

Отчет по лабораторной работе №9

дисциплина: Архитектура компьютера

Михайлова Регина Алексеевна

Содержание

1	Цель работы	6
2	Выполнение лабораторной работы	7
3	Выполнение заданий для самостоятельной работы	27
4	Выводы	40
	Список литературы	41

Список иллюстраций

2.1	Создание файла	7
2.2	Проверка работы программы	10
2.3	Изменённый текст файла lab09-1.asm	11
2.4	Изменённый текст файла lab09-1.asm	12
2.5	Создание и запуск изменённого исполняемого файла lab09-1 . . .	13
2.6	Создание исполняемого файла lab09-2 и файла листинга lab09-2.lst	14
2.7	Отладчик gdb	14
2.8	Запуск программы с помощью команды run	15
2.9	Установка брейкпоинта на метку _start	15
2.10	Дисассимилированный код программы lab09-2	15
2.11	Дисассимилированный код программы lab09-2	16
2.12	Синтаксис Intel	16
2.13	Режим псевдографики	17
2.14	Проверка точек останова	18
2.15	Установка точки останова по адресу	18
2.16	Инструкция stepi	19
2.17	Инструкция stepi	19
2.18	Инструкция stepi	20
2.19	Инструкция stepi	20
2.20	Инструкция stepi	21
2.21	Инструкция stepi	21
2.22	Значение переменной msg1 по имени	22
2.23	Значение переменной msg2 по имени	22
2.24	Изменение первого символа	22
2.25	Вывод значения регистра edx	23
2.26	Изменение значения регистра edx	23
2.27	Завершение программы	24
2.28	Загрузка программы lab09-3 в gdb	24
2.29	Установка точки останова	25
2.30	Содержимое регистра esp	25
2.31	Содержимое стека	26
3.1	Текст файла funcR.asm	28
3.2	Создание и запуск исполняемого файла	28
3.3	Файл	29
3.4	Создание и запуск исполняемого файла	29
3.5	Загрузка файла в отладчик	30

3.6	Дисассимилированный код программы	30
3.7	Режим псевдографики	31
3.8	Просмотр регистров	32
3.9	Точка останова на инструкции add	33
3.10	Значения регистров	34
3.11	Команда si	35
3.12	Изменение значения регистра eax	36
3.13	Результат вычисления произведения	37
3.14	Завершение выполнения программы	38
3.15	Измененный текст программы в файле func2	39
3.16	Проверка работы программы	39

Список таблиц

1 Цель работы

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

2 Выполнение лабораторной работы

1. Создайте каталог для выполнения лабораторной работы № 9, перейдите в него и создайте файл lab09-1.asm (рис. 2.1): `mkdir ~/work/arch-pc/lab09 cd ~/work/arch-pc/lab09 touch lab09-1.asm`

