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:Безлепкина Татьяна Игоревна

Группа:НКАбд-01-25

МОСКВА

2025г.

Оглавление

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

2.1 Реализация подпрограмм в NASM

Создам каталог для выполнения лабораторной работы № 9, перейду в него и создам файл lab09-1.asm:

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab08$ mkdir ~/work/arch-pc/lab09
tibezlepkina1@localhost-live:~/work/arch-pc/lab08$ cd ~/work/arch-pc/lab09
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ touch lab09-1.asm
```

Рисунок 2.1.1 - создание каталога,переход в него,создание файла lab09-1.asm

Далее нахожу путь к файлу «in_out.asm», копирую этот файл в каталог ~/work/arch-pc/lab09

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ find / -name "in_out.asm" 2>/dev/null
/home/tibezlepkina1/work/arch-pc/lab08/in_out.asm
/home/tibezlepkina1/work/arch-pc/lab07/in_out.asm
/home/tibezlepkina1/work/arch-pc/lab06/in_out.asm
/home/tibezlepkina1/work/arch-pc/lab05/in_out.asm
/run/overlayfs/home/tibezlepkina1/work/arch-pc/lab08/in_out.asm
/run/overlayfs/home/tibezlepkina1/work/arch-pc/lab07/in_out.asm
/run/overlayfs/home/tibezlepkina1/work/arch-pc/lab06/in_out.asm
/run/overlayfs/home/tibezlepkina1/work/arch-pc/lab05/in_out.asm
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ cp /home/tibezlepkina1/work/arch-pc/lab08/in_out.
asm ~/work/arch-pc/lab09/
```

Рисунок 2.1.2 - нахождение пути к файлу «in_out.asm»,его копирование в каталог ~/work/arch-pc/lab09

В качестве примера рассмотрю программу вычисления арифметического выражения f(x) = 2x + 7 с помощью подпрограммы _calcul. В данном примере x вводится с клавиатуры, а само выражение вычисляется в подпрограмме.

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ nano lab09-1.asm
```

Рисунок 2.1.3 - переход в текстовый редактор nano

```
 GNU nano 8.3                                    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:
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
```

Рисунок 2.1.4 - запись текста программы в файл

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1  lab09-1.o
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2x+7=17
```

Рисунок 2.1.5 - компиляция,линковка,запуск

Изменю 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. T.e. 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)). Результат возвращается в основную программу для вывода результата на экран.

```
 GNU nano 8.3                                                    lab09-1.asm
%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB 'f(g(x)) = 2*(3x-1)+7 = ',0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
    ; Ввод x
    mov eax, msg
    call sprint
    mov ecx, x
    mov edx, 80
    call sread

    ; Преобразование строки в число
    mov eax, x
    call atoi        ; результат в eax

    ; Вызов главной подпрограммы
    call _calcul     ; вызов _calcul с x в eax

    ; Вывод результата
    mov eax, result
    call sprint
    mov eax, [res]   ; берем результат из переменной res
    call iprintLF

    call quit
```

Рисунок 2.1.6 - изменение теста программы

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1  lab09-1.o
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
f(g(x)) = 2*(3x-1)+7 = 35
```

Рисунок 2.1.7 - компиляция,линковка,запуск

2.2 Отладка программам с помощью GDB

Создам файл lab09-2.asm

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ touch lab09-2.asm
```

Рисунок 2.2.1 - создание файла lab09-2.asm

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ nano lab09-2.asm
```

Рисунок 2.2.2 - переход в текстовый редактор nano

```
 GNU nano 8.3                                          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
```

Рисунок 2.2.3 - запись текста программы в файл

Получу исполняемый файл.Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’.

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ gdb lab09-2
```

Рисунок 2.2.4 -компиляция,линковка,запуск отладчика

Проверю работу программы, запустив ее в оболочке GDB с помощью команды run (сокращённо r):

```
(gdb) run
Starting program: /home/tibezlepkina1/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 51.96 K separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 56747) exited normally]
```

