonov@fedora:~/work/arch-pc/lab09$ ./work 21
Функция: f(x)=6x+13
Результат: 139
ambessonov@fedora:~/work/arch-pc/lab09$ ./work 2 33 34
Функция: f(x)=6x+13
Результат: 453
```

Рис 4.2: 42

2. В листинге 9.3 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.

Решение:

Создадим файл lab09-4.asm, создадим исполняемый файл и проверим с помощью GDB.

```
ambessonov@fedora:~/work/arch-pc/lab09$ touch lab09-4.asm
ambessonov@fedora:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-4.lst lab09-4.asm
ambessonov@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
ambessonov@fedora:~/work/arch-pc/lab09$ gdb lab09-4
GNU gdb (Fedora Linux) 16.2-3.fc42
Copyright (C) 2024 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 lab09-4...
```

Рис 4.3: 43

Выполним программу по шагам и проследим за регистрами.

Ошибка: инструкция mul ecx умножает значение в eax (которое равно 2) на ecx (4), получается 8. Но нам нужно умножить результат сложения (5), который хранится в ebx.

Результат: Программа выдает 10 вместо 25.

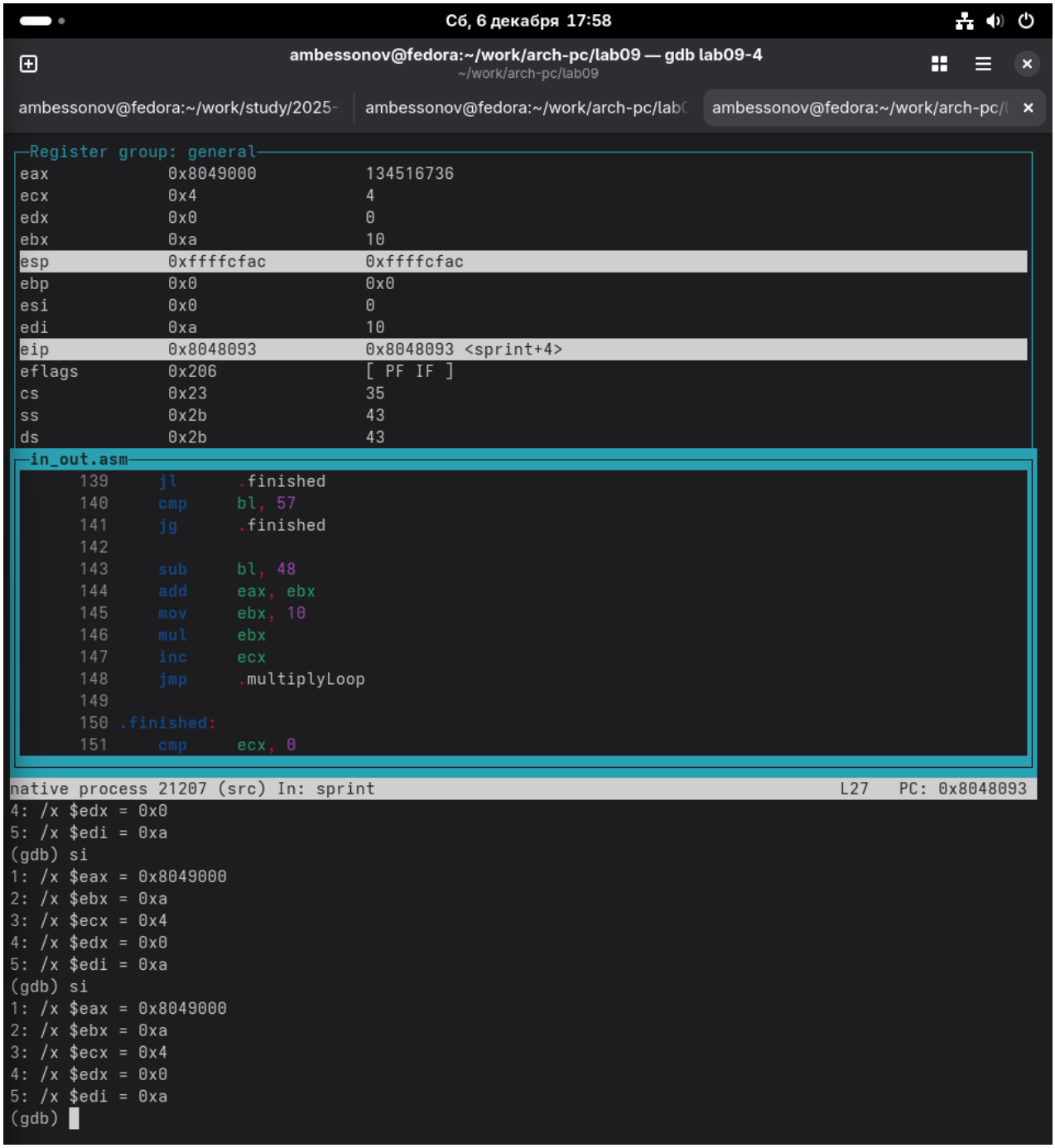


Рис 4.4: 44

Исправим программу:

Сб, 6 декабря 17:58

Открыть

lab09-4.asm
~/work/arch-pc/lab09

```
%include 'in_out.asm'

SECTION .data
    result_msg: DB 'Результат: ',0

SECTION .text
    GLOBAL _start

_start:
    ; ---- Вычисление выражения (3+2)*4+5
    mov ebx, 3          ; ebx = 3
    mov eax, 2          ; eax = 2
    add ebx, eax        ; ebx = 3 + 2 = 5

    ; Теперь нужно умножить ebx (5) на 4
    mov eax, ebx        ; eax = 5 (перемещаем результат сложения в eax)
    mov ecx, 4          ; ecx = 4
    mul ecx             ; eax = 5 * 4 = 20

    add eax, 5          ; eax = 20 + 5 = 25
    mov edi, eax        ; сохраняем результат

    ; ---- Вывод результата на экран
    mov eax, result_msg
    call sprint
    mov eax, edi
    call iprintLF

    call quit
```

Рис 4.5: 45

```
ambessonov@fedora:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
ambessonov@fedora:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-4 lab09-4.o
ambessonov@fedora:~/work/arch-pc/lab09$ ./lab09-4
Результат: 25
```

Рис 4.6: 46

В исправленной версии сначала вычисляется 3+2=5, затем 5*4=20, затем 20+5=25

Вывод: выполнили задания самостоятельной работы, отработали навыки, полученные в ходе лабораторной работы.

# 5. Выводы

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