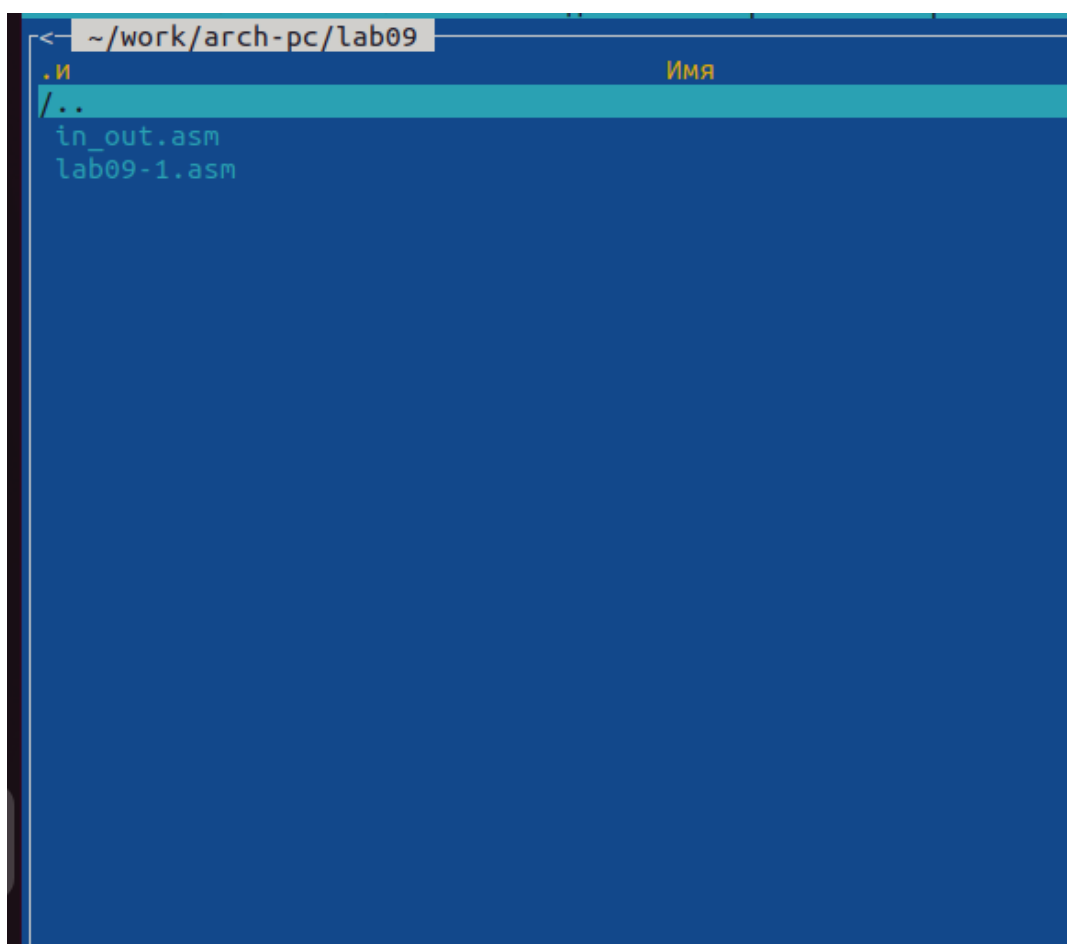


Рис. 2.1: Создание файла

2. В качестве примера рассмотрим программу вычисления арифметического выражения $x(x) = 2x + 7$ с помощью подпрограммы `_calcul`. В данном примере `x` вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Внимательно изучите текст программы (Листинг 9.1).

Листинг 9.1. Пример программы с использованием вызова подпрограммы

```
%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
mov [res],eax
ret ; выход из подпрограммы

```

Первые строки программы отвечают за вывод сообщения на экран (call sprint), чтение данных введенных с клавиатуры (call sread) и преобразования введенных данных из символьного вида в численный (call atoi).

```

mov eax, msg ; вызов подпрограммы печати сообщения
call sprint ; 'Введите x: '
mov ecx, x
mov edx, 80
call sread ; вызов подпрограммы ввода сообщения
mov eax,x ; вызов подпрограммы преобразования
call atoi ; ASCII кода в число, `eax=x`

```

После следующей инструкции `call _calcul`, которая передает управление подпрограмме `_calcul`, будут выполнены инструкции подпрограммы:

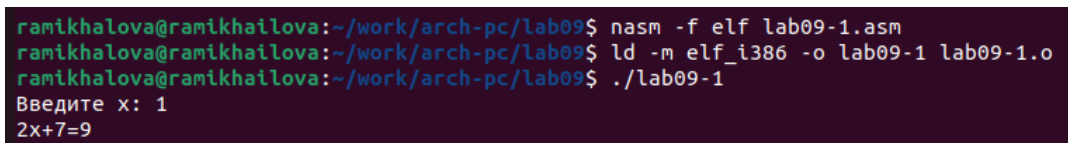
```

mov ebx,2
mul ebx
add eax,7

```

```
mov [res], eax
ret
```

Инструкция `ret` является последней в подпрограмме и ее исполнение приводит к возвращению в основную программу к инструкции, следующей за инструкцией `call`, которая вызвала данную подпрограмму. Последние строки программы реализуют вывод сообщения (`call sprint`), результата вычисления (`call iprintLF`) и завершение программы (`call quit`). Введите в файл `lab09-1.asm` текст программы из листинга 9.1. Создайте исполняемый файл и проверьте его работу (рис. 2.2).



```
ramikhalova@ramikhailova:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
ramikhalova@ramikhailova:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
ramikhalova@ramikhailova:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 1
2x+7=9
```

Рис. 2.2: Проверка работы программы

Измените текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $\text{X}(\text{X}(\text{X}))$, где X вводится с клавиатуры, $\text{X}(\text{X}) = 2\text{X} + 7$, $\text{X}(\text{X}) = 3\text{X} - 1$. Т.е. X передается в подпрограмму `_calcul` из нее в подпрограмму `_subcalcul`, где вычисляется выражение $\text{X}(\text{X})$, результат возвращается в `_calcul` и вычисляется выражение $\text{X}(\text{X}(\text{X}))$. Результат возвращается в основную программу для вывода результата на экран.

```

%include 'in_out.asm'
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2(3x-1)+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:
call _subcalcul
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы

_subcalcul:
mov ebx, 3
mul ebx
dec eax; eax=3x-1
ret

```

Рис. 2.3: Изменённый текст файла lab09-1.asm

```
SECTION .data
msg1: db "Hello, ",0x0
msg1Len: equ $ - msg1
msg2: db "world!",0xa
msg2Len: equ $ - msg2
SECTION .text
global _start
_start:
mov eax, 4
mov ebx, 1
mov ecx, msg1
mov edx, msg1Len
int 0x80
mov eax, 4
mov ebx, 1
mov ecx, msg2
mov edx, msg2Len
int 0x80
mov eax, 1
mov ebx, 0
int 0x80
```

Рис. 2.4: Изменённый текст файла lab09-1.asm

```

ramikhalova@ramikhailova:~/work/arch-pc/lab09$ nasm -f elf lab09-1.asm
ramikhalova@ramikhailova:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-1 lab09-1.o
ramikhalova@ramikhailova:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 1
2(3x-1)+7=11
ramikhalova@ramikhailova:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 5
2(3x-1)+7=35
ramikhalova@ramikhailova:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 8
2(3x-1)+7=53
ramikhalova@ramikhailova:~/work/arch-pc/lab09$ ./lab09-1
Введите x: 9
2(3x-1)+7=59

