

Отчёт по лабораторной работе №9

Дисциплина: Архитектура компьютера

Буриева Шахзода Акмаловна

Содержание

1	Цель работы	6
2	Теоретическое введение	7
3	Выполнение лабораторной работы	10
4	Выполнение заданий для самостоятельной работы	21
5	Выводы	26
	Список литературы	27

Список иллюстраций

3.1	Создание каталога lab9, переход в него и создание файла lab09-1.asm	10
3.2	Ввод текста программы в файл lab09-1.asm	10
3.3	Создание исполняемого файла и его запуск	11
3.4	Изменение текста программы lab09-1.asm	11
3.5	Создание исполняемого файла и его запуск	11
3.6	Создание файла lab09-2.asm	12
3.7	Ввод текста программы в файл lab09-2.asm	12
3.8	Создание исполняемого файла и его запуск	12
3.9	Создание исполняемого файла для работы с GDB и его загрузка .	13
3.10	Проверка работы программы с помощью команды r	13
3.11	Установка брейкпоинта для подробного анализа программы . . .	13
3.12	Просмотр дисассемблированного кода программы	14
3.13	Переключение на отображение команд с Intel'овским синтаксисом	14
3.14	Включение режима псевдографики с командой layout asm	15
3.15	Включение режима псевдографики с командой layout regs	15
3.16	Команда info breakpoints	15
3.17	Установка еще одной точки останова по адресу инструкции	16
3.18	Просмотр информации с помощью i b	16
3.19	Просмотр содержимого регистров	16
3.20	Просмотр значения переменной msg1 по имени	16
3.21	Определение адреса переменной msg2 по дизассемблированной инструкции	17
3.22	Просмотр значения переменной msg2 по адресу	17
3.23	Изменение символа у msg1	17
3.24	Изменение символа у msg2	17
3.25	Выведение различных значений регистра edx	18
3.26	Изменение значения регистра ebx	18
3.27	Копирование файла	18
3.28	Создание исполняемого файла	18
3.29	Загрузка исполняемого файла в отладчик	19
3.30	Установка точки останова и ее запуск	19
3.31	Просмотр адреса в регистре esp	19
3.32	Просмотр адреса остальных позиций стека	20
4.1	Создание файла lab09-4.asm	21
4.2	Ввод текста программы как подпрограмма	21
4.3	Создание исполняемого файла и его запуск	22

4.4	Создание файла lab09-5.asm	22
4.5	Ввод текста программы для проверки	22
4.6	Создание исполняемого файла для работы с GDB и его загрузка .	23
4.7	Проверка программы с помощью GDB	23
4.8	Проверка программы с помощью GDB	24
4.9	Нахождение ошибки в строках	24
4.10	Исправление строк в программе	24
4.11	Создание исполняемого файла, загрузка в отладчик gdb и ее запуск	25

Список таблиц

1 Цель работы

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

2 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. Вторым этапом — 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.

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 строки, 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.

GDB (GNU Debugger — отладчик проекта GNU) 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.

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 программе можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом `-g`. Существует два режима отображения синтаксиса машинных команд: режим Intel, используемый в том числе в NASM, и режим AT&T (значительно отличающийся внешне). По умолчанию в дизассемблере GDB принят режим AT&T. Переключиться на отображение команд с привычным Intel'овским синтаксисом можно, введя команду `set disassembly-flavor intel`.

Для продолжения остановленной программы используется команда `continue` (`c`) (`gdb`) с '[аргумент]'. Выполнение программы будет происходить до следующей точки останова. В качестве аргумента может использоваться целое число `N`.

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.

Для вызова подпрограммы из основной программы используется инструкция `call`, которая заносит адрес следующей инструкции в стек и загружает в регистр `esp` 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 `ip`. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией `call`.