Рисунок 2.2.5 - проверка работы с помощью команды run

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

```
(gdb) break _start
Breakpoint 1 at 0x8048080: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/tibezlepkina1/work/arch-pc/lab09/lab09-2

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

Рисунок 2.2.6 - установлю брейкпоинт на метку _start,запущу ее

Посмотрю дизассемблированный код программы с помощью команды disassemble начиная с метки _start

```
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08048080 <+0>:     mov    $0x4,%eax
   0x08048085 <+5>:     mov    $0x1,%ebx
   0x0804808a <+10>:    mov    $0x8049000,%ecx
   0x0804808f <+15>:    mov    $0x8,%edx
   0x08048094 <+20>:    int    $0x80
   0x08048096 <+22>:    mov    $0x4,%eax
   0x0804809b <+27>:    mov    $0x1,%ebx
   0x080480a0 <+32>:    mov    $0x8049008,%ecx
   0x080480a5 <+37>:    mov    $0x7,%edx
   0x080480aa <+42>:    int    $0x80
   0x080480ac <+44>:    mov    $0x1,%eax
   0x080480b1 <+49>:    mov    $0x0,%ebx
   0x080480b6 <+54>:    int    $0x80
End of assembler dump.
```

Рисунок 2.2.7 - просмотр дизасемблированного кода программы

Переключусь на отображение команд с Intel’овским синтаксисом, введя команду set disassembly-flavor intel

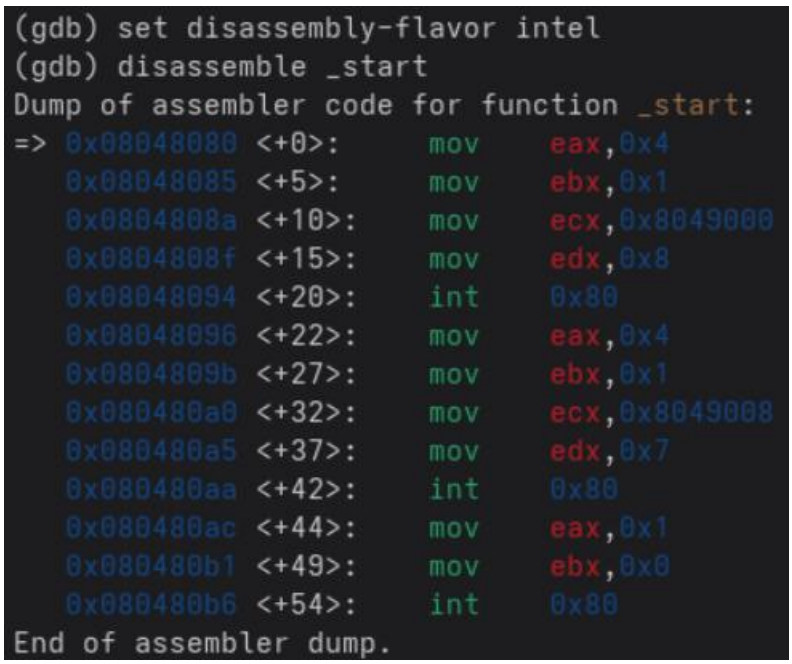

Рисунок 2.2.8 - просмотр дизассемблированного кода программы с синтаксисом отображения ассемблерного кода в формате Intel

Перечислю различия отображения синтаксиса машинных команд в режимах ATT и Intel. Включу режим псевдографики для более удобного анализа программы.

Различия:

-Порядок операндов

-Символы перед операндами

-Синтаксис прерываний

2.2.1 Добавление точек останова

На предыдущих шагах была установлена точка останова по имени метки (_start).

Проверю это с помощью команды info breakpoints (кратко i b)

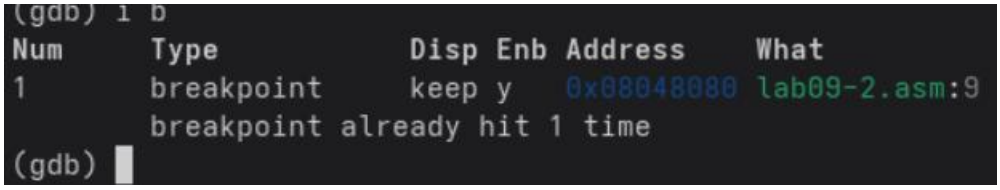

Рисунок 2.2.1.1 - проверка установки точки остановы по имени метки (_start)

Установлю еще одну точку останова по адресу инструкции.

Рисунок 2.2.1.2 - установка точки останова

Посмотрю 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
```