```

Рис. 2.5: Создание и запуск изменённого исполняемого файла lab09-1

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

Листинг 9.2. Программа вывода сообщения Hello world!

```

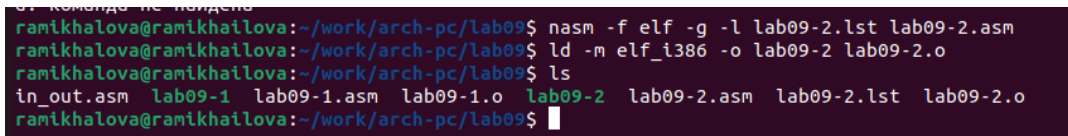
SECTION .data
msg1: db "Hello, ",0x0
msg1Len: equ $ - msg1
msg2: db "world!",0xa
msg2Len: equ $ - msg2
SECTION .text
global _start
_start:
mov eax, 4
mov ebx, 1
mov ecx, msg1
mov edx, msg1Len
int 0x80
mov eax, 4
mov ebx, 1
mov ecx, msg2
mov edx, msg2Len

```

```
int 0x80
mov eax, 1
mov ebx, 0
int 0x80
```

Получите исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’ (рис. 2.6).

```
nasm -f elf -g -l lab09-2.lst lab09-2.asm ld -m elf_i386 -o lab09-2 lab09-2.o
```



```
ramikhalova@ramikhalilova:~/work/arch-pc/lab09$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
ramikhalova@ramikhalilova:~/work/arch-pc/lab09$ ld -m elf_i386 -o lab09-2 lab09-2.o
ramikhalova@ramikhalilova:~/work/arch-pc/lab09$ ls
in_out.asm lab09-1 lab09-1.asm lab09-1.o lab09-2 lab09-2.asm lab09-2.lst lab09-2.o
ramikhalova@ramikhalilova:~/work/arch-pc/lab09$
```

Рис. 2.6: Создание исполняемого файла lab09-2 и файла листинга lab09-2.lst

Загрузите исполняемый файл в отладчик gdb (рис. 2.7): user@dk4n31:~\$ gdb lab09-2 Проверьте работу программы, запустив ее в оболочке GDB с помощью команды run (со- кращённо r) (рис. 2.8).



```
ramikhalova@ramikhalilova:~/work/arch-pc/lab09$ gdb lab09-2
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
(gdb)
```

Рис. 2.7: Отладчик gdb

```

(gdb) run
Starting program: /home/ramikhalova/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 42301) exited normally]
(gdb)

```

Рис. 2.8: Запуск программы с помощью команды run

Для более подробного анализа программы установите брейкпоинт на метку `_start` (рис. 2.9), с которой начинается выполнение любой ассемблерной программы, и запустите её.

```

[Inferior 1 (process 42301) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /home/ramikhalova/work/arch-pc/lab09/lab09-2

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

```

Рис. 2.9: Установка брейкпоинта на метку `_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. 2.10: Дисассимилированный код программы lab09-2

Посмотрите дисассимилированный код программы с помощью команды `disassemble` начиная с метки `_start` (рис. 2.11).

```

(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:    mov     $0x4,%eax
      0x08049005 <+5>:    mov     $0x1,%ebx
      0x0804900a <+10>:   mov     $0x804a000,%ecx
      0x0804900f <+15>:   mov     $0x8,%edx
      0x08049014 <+20>:   int     $0x80
      0x08049016 <+22>:   mov     $0x4,%eax
      0x0804901b <+27>:   mov     $0x1,%ebx
      0x08049020 <+32>:   mov     $0x804a008,%ecx
      0x08049025 <+37>:   mov     $0x7,%edx
      0x0804902a <+42>:   int     $0x80
      0x0804902c <+44>:   mov     $0x1,%eax
      0x08049031 <+49>:   mov     $0x0,%ebx
      0x08049036 <+54>:   int     $0x80
End of assembler dump.
(gdb)

```

Рис. 2.11: Дисассимилированный код программы lab09-2

Переключитесь на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (рис. 2.12).

```

(gdb) set disassembly-flavor intel
(gdb) disassemble _start
Dump of assembler code for function _start:
=> 0x08049000 <+0>:    mov     eax,0x4
      0x08049005 <+5>:    mov     ebx,0x1
      0x0804900a <+10>:   mov     ecx,0x804a000
      0x0804900f <+15>:   mov     edx,0x8
      0x08049014 <+20>:   int     0x80
      0x08049016 <+22>:   mov     eax,0x4
      0x0804901b <+27>:   mov     ebx,0x1
      0x08049020 <+32>:   mov     ecx,0x804a008
      0x08049025 <+37>:   mov     edx,0x7
      0x0804902a <+42>:   int     0x80
      0x0804902c <+44>:   mov     eax,0x1
      0x08049031 <+49>:   mov     ebx,0x0
      0x08049036 <+54>:   int     0x80
End of assembler dump.
(gdb)

```

Рис. 2.12: Синтаксис Intel

Перечислите различия отображения синтаксиса машинных команд в режимах АТТ и Intel. Включите режим псевдографики для более удобного анализа программы (рис. 2.13):



Рис. 2.13: Режим псевдографики

В этом режиме есть три окна: • В верхней части видны названия регистров и их текущие значения; • В средней части виден результат дисассимилирования программы; • Нижняя часть доступна для ввода команд.