3 Выполнение лабораторной работы

Создала каталог для программ лабораторной работы №9, перешла в него и создала файл lab09-1.asm.

```
saburieva@edk3n35 ~/work/arch-pc $ mkdir ~/work/arch-pc/lab09
saburieva@edk3n35 ~/work/arch-pc $ cd ~/work/arch-pc/lab09
saburieva@edk3n35 ~/work/arch-pc/lab09 $ touch lab09-1.asm
saburieva@edk3n35 ~/work/arch-pc/lab09 $
```

Рис. 3.1: Создание каталога lab9, переход в него и создание файла lab09-1.asm

Внимательно изучила текст программы для вычисления арифметического выражения $f(x) = 2x + 7$ с помощью подпрограммы _calcul. Затем ввела в файл lab09-1.asm текст программы из листинга и ввела его в файл lab09-1.asm

```
GNU nano 2.2 ~/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 iprintf
call quit
;-----
; Подпрограмма вычисления
; выражения "2x+7"
;-----
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы
```

Рис. 3.2: Ввод текста программы в файл lab09-1.asm

Создала исполняемый файл и запустила его. На запрос “Введите x” ввела число 6. На экран вывелось уравнение, результатом которого является число 19 при подстановке вместо x числа 6. Само выражение вычисляется в подпрограмме.

```
sabur@eva8dk2n26: ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
sabur@eva8dk2n26: ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
sabur@eva8dk2n26: ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 6
2x+7=19
sabur@eva8dk2n26: ~/work/arch-pc/lab09 $
```

Рис. 3.3: Создание исполняемого файла и его запуск

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

```

;-----
;lab09-1.asm
SECTION .data
msg: db "Введите x: ",0
result: db "f(g(x))=",0
SECTION .bss
x: resb 80
res: resb 80
SECTION .text
GLOBAL _start
_start:
;-----
; Основная программа
;-----
mov eax, msg
call printf
mov ecx, x
mov edx, 80
call read
mov eax, x
call atoi
call _subcalcul ; Вызов подпрограммы _subcalcul
call _calcul
mov eax, result
call printf
mov ecx, [res]
call printf
call exit

_calcul:
push ebx
mov ebx, 2
mul ebx
add eax, 7
pop ebx
ret

_subcalcul:

```

Рис. 3.4: Изменение текста программы lab09-1.asm

Создала исполняемый файл и запустила его. На запрос “Введите x” ввела число 6. На экран вывелось значение 18. Получается справа x передается в подпрограмму `_calcul` из нее в подпрограмму `_subcalcul`, где вычисляется выражение $g(x)$, результат возвращается в `_calcul` и вычисляется выражение $f(g(x))$.

```
sabur@eva8dk2n26: ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
sabur@eva8dk2n26: ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
sabur@eva8dk2n26: ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 6
f(g(x))=18
sabur@eva8dk2n26: ~/work/arch-pc/lab09 $
```

Рис. 3.5: Создание исполняемого файла и его запуск

Создала файл lab08-2.asm для выполнения дальнейшей работы.

```
saburieva@dk2n26 ~/work/arch-pc/lab09 $ touch lab09-2.asm
saburieva@dk2n26 ~/work/arch-pc/lab09 $
```

Рис. 3.6: Создание файла lab09-2.asm

Внимательно изучила текст программы из листинга и ввела его в файл lab09-2.asm.

```
GNU nano 7.2 /afs/dk.sci.pfu.edu.ru/home/s/a/saburieva/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. 3.7: Ввод текста программы в файл lab09-2.asm

Создала исполняемый файл и запустила его. На экран вывелось сообщение “Hello, world!”.

```
saburieva@dk2n26 ~/work/arch-pc/lab09 $ nasm -f elf lab09-2.asm
saburieva@dk2n26 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
saburieva@dk2n26 ~/work/arch-pc/lab09 $ ./lab09-2
Hello, world!
saburieva@dk2n26 ~/work/arch-pc/lab09 $
```

Рис. 3.8: Создание исполняемого файла и его запуск

Для работы с GDB в исполняемый файл добавила отладочную информацию, для этого трансляцию программ проводила с ключом ‘-g’. Загрузила исполняемый файл в отладчик gdb.

