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: Сачковская София Александровна

Группа: НКАбд-06-25

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

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.

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 программаотладчик 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.

Основные возможности отладчика GDB

GDB (GNU Debugger — отладчик проекта GNU) [1] работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ. Отладчик не содержит собственного графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Однако для GDB существует несколько сторонних графических надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки. Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя. GDB может выполнять следующие действия:

• начать выполнение программы, задав всё, что может повлиять на её поведение;

• остановить программу при указанных условиях;

• исследовать, что случилось, когда программа остановилась;

• изменить программу так, чтобы можно было поэкспериментировать с устранением эффектов одной ошибки и продолжить выявление других.

Запуск отладчика GDB; выполнение программы; выход

Синтаксис команды для запуска отладчика имеет следующий вид:

gdb [опции] [имя_файла | ID процесса]

После запуска gdb выводит текстовое сообщение — так называемое «nice GDB logo». В следующей строке появляется приглашение (gdb) для ввода команд. Далее приведён список некоторых команд GDB. Команда run (сокращённо r) — запускает отлаживаемую программу в оболочке GDB. Если точки останова не были установлены, то программа выполняется и выводятся сообщения: (gdb) run Starting program: test Program exited normally. (gdb) Если точки останова были

заданы, то отладчик останавливается на соответствующей команде и выдаёт номер точки останова, адрес и дополнительную информацию — текущую строку, имя процедуры, и др. Команда kill (сокращённо k) прекращает отладку программы, после чего следует вопрос о прекращении процесса отладки: Kill the program being debugged? (y or n) y Если в ответ введено y (то есть «да»), отладка программы прекращается. Командой run её можно начать заново, при этом все точки останова (breakpoints), точки просмотра (watchpoints) и точки отлова (catchpoints) сохраняются. Для выхода из отладчика используется команда quit (или сокращённо q): (gdb) q

Если есть файл с исходным текстом программы, а в исполняемый файл включена информация о номерах строк исходного кода, то программу можно отлаживать, работая в отладчике непосредственно с её исходным текстом. Чтобы программу можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом -g. Посмотреть дизассемблированный код программы можно с помощью команды

disassemble <метка/адрес>: (gdb) disassemble _start

Существует два режима отображения синтаксиса машинных команд: режим Intel, используемый в том числе в NASM, и режим ATT (значительно отличающийся внешне). По умолчанию в дизассемблере GDB принят режим ATT. Переключиться на отображение команд с привычным Intel’овским синтаксисом можно, введя команду

set disassembly-flavor intel

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.

Инструкция call и инструкция ret

Для вызова подпрограммы из основной программы используется инструкция call, которая заносит адрес следующей инструкции в стек и загружает в регистр eip адрес соответствующей подпрограммы, осуществляя таким образом переход. Затем начинается выполнение подпрограммы, которая, в свою очередь, также может содержать подпрограммы. Подпрограмма завершается инструкцией ret, которая извлекает из стека адрес, занесённый туда соответствующей инструкцией call, и заносит его в eip. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией call.

Подпрограмма может вызываться как из внешнего файла, так и быть частью основной программы.

Важно помнить, что если в подпрограмме занести что-то в стек и не извлечь,то на вершине стека окажется не адрес возврата и это приведёт к ошибке выхода из подпрограммы. Кроме того, надо помнить, что подпрограмма без команды возврата не вернётся в точку вызова, а будет выполнять следующий за подпрограммой код, как будто он является её продолжением.