Установить точку останова можно командой `break` (кратко `b`). Типичный аргумент этой команды — место установки. Его можно задать или как номер строки программы (имеет смысл, если есть исходный файл, а программа компилировалась с информацией об отладке), или как имя метки, или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка»: На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверьте это с помощью команды `info breakpoints` (кратко `i b`) (рис. 2.14):

```

0x804903e      add     BYTE PTR [eax],al

native process 42494 In: _start
(gdb) layout regs
(gdb) info breakpoints
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab09-2.asm:9
        breakpoint already hit 1 time
(gdb)

```

Рис. 2.14: Проверка точек останова

Установим еще одну точку останова по адресу инструкции. Адрес инструкции можно увидеть в средней части экрана в левом столбце соответствующей инструкции. Определите адрес предпоследней инструкции (`mov ebx,0x0`) и установите точку останова (рис. 2.15).

```

B+> 0x8049000 <_start>      mov     eax,0x4
0x8049005 <_start+5>      mov     ebx,0x1
0x804900a <_start+10>     mov     ecx,0x804a000
0x804900f <_start+15>     mov     edx,0x8
0x8049014 <_start+20>     int     0x80
0x8049016 <_start+22>     mov     eax,0x4
0x804901b <_start+27>     mov     ebx,0x1
0x8049020 <_start+32>     mov     ecx,0x804a008
0x8049025 <_start+37>     mov     edx,0x7
0x804902a <_start+42>     int     0x80
0x804902c <_start+44>     mov     eax,0x1
b+> 0x8049031 <_start+49>     mov     ebx,0x0
0x8049036 <_start+54>     int     0x80
0x8049038      add     BYTE PTR [eax],al
0x804903a      add     BYTE PTR [eax],al
0x804903c      add     BYTE PTR [eax],al
0x804903e      add     BYTE PTR [eax],al

native process 42494 In: _start
(gdb) layout regs
(gdb) info breakpoints
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab09-2.asm:9
        breakpoint already hit 1 time
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) i b
Num      Type      Disp Enb Address      What
1        breakpoint keep y  0x08049000 lab09-2.asm:9
        breakpoint already hit 1 time
2        breakpoint keep y  0x08049031 lab09-2.asm:20
(gdb)

```

Рис. 2.15: Установка точки останова по адресу

Выполните 5 инструкций с помощью команды `stepi` (или `si`) и проследите за

изменением значений регистров. Значения каких регистров изменяются? Посмотреть содержимое регистров также можно с помощью команды `info registers` (или `i r`) (рис. 2.16 - 2.21).

```

B+ 0x8049000 <_start> mov eax,0x4
> 0x8049005 <_start+5> mov ebx,0x1
0x804900a <_start+10> mov ecx,0x804a000
0x804900f <_start+15> mov edx,0x8
0x8049014 <_start+20> int 0x80
0x8049016 <_start+22> mov eax,0x4
0x804901b <_start+27> mov ebx,0x1
0x8049020 <_start+32> mov ecx,0x804a008
0x8049025 <_start+37> mov edx,0x7
0x804902a <_start+42> int 0x80
0x804902c <_start+44> mov eax,0x1
b+ 0x8049031 <_start+49> mov ebx,0x0
0x8049036 <_start+54> int 0x80
0x8049038 add BYTE PTR [eax],al
0x804903a add BYTE PTR [eax],al
0x804903c add BYTE PTR [eax],al
0x804903e add BYTE PTR [eax],al

native process 42494 In: start
(gdb) layout regs
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb) b *0x8049031
Breakpoint 2 at 0x08049031: file lab09-2.asm, line 20.
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
2 breakpoint keep y 0x08049031 lab09-2.asm:20
(gdb) si

```

Рис. 2.16: Инструкция `stepi`

```

Register group: general
eax 0x4 0 ecx 0x0 0 edx 0x0 0
ebx 0x0 0 esp 0xffffd110 0xffffd110 ebp 0x0 0
esi 0x0 0 edi 0x0 0 eip 0x8049005 0x8049005 <_start+5>
eflags 0x202 43 [ IF ] cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43 fs 0x0 0
gs 0x0 0

B+ 0x8049000 <_start> mov eax,0x4
> 0x8049005 <_start+5> mov ebx,0x1
0x804900a <_start+10> mov ecx,0x804a000
0x804900f <_start+15> mov edx,0x8
0x8049014 <_start+20> int 0x80
0x8049016 <_start+22> mov eax,0x4
0x804901b <_start+27> mov ebx,0x1
0x8049020 <_start+32> mov ecx,0x804a008
0x8049025 <_start+37> mov edx,0x7
0x804902a <_start+42> int 0x80
0x804902c <_start+44> mov eax,0x1
b+ 0x8049031 <_start+49> mov ebx,0x0
0x8049036 <_start+54> int 0x80
0x8049038 add BYTE PTR [eax],al
0x804903a add BYTE PTR [eax],al
0x804903c add BYTE PTR [eax],al
0x804903e add BYTE PTR [eax],al

native process 42494 In: start
(gdb) layout regs
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb) b *0x8049031
Breakpoint 2 at 0x08049031: file lab09-2.asm, line 20.
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
2 breakpoint keep y 0x08049031 lab09-2.asm:20
(gdb) si