```

saburievadk2n26 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
saburievadk2n26 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
saburievadk2n26 ~/work/arch-pc/lab09 $ gdb lab09-2
GNU gdb (Gentoo 13.2 vanilla) 13.2
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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...

```

Рис. 3.9: Создание исполняемого файла для работы с GDB и его загрузка

Проверила работу программы, запустив ее в оболочке GDB с помощью команды `run` (сокращённо `r`).

```

(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/s/a/saburieva/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 10057) exited normally]

```

Рис. 3.10: Проверка работы программы с помощью команды `r`

Для более подробного анализа программы установила брейкпоинт на метку `_start`, с которой начинается выполнение любой ассемблерной программы, и запустила её.

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/s/a/saburieva/work/arch-pc/lab09/lab09-2

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

```

Рис. 3.11: Установка брейкпоинта для подробного анализа программы

Посмотрела дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start`.

```
(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. 3.12: Просмотр дисассимилированного кода программы

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

```
(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. 3.13: Переключение на отображение команд с Intel'овским синтаксисом

Различие отображения синтаксиса машинных команд в режимах АТТ и Intel в том, что в представлении АТТ в виде шестнадцатиричного числа записаны первые аргументы всех команд, а в представлении Intel записаны адреса вторых аргументов.

Включила режим псевдографики для более удобного анализа программы.

```

0x0040002a <_start+42> int 0x00
0x0040002c <_start+44> mov eax,eax
0x00400031 <_start+45> mov ebx,ebx
0x00400036 <_start+54> int 0x00
0x00400038 add BYTE PTR [eax],al
0x0040003a add BYTE PTR [eax],al
0x0040003c add BYTE PTR [eax],al
0x0040003e add BYTE PTR [eax],al
0x00400040 add BYTE PTR [eax],al
0x00400042 add BYTE PTR [eax],al
0x00400044 add BYTE PTR [eax],al
0x00400046 add BYTE PTR [eax],al
0x00400048 add BYTE PTR [eax],al
0x0040004a add BYTE PTR [eax],al
0x0040004c add BYTE PTR [eax],al
0x0040004e add BYTE PTR [eax],al
0x00400050 add BYTE PTR [eax],al
0x00400052 add BYTE PTR [eax],al
0x00400054 add BYTE PTR [eax],al
0x00400056 add BYTE PTR [eax],al
0x00400058 add BYTE PTR [eax],al
0x0040005a add BYTE PTR [eax],al
0x0040005c add BYTE PTR [eax],al
0x0040005e add BYTE PTR [eax],al

```

Active process 18861 In: _start LS PC: 0x00400000
(gdb) layout regs
(gdb) layout asm
(x000) []

Рис. 3.14: Включение режима псевдографики с командой layout asm

[Register Values Unavailable]

```

0x0040002a <_start+42> int 0x00
0x0040002c <_start+44> mov eax,eax
0x00400031 <_start+45> mov ebx,ebx
0x00400036 <_start+54> int 0x00
0x00400038 add BYTE PTR [eax],al
0x0040003a add BYTE PTR [eax],al
0x0040003c add BYTE PTR [eax],al
0x0040003e add BYTE PTR [eax],al
0x00400040 add BYTE PTR [eax],al
0x00400042 add BYTE PTR [eax],al
0x00400044 add BYTE PTR [eax],al
0x00400046 add BYTE PTR [eax],al

```

Active process 18861 In: _start LS PC: 0x00400000
(gdb) layout regs
(gdb) layout asm
(gdb) layout regs

Рис. 3.15: Включение режима псевдографики с командой layout regs

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

```

(gdb) info breakpoints
Num   Type             Disp Enb Address      What
1     breakpoint       keep y   0x00400000  0x007-2.asm:9
      breakpoint already hit 1 time
(gdb) [ ]