3.Выполнение лабораторной работы

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

```
sachkovskayasofiya@sachkovskayasofiya:~$ mkdir ~/work/arch-pc/lab09
sachkovskayasofiya@sachkovskayasofiya:~$ cd ~/work/arch-pc/lab09
```

```
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab09$ touch lab09-1.asm
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab09$
```

Рис.3.1

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

```
/home/sachkovskayasofiya/work/arch-pc/lab09/
%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.3.2

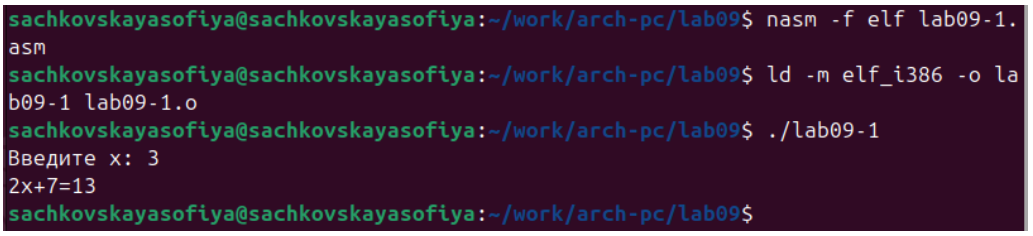

Рис.3.3

Изменим текст программы, добавив подпрограмму _subcalcul в подпрограмму _calcul, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$.

```
/home/sa~09-1.asm   [----]  9 L:[  1
%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:
;-------------------------------------
; Основная программа
;-------------------------------------
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

_subcalcul:
mov ebx,3
mul ebx
sub eax,1
ret
_calcul:
push eax
call _subcalcul

mov ebx,2
mul ebx
add eax,7
```

Рис.3.4

```
mov ebx,2
mul ebx
add eax,7

mov [res],eax
pop eax
ret

 1Помощь 2Сох~ть 3Блок   4Замена 5Ко
```

Рис.3.5

```
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab09$ nasm -f elf lab09-1.
asm
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab09$ ld -m elf_i386 -o la
b09-1 lab09-1.o
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 3
f(g(x))=2(3x-1)+7=23
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab09$
```

Рис.3.6

Создадим файл lab09-2.asm с текстом программы из Листинга 9.2. (Программа печати сообщения Hello world!):

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

Рис.3.7

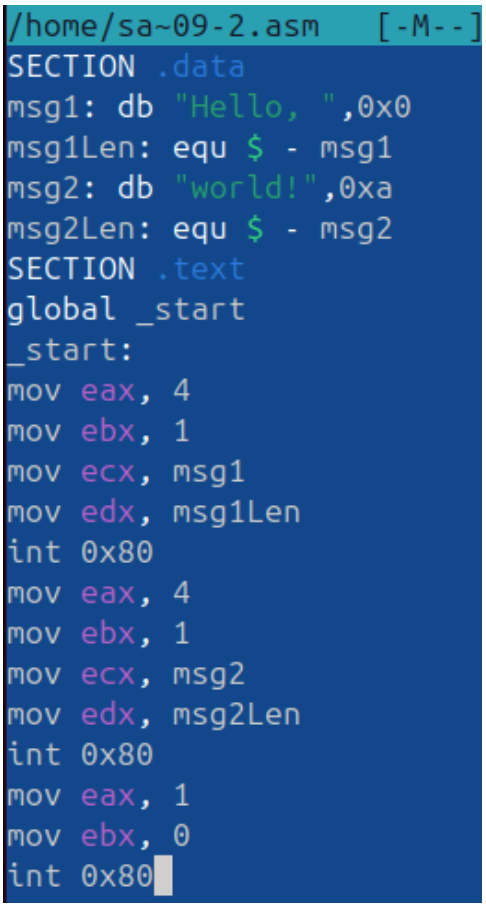

Рис.3.8

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

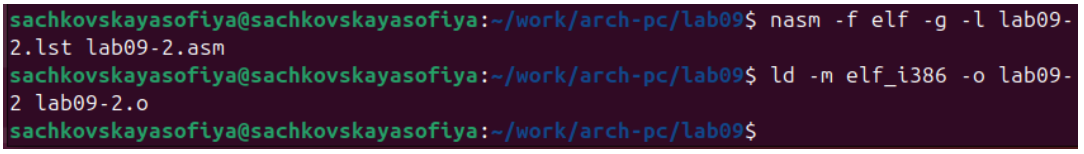

Рис.3.9

Загрузим исполняемый файл в отладчик gdb:

```
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
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-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)
```

Рис.3.10

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