Рисунок 2.2.1.3 - просмотр информации о всех установленных точек останова

2.2.2 Работа с данными программы в GDB.

Выполню 5 инструкций с помощью команды stepi (или si) и прослежу за изменением значений регистров.Посмотрю содержимое регистров с помощью команды info registers (или i r).Для этого напишу 5 раз подряд:

(gdb)si

(gdb) info registers

В итоге:

1) Регистр EAX изменился с 0x00000000 на 0x00000004

2) Регистр EBX изменился с 0x00000000 на 0x00000001

3) Регистр ECX изменился с 0x00000000 на 0x08049000

4)Регистр EDX изменился с 0x00000000 на 0x00000008

5)Регистр EAX изменился на 0x00000008 (результат системного вызова)

Посмотрю значение переменной msg1 по имени:

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

Рисунок 2.2.2.1 - просмотр значения переменной по имени

Посмотрю значение переменной msg2 по адресу:

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

Рисунок 2.2.2.2 - просмотр значения переменной по адресу

Изменю первый символ переменной msg1

Изменить значение для регистра или ячейки памяти можно с помощью команды set, задав ей в качестве аргумента имя регистра или адрес. При этом перед именем регистра ставится префикс $, а перед адресом нужно указать в фигурных скобках тип данных (размер сохраняемого значения; в качестве типа данных можно использовать типы языка Си).

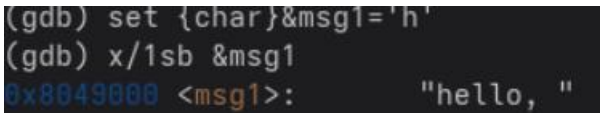

Рисунок 2.2.2.3 - изменение содержимого переменной

Заменю любой символ во второй переменной msg2:

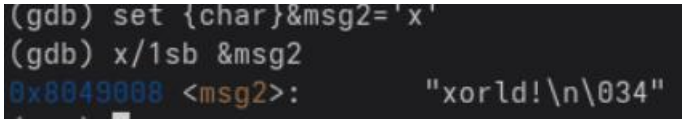

Рисунок 2.2.2.4 - изменение содержимого переменной

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

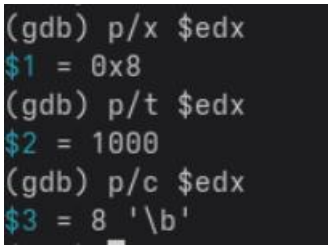

Рисунок 2.2.2.5 - вывод в различных форматах значение регистра edx

С помощью команды set изменю значение регистра ebx:

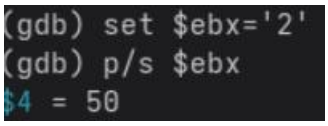

Рисунок 2.2.2.6 - изменение регистра ebx с помощью команды set

Рисунок 2.2.2.7 - пример использования команды set

Разница вывода команд p/s $ebx заключается в выводимых числах.

set $ebx=2 = "сделай EBX равным математическому числу 2"

set $ebx='2' = "сделай EBX равным коду символа '2' в ASCII (который равен 50)"

Рисунок 2.2.2.8 - выход из GDB

2.2.3 Обработка аргументов командной строки в GDB

Скопирую 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

```
(gdb) layout asm
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$
```

Рисунок 2.2.3.1 - копирование файла alb8-2.asm в lab09-3.asm

Создам исполняемый файл.