```

Рис. 3.16: Команда info breakpoints

Установила еще одну точку останова по адресу инструкции. Адрес инструкции посмотрела в средней части экрана в левом столбце. Определила адрес предпоследней инструкции (mov ebx,0x0) и установила точку останова.

```

0x00400007 <start+15> mov     $0x3,$edx
0x00400014 <start+20> int     $0x0
0x00400016 <start+22> mov     $0x4,$eax
0x0040001b <start+27> mov     $0x1,$edx
0x00400020 <start+32> mov     $0x00400000,$ecx
0x00400025 <start+37> mov     $0x7,$edx
0x0040002a <start+42> int     $0x0
0x0040002c <start+44> mov     $0x1,$eax
0x00400031 <start+49> mov     $0x0,$edx
0x00400036 <start+54> int     $0x0

```

```

(gdb) layout regs
(gdb) b 0x00400031
Function "0x00400031" not defined.
(gdb) b 0x00400031
Function "0x00400031" not defined.
(gdb) b "0x00400031"
unmatched quote
(gdb) b *0x00400031
Breakpoint 1 at 0x00400031: file lab09-2.asm, line 20.
(gdb)

```

Рис. 3.17: Установка еще одной точки останова по адресу инструкции

Посмотрела информацию о всех установленных точках останова.

```

(gdb) i b
Num  Type      Disp Crb Address  What
1    breakpoint keep y  0x00400000 lab09-2.asm:9
      breakpoint already hit 1 time
2    breakpoint keep y  0x00400031 lab09-2.asm:20
(gdb)

```

Рис. 3.18: Просмотр информации с помощью i b

Посмотрела содержимого регистров с помощью команды info registers.

```

eax      0x0      0
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffc360 0xffffc360
ebp      0x0      0
esi      0x0      0
edi      0x0      0
eip      0x00400000 0x00400000 <start>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
--Type <RET> for more, q to quit, c to continue without paging--

```

Рис. 3.19: Просмотр содержимого регистров

Посмотрела значение переменной msg1 по имени.

```

(gdb) x/1ab $msg1
0x00400000 0x00400000: "Hello, "
(gdb)

```

Рис. 3.20: Просмотр значение переменной msg1 по имени

Посмотрела значение переменной msg2 по адресу. Адрес переменной определила по дизассемблированной инструкции. Посмотрела инструкцию mov esx,msg2 которая записывает в регистр esx адрес переменной msg2.


```

0x8040007 <start+15> mov     $0x1, %edx
0x8040014 <start+20> int     $0x0
0x8040016 <start+22> mov     $0x1, %eax
0x804001b <start+27> mov     $0x1, %ebx
0x8040020 <start+32> mov     $0x804a008, %ecx
0x8040025 <start+37> mov     $0x1, %edx
0x804002a <start+42> int     $0x0
0x804002c <start+44> mov     $0x1, %eax
1> 0x8040031 <start+49> mov     $0x0, %ebx
0x8040036 <start+54> int     $0x0

```

Рис. 3.21: Определение адреса переменной msg2 по дизассемблированной инструкции

```

(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n"<error: Cannot access memory at address 0x804a00f>
(gdb) 

```

Рис. 3.22: Просмотр значения переменной msg2 по адресу

Изменила первый символ переменной msg1.

```

(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) 

```

Рис. 3.23: Изменение символа у msg1

Изменила любой символ второй переменной msg2.

```

(gdb) set {char}&msg2='W'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "World!\n\034"
(gdb) 

```

Рис. 3.24: Изменение символа у msg2

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

```

(gdb) p/s $edx
$1 = 0
(gdb) p/x
$2 = 0x0
(gdb) p/t
$3 = 0
(gdb) 

```

Рис. 3.25: Выведение различных значений регистра edx

С помощью команды set изменила значение регистра ebx.

```

(gdb) set $ebx=2
(gdb) p/s $ebx
$4 = 58
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb) 

```

Рис. 3.26: Изменение значения регистра ebx

Завершила выполнение программы с помощью команды continue и вышла из GDB с помощью команды quit.