```

Рис. 2.17: Инструкция `stepi`

```

Register group: general
eax    0x4      4      ecx    0x0      0      edx    0x0      0
ebx    0x1      1      esp    0xffffd110 0      ebp    0x0      0x0
esi    0x0      0      edi    0x0      0      iip    0x0049000a 0x0049000a < start+10>
eflags 0x202    [ IF ]  cs     0x23    35      ss     0x2b    43
ds      0x2b    43      es     0x2b    43      fs     0x0      0
gs      0x0      0

B+ 0x00490000 < start> mov eax,0x4
> 0x0049000a < start+10> mov ecx,0x004a0000
0x00490007 < start+15> mov edx,0x8
0x00490010 < start+20> int 0x0
0x00490016 < start+22> mov eax,0x4
0x0049001b < start+27> mov ebx,0x1
0x00490020 < start+32> mov ecx,0x004a0000
0x00490025 < start+37> mov edx,0x7
0x0049002a < start+42> int 0x80
0x0049002c < start+44> mov eax,0x1
B+ 0x00490031 < start+49> mov ebx,0x0
0x00490036 < start+54> int 0x80
0x00490038 add BYTE PTR [eax],al
0x0049003a add BYTE PTR [eax],al
0x0049003c add BYTE PTR [eax],al
0x0049003e add BYTE PTR [eax],al

Active process 42494 In: start
(gdb) layout regs
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x0049000a lab09-2.asm:9
breakpoint already hit 1 time
(gdb) b *0x00490031
breakpoint 2 at 0x00490031: file lab09-2.asm, line 20.
(gdb) t b
Num Type Disp Enb Address What
1 breakpoint keep y 0x0049000a lab09-2.asm:9
breakpoint already hit 1 time
2 breakpoint keep y 0x00490031 lab09-2.asm:20
(gdb) st
(gdb) st
(gdb) st

```

Рис. 2.18: Инструкция stepi

```

Register group: general
eax    0x4      4      ecx    0x004a0000 111320832  edx    0x0      0
ebx    0x1      1      esp    0xffffd110 0xffffd110  ebp    0x0      0x0
esi    0x0      0      edi    0x0      0      iip    0x0049000f 0x0049000f < start+15>
eflags 0x202    [ IF ]  cs     0x23    35      ss     0x2b    43
ds      0x2b    43      es     0x2b    43      fs     0x0      0
gs      0x0      0

B+ 0x00490000 < start> mov eax,0x4
0x00490005 < start+5> mov ebx,0x1
0x0049000a < start+10> mov ecx,0x004a0000
> 0x0049000f < start+15> mov ecx,0x0
0x00490016 < start+22> int 0x0
0x0049001b < start+27> mov eax,0x4
0x00490020 < start+32> mov ecx,0x004a0000
0x00490025 < start+37> mov edx,0x7
0x0049002a < start+42> int 0x80
0x0049002c < start+44> mov eax,0x1
B+ 0x00490031 < start+49> mov ebx,0x0
0x00490036 < start+54> int 0x80
0x00490038 add BYTE PTR [eax],al
0x0049003a add BYTE PTR [eax],al
0x0049003c add BYTE PTR [eax],al
0x0049003e add BYTE PTR [eax],al

Active process 42494 In: start
(gdb) layout regs
(gdb) info breakpoints
Num Type Disp Enb Address What
1 breakpoint keep y 0x0049000a lab09-2.asm:9
breakpoint already hit 1 time
(gdb) b *0x00490031
breakpoint 2 at 0x00490031: file lab09-2.asm, line 20.
(gdb) t b
Num Type Disp Enb Address What
1 breakpoint keep y 0x0049000a lab09-2.asm:9
breakpoint already hit 1 time
2 breakpoint keep y 0x00490031 lab09-2.asm:20
(gdb) st
(gdb) st
(gdb) st