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-3 lab09-3.o
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$
```

Рисунок 2.2.3.2 - компиляция,линковка

Для загрузки в gdb программы с аргументами необходимо использовать ключ --args. Загружу исполняемый файл в отладчик, указав аргументы.

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ gdb --args lab09-3 аргумент1 аргумент 2 'аргум
ент 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)
```

Рисунок 2.2.3.3 - загрузка исполняемого файла в отладчик

В 8 лабораторной работе, при запуске программы аргументы командной строки загружаются в стек. Исследую расположение аргументов командной строки в стеке после запуска программы с помощью gdb. Для начала установлю точку останова перед первой инструкцией в программе и запущу ее.

```
(gdb) b _start
Breakpoint 1 at 0x8048148: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/tibezlepkina1/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3

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
```

Рисунок 2.2.3.4 - установление точки останова перед первой инструкцией в программе и ее запуск

Адрес вершины стека храниться в регистре esp и по этому адресу располагается

число равное количеству аргументов командной строки (включая имя программы):

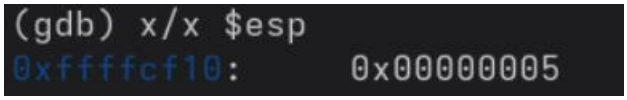

Рисунок 2.2.3.5 - просмотр содержимого памяти по адресу в регистре ESP

Число аргументов равно 5 – это имя программы lab09-3 и аргументы: аргумент1, аргумент, 2 и 'аргумент 3'.