```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
(gdb) run
Starting program: /home/sachkovskayasofiya/work/arch-pc/lab09/lab09-2

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
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 10411) exited normally]
(gdb)
```

Рис.3.11

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 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/sachkovskayasofiya/work/arch-pc/lab09/lab09-2

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

Рис.3.12

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

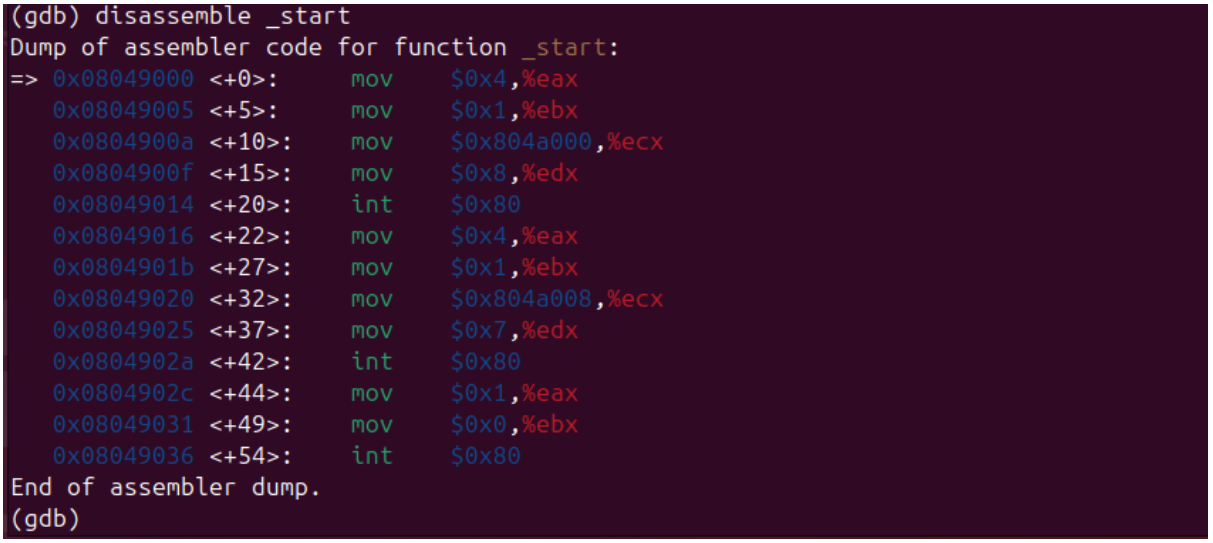

Рис.3.13

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

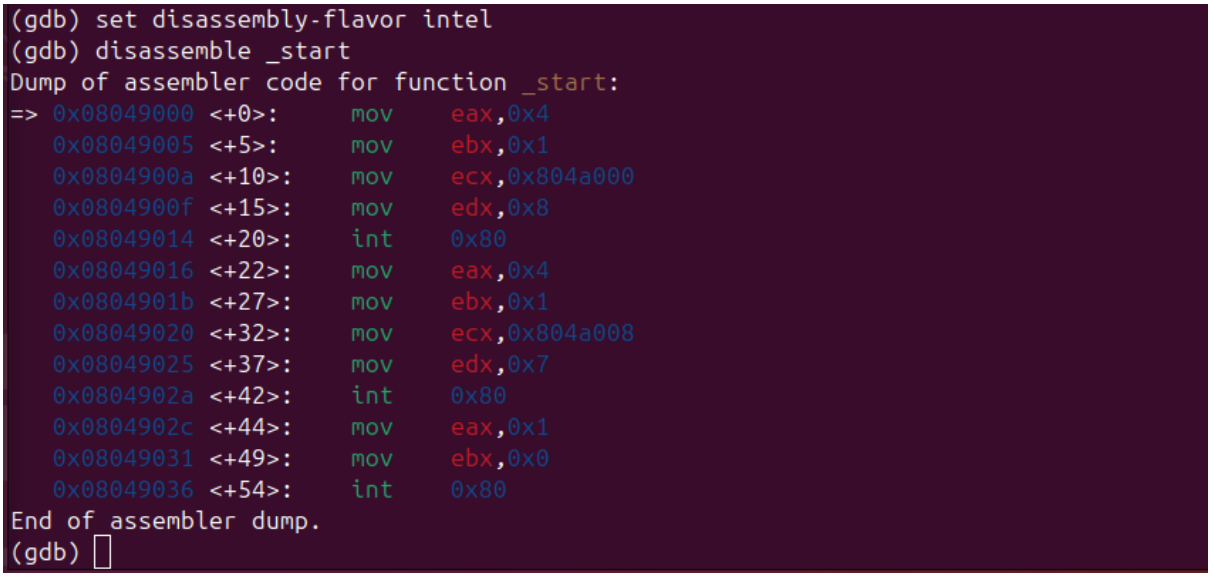

Рис.3.14

Перечислите различия отображения синтаксиса машинных команд в режимах ATT и Intel:

Ответ:

Основное различие между синтаксисом AT&T и Intel в GDB заключается в порядке операндов: в Intel используется порядок "назначение, источник" (например, mov eax, 4), тогда как в AT&T — "источник, назначение" (например, mov $4, %eax). Кроме того, в AT&T регистры обозначаются с префиксом %, а непосредственные значения с префиксом $, в то время как в Intel префиксы не используются. Обращение к памяти также отличается: в Intel используются квадратные скобки, а в AT&T — нет.

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