```

Рис. 2.19: Инструкция stepi

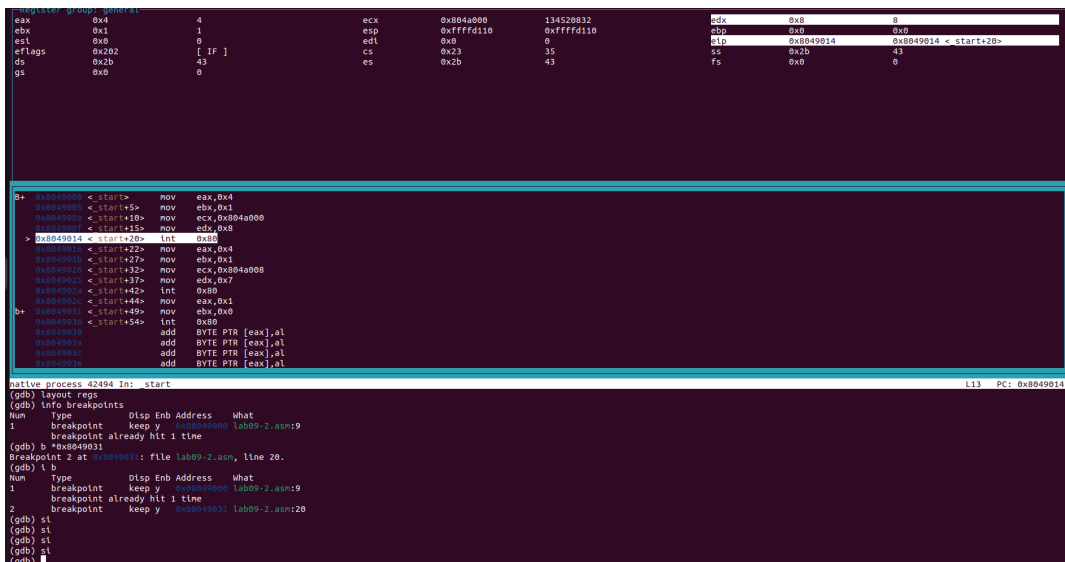


Рис. 2.20: Инструкция stepi

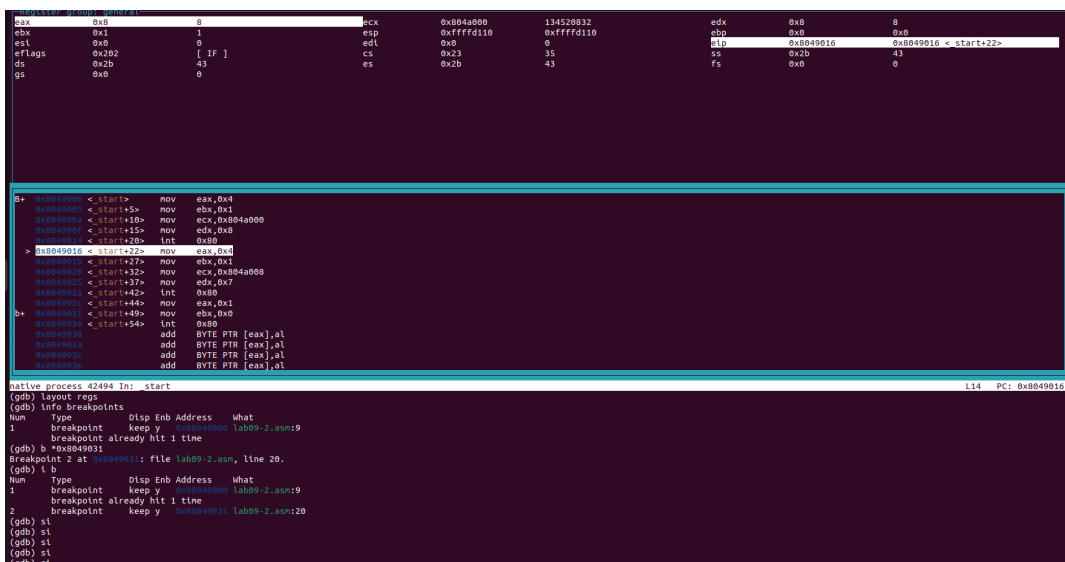


Рис. 2.21: Инструкция stepi

Для отображения содержимого памяти можно использовать команду x, которая выдаёт содержимое ячейки памяти по указанному адресу. Формат, в котором выводятся данные, можно задать после имени команды через косую черту: x/NFU. С помощью команды x & также можно посмотреть содержимое переменной.

Посмотрите значение переменной `msg1` по имени (рис. 2.22).

```
gs      0x0      0
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
(gdb) █
```

Рис. 2.22: Значение переменной `msg1` по имени

Посмотрите значение переменной `msg2` по адресу. Адрес переменной можно определить по дизассемблированной инструкции. Посмотрите инструкцию `mov esx,msg2` которая записывает в регистр `esx` адрес переменной `msg2` (рис. 2.23).

```
gs      0x0      0
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb) █
```

Рис. 2.23: Значение переменной `msg2` по имени

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). Измените первый символ переменной `msg1` (рис. 2.24):

```
gs      0x0      0
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hello, "
(gdb) set {char}msg2='R'
'msg2' has unknown type; cast it to its declared type
(gdb) set {char}&msg2='R'
(gdb) x/1sb &msg2
0x804a008 <msg2>: "Rorld!\n\034"
(gdb) █
```

Рис. 2.24: Изменение первого символа

Замените любой символ во второй переменной `msg2`. Чтобы посмотреть значения регистров используется команда `print /F` (перед именем регистра обя-

зательно ставится префикс \$) (рис. 9.6): p/F \$ Выведете в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx (рис. 2.25). С помощью команды set измените значение регистра ebx (рис. 2.26):

```
(gdb) p/x $edx
$1 = 0x8
(gdb) p/t $edx
$2 = 1000
(gdb) p/s $edx
$3 = 8
(gdb)
```

Рис. 2.25: Вывод значения регистра edx

```
(gdb) set $ebx='2'
(gdb) p/s $ebx
$4 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb)
```

Рис. 2.26: Изменение значения регистра ebx

В первом случае мы занесли в регистр ebx символ '2', поэтому после запроса p/s на вывод значения регистра на экране мы увидели код символа "2", а именно - 50. А в случае, когда в регистр изначально было занесено число 2, а не символ, команда p/s \$ebx вывела значение 2.