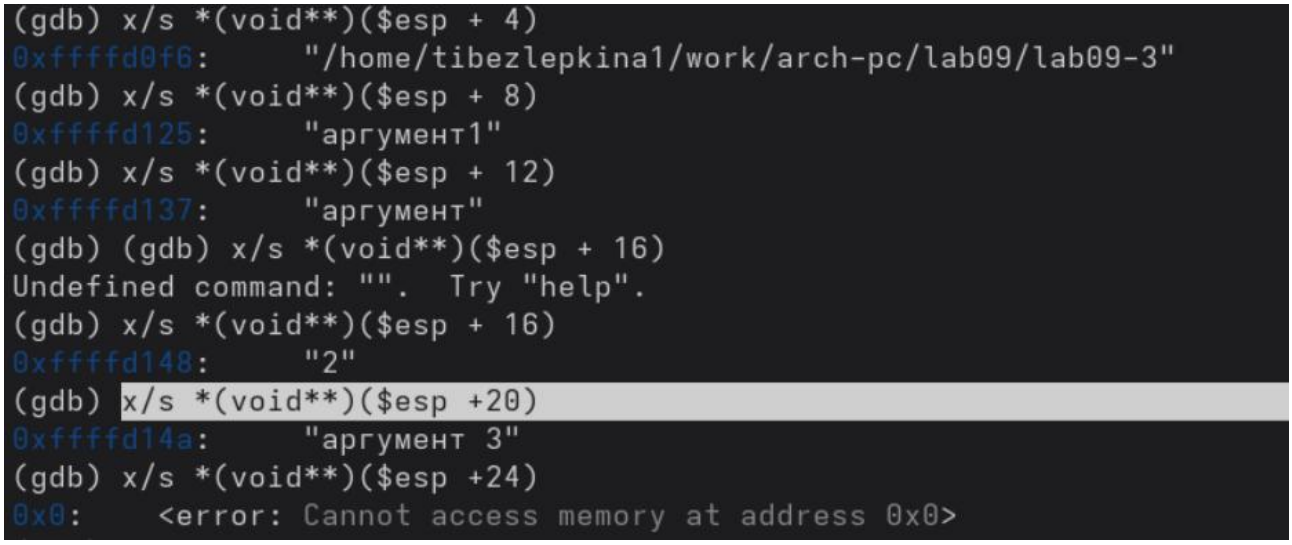

Рисунок 2.2.3.6 - просмотр позиций стека

по адесу [esp+4] располагается адрес в памяти где находиться имя программы, по адесу [esp+8] храниться адрес первого аргумента, по аресу [esp+12] – второго и т.д.Шаг 4 байта обусловлен фундаментальным свойством 32-битной архитектуры x86, где размер адреса (указателя) равен 4 байтам. Каждый элемент массива аргументов argv[] является указателем на строку и занимает ровно 4 байта в памяти, поэтому для доступа к следующему аргументу необходимо увеличивать адрес на 4.

3 Задания для самостоятельной работы

1 задание

Преобразую программу из лабораторной работы №8 (Задание №1 для самостоятельной работы), реализовав вычисление значения функции f(x) как подпрограмму.Для начала создам файл lab09-4.asm и перейду в текстовый редактор nano.

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ touch lab09-4.asm
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ nano lab09-4.asm
```

Рисунок 3.1 - создание файла lab09-4.asm и переход в текстовый редактор nano

```
 GNU nano 8.3                                  lab09-4.asm
%include 'in_out.asm'

SECTION .data
msg1 db 'Функция: f(x)=3(10+x)',0
msg2 db 'Результат: ',0
SECTION .bss
res resb 10

SECTION .text
global _start

; Подпрограмма f(x) = 3(10+x)
_fx:
    add eax, 10    ; x + 10
    mov ebx, 3     ; умножаем на 3
    mul ebx        ; 3 * (x+10)
    ret

_start:
    ; Выводим что вычисляем
    mov eax, msg1
```

Рисунок 3.2 - запись текста программы в файл

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ nasm -f elf lab09-4.asm
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$  ld -m elf_i386 lab09-4.o -o lab09-4
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ ./lab09-4 1 2 3 4
Функция: f(x)=3(10+x)
Результат: 42
```

Рисунок 3.3 - компиляция,линковка,запуск

2 задание

Создам файл lab09-5.asm для выполнения 2 задания.Перейду в текстовый редактор.

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ touch lab09-5.asm
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ nano lab09-5.asm
```

Рисунок 3.4 - создание файла lab09-5.asm и переход в текстовый редактор nano

```
 GNU nano 8.3                                  lab09-5.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
```

Рисунок 3.5 - запись текста программы в файл

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ ld -m elf_i386 lab09-5.o -o lab09-5
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ ./lab09-5
Результат: 10
```

Рисунок 3.6 - компиляция,линковка,запуск

Как итог возникает ошибка,так как (3+2)*4+5 не равно 10.С помощью отладчика GDB, анализируя изменения значений регистров, определю ошибку и исправлю ее.

-Запускаю программу в режиме отладчика,

gdb lab09-5

-Устанавливаю Intel-синтаксис (как в NASM)

(gdb) set disassembly-flavor intel

- Включаю графический режим для наглядности

(gdb) layout regs

-Ставлю точку останова на начало программы

(gdb) break _start

-Запускаем программу

(gdb) run

Пошагово через si + info registers просматриваю изменение значений регистров,через 5 инструкций замечаю,что ошибка в строке mul ecx - умножается не то значение!Результат вычисления 3+2=5 остаётся в EBX, но команда mul использует для умножения значение из EAX, где до сих пор лежит старая двойка. Нужна дополнительная инструкция mov eax,ebx для переноса значения перед умножением.