```
(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.3.15

Установим еще одну точку останова по адресу инструкции.Определим адрес предпоследней инструкции (mov ebx,0x0) и установим точку останова.

```
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb)
```

Рис.3.16

Посмотрим информацию о всех установленных точках останова:

```
(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.3.17

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Выполним 5 инструкций с помощью команды stepi (или si) и проследим за изменением значений регистров. Значения каких регистров изменяются?

Рис.3.18

Ответ:Изменяются значения регистров:eax,ebx,ecx,edx

С помощью команды x &<имя переменной> также можно посмотреть содержимое переменной. Посмотрим значение переменной msg1 по имени

Рис.3.19

Посмотрим значение переменной msg2 по адресу

Рис.3.20

Изменим первый символ переменной msg1

Рис.3.21

Заменим любой символ во второй переменной msg2

Рис.3.22

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

Рис.3.23

С помощью команды set изменим значение регистра ebx:

Рис.3.24

Рис.3.25

Объясните разницу вывода команд p/s $ebx.

Ответ:В первом случае в регистр ebx записывается ASCII код символа 2,а во втором случае в регистр ebx записывается числовое значение 2

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

Рис.3.26

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:

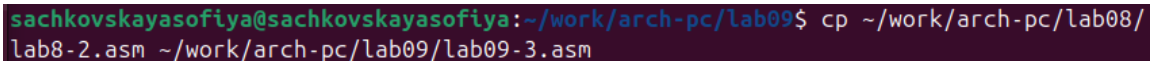

Рис.3.27

Создадим исполняемый файл.

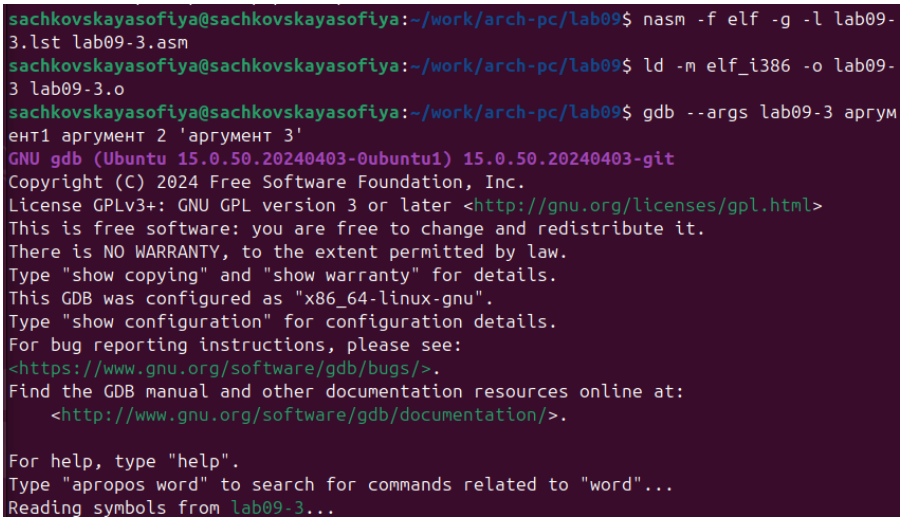

Рис.3.28

Загрузим исполняемый файл в отладчик, указав аргументы:

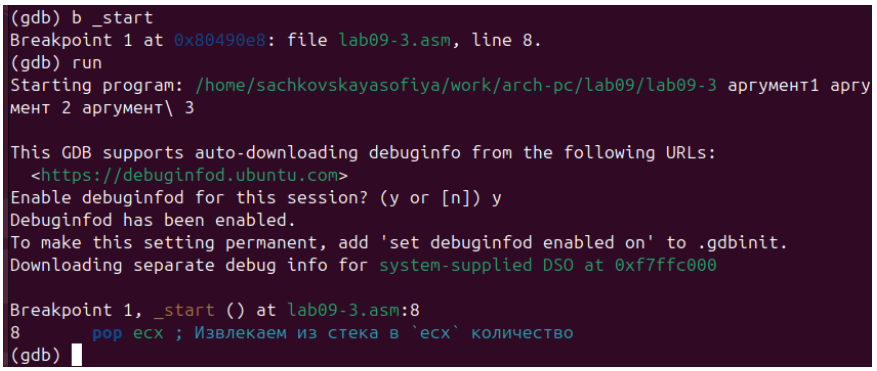

Рис.3.29

Посмотрим остальные позиции 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

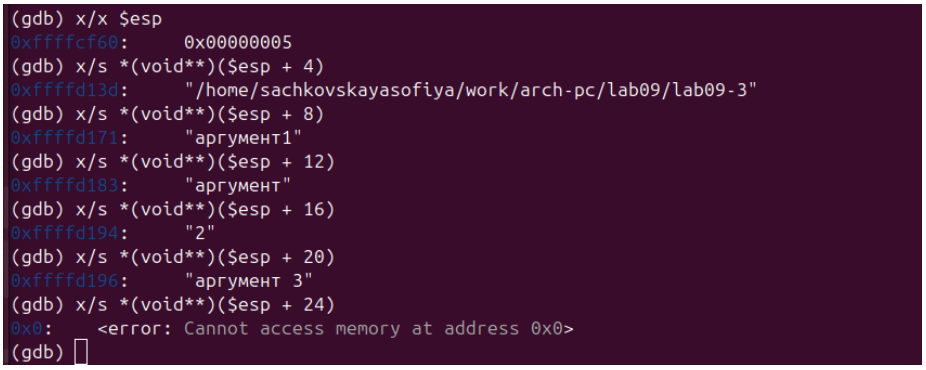

Рис.3.30

Объясните, почему шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] и т.д.).

Ответ:Так как в архитектуре x86 стандартный размер адреса составляет 32 бита,то его размер составляет 4 байта. Поскольку каждый элемент занимает ровно 4 байта, чтобы перейти к следующему элементу, необходимо увеличить смещение на 4.

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

Ответ:

```
/home/sach~9/main.asm   [----]  0 L:[  1+31
%include 'in_out.asm'

SECTION .data
msg: DB 'Введите x: ', 0
res_msg: DB 'f(x) = (x+1)*7. Результат: ', 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 _calc_func.
....
    mov [res], eax..
....
    mov eax, res_msg
    call sprint
    mov eax, [res]
    call iprintLF
    call quit

_calc_func:
    add eax, 1...
    mov ebx, 7...
    mul ebx......
....
    ret
```

Рис.4.1