Завершите выполнение программы с помощью команды continue (рис. 2.27) (сокращенно c) или stepi (сокращенно si) и выйдите из GDB с помощью команды quit (сокращенно q).

```

$5 = 2
(gdb) c
Continuing.
Rorld!

Breakpoint 2, _start () at lab09-2.asm:20
(gdb)

```

Рис. 2.27: Завершение программы

Скопируйте 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: `cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm` Создайте исполняемый файл.

Для загрузки в gdb программы с аргументами необходимо использовать ключ `-args`. Загрузите исполняемый файл в отладчик, указав аргументы (рис. 2.28): `gdb -args lab09-3 аргумент1 аргумент 2 'аргумент 3'`

```

panikhalov@panikhallova:~/work/arch-pc/lab09$ gdb -args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (Ubuntu 12.1-0ubuntu1-22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-3...
(gdb)

```

Рис. 2.28: Загрузка программы lab09-3 в gdb

Как отмечалось в предыдущей лабораторной работе, при запуске программы аргументы командной строки загружаются в стек. Исследуем расположение аргументов командной строки в стеке после запуска программы с помощью gdb. Для начала установим точку останова перед первой инструкцией в программе и запустим ее (рис. 2.29).


```

(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /home/ranikhalova/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3
Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в "ecx" количество
(gdb)

```

Рис. 2.29: Установка точки останова

Адрес вершины стека храниться в регистре `esp` и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы) (рис. 2.30).

```

5      pop ecx ; Извлекаем из стека в "ecx" количество
(gdb) x/x $esp
0xffffd0c0: 0x00000005
(gdb)

```

Рис. 2.30: Содержимое регистра `esp`

Как видно, число аргументов равно 5 – это имя программы `lab09-3` и непосредственно аргументы: `аргумент1`, `аргумент 2` и `‘аргумент 3’`. Посмотрите 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 (рис. 2.31).

```

(gdb) x/x $esp
0xfffffd0c0: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xfffffd29f: "/home/ramikhalova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xfffffd2cc: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xfffffd2de: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xfffffd2ef: "2"
(gdb) x/s *(void**)(esp + 20)
0xfffffd2f1: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb) █

```

Рис. 2.31: Содержимое 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. Как мы видим, шаг изменения равен 4. Шаг имеет такое значение, потому что при добавлении значения каждого аргумента в стек значение регистра esp увеличивается на 4

3 Выполнение заданий для самостоятельной работы

Задание 1

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

Теперь для вычисления значения этой функции в цикле мы будем вызывать вспомогательную подпрограмму. Создадим файл `funcR.asm` и впишем в него текст программы (рис. 3.1).

```

GNU nano 6.2
#include 'in_out.asm'
section .data
msg db "Результат: ",0
section .text
global _start
_start:
pop ecx
pop edx
sub ecx,1
mov esi,0
;10(x-1)
next:
cmp ecx,0h
jz _end
pop eax
call atoi
call _calcul
add esi,eax
loop next

_end:
mov eax,msg
call sprint
mov eax,esi
call iprintLF
call quit

_calcul:
dec eax
mov ebx,10
mul ebx
ret

```

Рис. 3.1: Текст файла funcR.asm

Создадим исполняемый файл и проверим его работу с теми же аргументами (рис. 3.2).

```

ramikhalova@ramikhailova:~/work/arch-pc/lab09$ nasm -f elf funcR.asm
ramikhalova@ramikhailova:~/work/arch-pc/lab09$ ld -m elf_i386 -o funcR funcR.o
ramikhalova@ramikhailova:~/work/arch-pc/lab09$ ./funcR 1 2 3
Результат: 30
ramikhalova@ramikhailova:~/work/arch-pc/lab09$ ./funcR 4 7 40
4: команда не найдена
ramikhalova@ramikhailova:~/work/arch-pc/lab09$ ./funcR 4 7 40
Результат: 480
ramikhalova@ramikhailova:~/work/arch-pc/lab09$ ./funcR 5 60 23
Результат: 850
ramikhalova@ramikhailova:~/work/arch-pc/lab09$

```

Рис. 3.2: Создание и запуск исполняемого файла

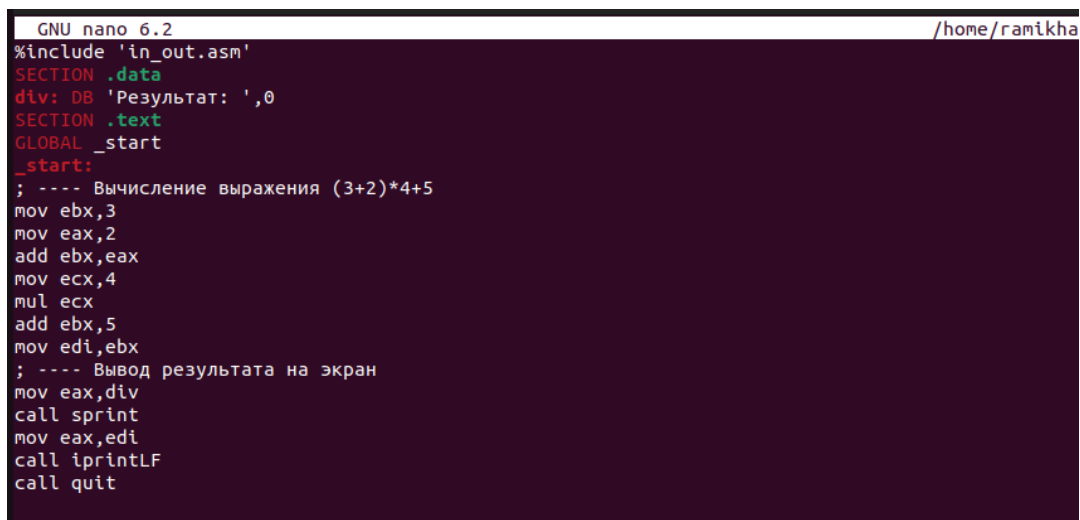
Программа работает корректно: выдаются верные значения сумм. Приступим к выполнению следующего задания.