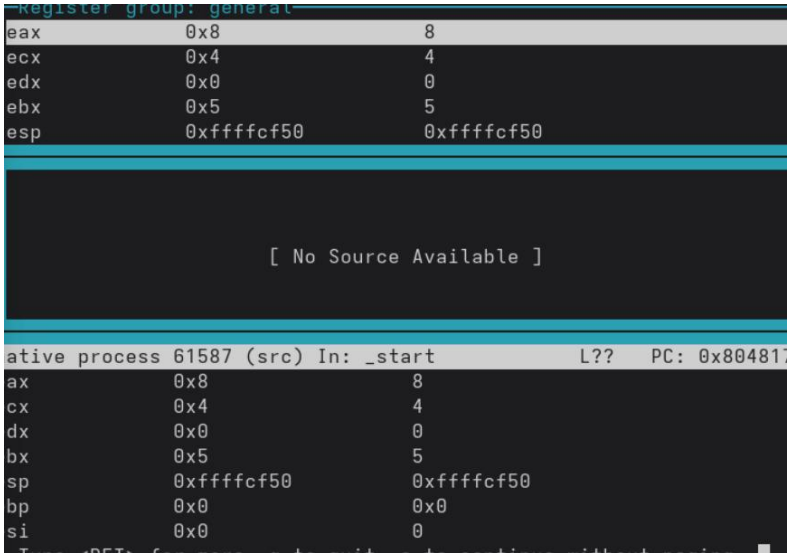

Рисунок 3.7 - ошибка в строке mul ecx - умножается не то значение

Через 6 инструкций замечаю,что EBX = 5 + 5 = 10 (ОШИБКА!)

Должно быть: 20 + 5 = 25.Вторая ошибка add ebx,5 неправильно выбирает регистр для сложения. После умножения результат всегда в EAX, поэтому и прибавлять нужно к EAX, а не к EBX.

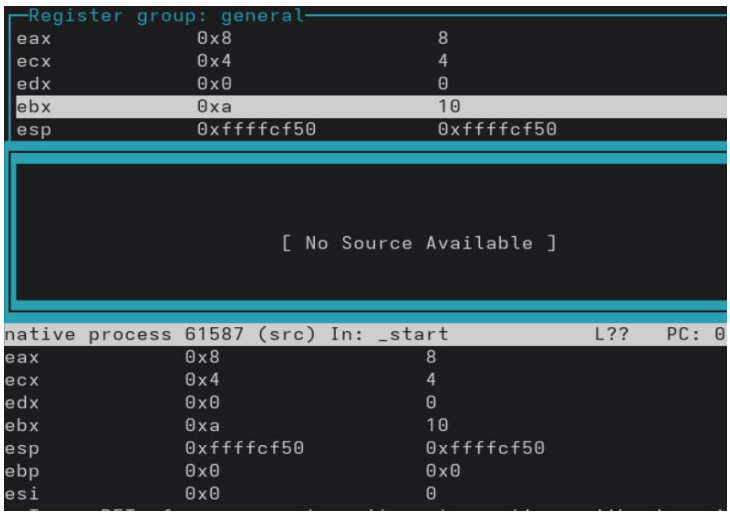

Рисунок 3.8 - ошибка,складываем не с тем регистром(add ebx,5)

Завершаю выполнение с помощью continue.

Исправлю ошибки.

```
 GNU nano 8.3                    lab09-5.asm                         Modified
%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     ; EBX = 3+2 = 5
mov eax,ebx     ; ← ДОБАВЛЕНО: переносим 5 в EAX для умножения
mov ecx,4
mul ecx         ; EAX = 5*4 = 20
add eax,5       ; ← ИСПРАВЛЕНО: было add ebx,5
mov edi,eax     ; ← ИСПРАВЛЕНО: было mov edi,ebx

; ---- Вывод результата на экран
mov eax,div
call sprint
```

Рисунок 3.9 - изменение текста программы

```
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ nasm -f elf lab09-5.asm
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ ld -m elf_i386 lab09-5.o -o lab09-5
tibezlepkina1@localhost-live:~/work/arch-pc/lab09$ ./lab09-5
Результат: 25
```

Рисунок 3.10 компиляция,линковка,запуск

Теперь вывод верный.

4 Вывод

В результате выполнения лабораторной работы я не только приобрела навыки написания программы с использованием подпрограмм,но и познакомилась с методами отладки GDB,его основными возможностями.