```
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab09$ nasm -f elf main.asm
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab09$ ld -m elf_i386 -o main m
ain.o
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab09$ ./main
Введите x: 10
f(x) = (x+1)*7. Результат: 77
sachkovskayasofiya@sachkovskayasofiya:~/work/arch-pc/lab09$
```

Рис.4.2

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

Ответ:

Создадим файл lab09-4.asm и проверим его на наличие ошибок с помощью GDB

```
/home/sach~ab09-4.asm   [----]  9 L:[  1
%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.4.3

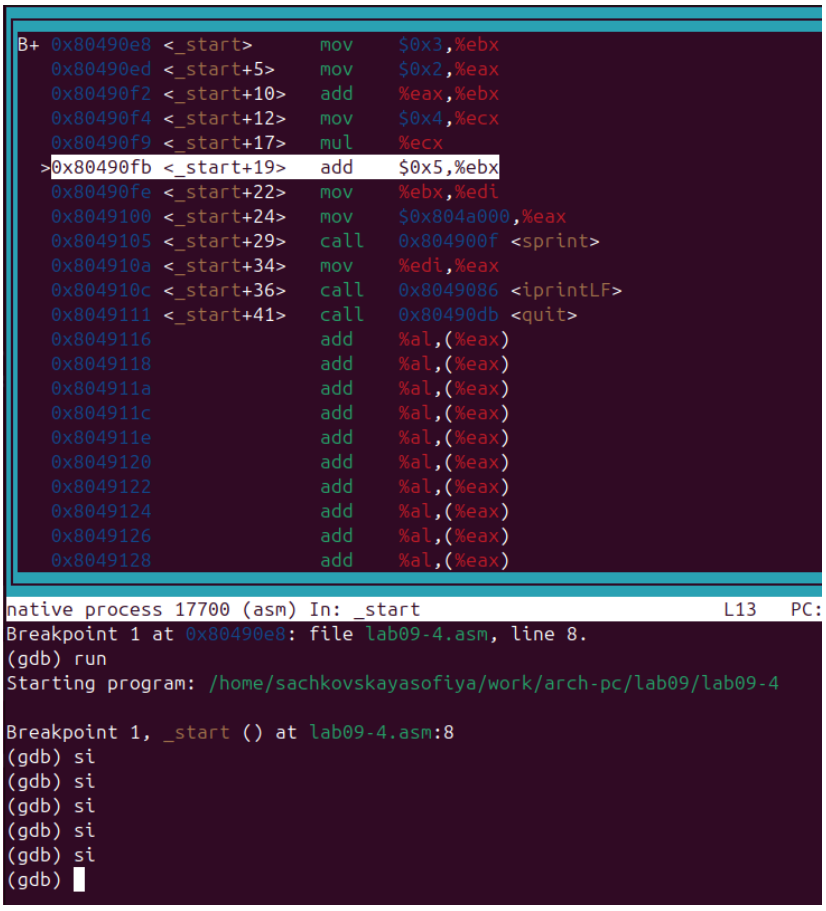

Рис.4.4

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.

Исправим программу

```
/home/sach~ab09-4.asm   [-M--] 13 L:[  1+29  30/ 30] *(786 / 78
%include 'in_out.asm'

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

SECTION .text
GLOBAL _start ; Обязательное объявление глобальной точки входа

_start:         ; <-- Метка начала выполнения программы
    ; Вычисление выражения (3+2)*4+5 = 25
    mov ebx, 3
    mov eax, 2
    add ebx, eax     ; EBX = 5
....
    mov eax, ebx     ; ИСПРАВЛЕНИЕ: Перенос 5 в EAX для MUL
....
    mov ecx, 4
    mul ecx          ; EAX = 5 * 4 = 20
....
    add eax, 5       ; EAX = 20 + 5 = 25
....
    mov edi, eax     ; Сохраняем результат в EDI

    ; Вывод результата
    mov eax, div
    call sprint
    mov eax, edi
    call iprintLF
....
    call quit
```

Рис.4.5

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

Рис.4.6

Все работает исправно

5.Выводы

Я приобрела навыки написания программ с использованием подпрограмм. Познакомилась с методами отладки при помощи GDB и его основными возможностями.