Скопировала 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.

```

sabrieva@kdn54: ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
sabrieva@kdn54: ~/work/arch-pc/lab09 $ 

```

Рис. 3.27: Копирование файла

Создала исполняемый файл.

```

sabrieva@kdn54: ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
sabrieva@kdn54: ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
sabrieva@kdn54: ~/work/arch-pc/lab09 $ 

```

Рис. 3.28: Создание исполняемого файла

Для загрузки в gdb программы с аргументами использовала ключ -args. Загрузила исполняемый файл в отладчик, указав аргументы.

```

GNU gdb (Gentoo 12.1 - vanilla) 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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>
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. 3.29: Загрузка исполняемого файла в отладчик

Дальше исследовала расположение аргументов командной строки в стеке после запуска программы с помощью gdb. Для начала установила точку останова перед первой инструкцией в программе и запустила ее.

```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/s/a/saburieva/work/arch-
pc/lab09/lab09-3 аргумент1 аргумент2 аргумент\ 3

Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb)

```

Рис. 3.30: Установка точки останова и ее запуск

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы), проверим так ли это.

```

5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) x/x $esp
0xfffffc3f0: 0x00000005
(gdb)

```

Рис. 3.31: Просмотр адреса в регистре esp

Посмотрела остальные позиции стека – по адресу [esp+4] располагается адрес в памяти, где находится имя программы, по адресу [esp+8] храниться адрес первого аргумента, по адресу [esp+12] – второго и т.д.

```

(gdb) x/s *(void**)(bexp + 4)
0xffffc54:  /afs/.dk.scl.pfu.edu.ru/home/s/af/saburieva/work/arch-pc/1ab09/1ab09-3"
(gdb) x/s *(void**)(bexp + 8)
0xffffc58:  "argument1"
(gdb) x/s *(void**)(bexp + 12)
0xffffc5c:  "argument"
(gdb) x/s *(void**)(bexp + 16)
0xffffc60:  "2"
(gdb) x/s *(void**)(bexp + 20)
0xffffc64:  "argument 3"
(gdb) x/s *(void**)(bexp + 24)
0x0:  -error: Cannot access memory at address 0x0>
(gdb) 

```

Рис. 3.32: Просмотр адреса остальных позиций стека

Их адреса располагаются в 4 байтах друг от друга, потому что именно столько занимает элемент стека.

4 Выполнение заданий для самостоятельной работы

Создала файл lab09-4.asm для преобразования программы из лабораторной работы №8 (Задание №1 для самостоятельной работы).



```
saburieva@dk8n69 ~/work/arch-pc/lab09 $ touch lab09-4.asm
saburieva@dk8n69 ~/work/arch-pc/lab09 $
```

Рис. 4.1: Создание файла lab09-4.asm

Вводя программу, изменила ее так, чтобы она вычислялась как подпрограмма.



```
GNU nano 6.4 /afs/dk.sci.pfu.edu.ru/home/s/a/saburieva/work/arch-pc/lab09/lab09-4.asm
#include "in_out.asm"
SECTION .data
f_x db "Функция: 2(x-1)",0h
msg db "Результат: ",0h

SECTION .text
global _start
_start:
    push ebx
    dec eax
    mov ebx, 2
    mul ebx
    pop ebx
    ret

    _start:
    pop ecx
    pop edx
    sub ecx, 1
    mov esi, 0

next:
    cmp ecx, 0h
    jz _end
    pop eax
    call atoi
    dec eax
    mov ebx, 2
    mul ebx
    add esi, eax
    loop next

_end:
    mov eax, f_x
    call sprint
    mov eax, msg
```

Рис. 4.2: Ввод текста программы как подпрограмма

Создала исполняемый файл и запустила его.

```

saburieva@dk3n35 ~/work/arch-pc/lab09 $ nasm -f elf lab09-4.asm
saburieva@dk3n35 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-4 lab09-4.o
saburieva@dk3n35 ~/work/arch-pc/lab09 $ ./lab09-4 1 2 3 4
функция: 2(x-1)Результат: 12
saburieva@dk3n35 ~/work/arch-pc/lab09 $

```

Рис. 4.3: Создание исполняемого файла и его запуск

Создала файл lab09-5.asm для проверки программы вычисления выражения $(3 + 2) * 4 + 5$.

```

saburieva@dk8n69 ~/work/arch-pc/lab09 $ touch lab09-5.asm
saburieva@dk8n69 ~/work/arch-pc/lab09 $

```

Рис. 4.4: Создание файла lab09-5.asm

Ввела текст программы из листинга для вычисления выражения $(3 + 2) * 4 + 5$.

```

saburieva@dk8n69:~/work/arch-pc/lab09  x  mrc [saburieva@dk8n69]:~/work/arch-pc/lab09
GNU nano 6.4 /afs/.dk.sci.pfu.edu.ru/home/s
#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 iprintlnf
call quit

```

Рис. 4.5: Ввод текста программы для проверки

Для работы с GDB в исполняемый файл добавила отладочную информацию, для этого трансляцию программ проводила с ключом '-g'. Загрузила исполняемый файл в отладчик gdb.

```
saburieva@dk8n69:~/work/arch-pc/lab09 x mc [saburieva@dk8n69]:~/work/arch-pc/lab09 x saburieva@dk8n69
saburieva@dk8n69 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-5.lst lab09-5.asm
saburieva@dk8n69 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-5 lab09-5.o
saburieva@dk8n69 ~/work/arch-pc/lab09 $ gdb lab09-5
GNU gdb (Gentoo 12.1 vanilla) 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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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 lab09-5...
(gdb) █
```

Рис. 4.6: Создание исполняемого файла для работы с GDB и его загрузка

При запуске данная программа дает неверный результат. Проверила это с помощью отладчика GDB, анализируя изменения значений регистров.

```
145     mov     ebx, 10
146     mul     ebx
147     inc     ecx
148     jmp     .multiplyLoop
149
150 .finished:
151     cmp     ecx, 0
152     je      .restore
153     mov     ebx, 10
154     div     ebx
155
156 .restore:
157     pop     esi
158     pop     edx
159     pop     ecx
160     pop     ebx
161     ret
162
```

Рис. 4.7: Проверка программы с помощью GDB

```

142
143     sub     bl, 48
144     add     eax, ebx
145     mov     ebx, 10
146     mul     ebx
147     inc     ecx
148     jmp     .multiplyLoop
149
150 .finished:
151     cmp     ecx, 0
152     je      .restore
153     mov     ebx, 10

```

Рис. 4.8: Проверка программы с помощью GDB

Проверив и проанализировав изменения значений регистров, нашла ошибку в следующих строках.

```

add ebx,eax
mov ecx,4
mul ecx
add ebx,5
mov edi,ebx

```

Рис. 4.9: Нахождение ошибки в строках

Далее исправила текст программы правильным образом, чтобы получился верный ответ.

```

GNU nano 7.2 /afs/.dk.sci.pfu.edu.ru/home/s/a/saburieva/work/arch-pc/lab09/
#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 sprintf
mov eax,edi
call iprintfLF
call quit

```

Рис. 4.10: Исправление строк в программе

Создала исполняемый файл и запустила его. Для работы с GDB в исполняемый файл добавила отладочную информацию, для этого трансляцию программ проводила с ключом '-g'. Загрузила исполняемый файл в отладчик gdb. В итоге

проверила работу программы, запустив ее в оболочке GDB с помощью команды `run`. Изменённая программа корректна.

```
saburieva@dk3n35 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-5.lst lab09-5.asm
saburieva@dk3n35 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-5 lab09-5.o
saburieva@dk3n35 ~/work/arch-pc/lab09 $ gdb lab09-5
GNU gdb (Gentoo 13.2 vanilla) 13.2
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-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
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 lab09-5...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/s/a/saburieva/work/arch-pc/lab09/lab09-5
Pezynrat: 25
[Inferior 1 (process 4805) exited normally]
(gdb) █
```

Рис. 4.11: Создание исполняемого файла, загрузка в отладчик gdb и ее запуск

5 Выводы

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

Список литературы