Задание 2

Создадим файл `func2.asm` и впишем в него текст программы вычисления выражения

$$(3 + 2) * 4 + 5$$

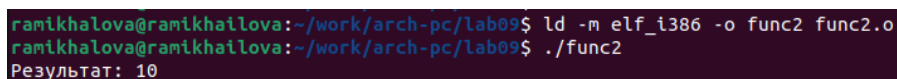
(рис. 3.3).



```
GNU nano 6.2 /home/ramikha
#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. 3.3: Файл

Создадим исполняемый файл и запустим его (рис. 3.4).



```
ramikhalova@ramikhailova:~/work/arch-pc/lab09$ ld -m elf_i386 -o func2 func2.o
ramikhalova@ramikhailova:~/work/arch-pc/lab09$ ./func2
Результат: 10
```

Рис. 3.4: Создание и запуск исполняемого файла

Программа и правда выдает неверный результат. С помощью отладчика GDB проанализируем изменения значений регистров, определим ошибку и исправим её. Для начала загрузим исполняемый файл в отладчик (рис. 3.5).

```

Результат: 10
ramikhalova@ramikhalova:~/work/arch-pc/lab09$ gdb func2
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from func2...
(gdb)

```

Рис. 3.5: Загрузка файла в отладчик

Посмотрим дисассимилированный код программы с помощью команды `disassemble`, начиная с метки `_start` (рис. 3.6).

```

(gdb) disassemble _start
Dump of assembler code for function _start:
   0x080490e8 <+0>:  mov     $0x3,%ebx
   0x080490ed <+5>:  mov     $0x2,%eax
   0x080490f2 <+10>: add     %eax,%ebx
   0x080490f4 <+12>: mov     $0x4,%ecx
   0x080490f9 <+17>: mul     %ecx
   0x080490fb <+19>: add     $0x5,%ebx
   0x080490fe <+22>: mov     %ebx,%edi
   0x08049100 <+24>: mov     $0x804a000,%eax
   0x08049105 <+29>: call    0x0804900f <_sprintf>
   0x0804910a <+34>: mov     %edi,%eax
   0x0804910c <+36>: call    0x08049086 <_lprintf@plt>
   0x08049111 <+41>: call    0x080490db <_quit>
End of assembler dump.
(gdb)

```

Рис. 3.6: Дисассимилированный код программы

Включим режим псевдографики (рис. 3.7).

```
[ Register Values Unavailable ]

B+> 0x80490e8 <_start> mov ebx,0x3
0x80490ed <_start+5> mov eax,0x2
0x80490f2 <_start+10> add ebx,eax
0x80490f4 <_start+12> mov ecx,0x4
0x80490f9 <_start+17> mul ecx
0x80490fb <_start+19> add ebx,0x5
0x80490fe <_start+22> mov edi,ebx
0x8049100 <_start+24> mov eax,0x804a000
0x8049105 <_start+29> call 0x804900f <sprint>
0x804910a <_start+34> mov eax,edi
0x804910c <_start+36> call 0x8049086 <fprintf>
0x8049111 <_start+41> call 0x80490db <quit>
0x8049116 add BYTE PTR [eax],al
0x8049118 add BYTE PTR [eax],al
0x804911a add BYTE PTR [eax],al
0x804911c add BYTE PTR [eax],al
0x804911e add BYTE PTR [eax],al

native process 24813 In: _start
(gdb) layout regs
(gdb)
```

Рис. 3.7: Режим псевдографики

Используем команду для просмотра значений регистров (рис. 3.8).

```
B+> 0x80490e8 <_start> mov ebx,0x3
0x80490ed <_start+5> mov eax,0x2
0x80490f2 <_start+10> add ebx,eax
0x80490f4 <_start+12> mov ecx,0x4
0x80490f9 <_start+17> mul ecx
0x80490fb <_start+19> add ebx,0x5
0x80490fe <_start+22> mov edi,ebx
0x8049100 <_start+24> mov eax,0x804a000
0x8049105 <_start+29> call 0x804900f <sprint>
0x804910a <_start+34> mov eax,edi
0x804910c <_start+36> call 0x8049086 <iprintLF>
0x8049111 <_start+41> call 0x80490db <quit>
0x8049116 add BYTE PTR [eax],al
0x8049118 add BYTE PTR [eax],al
0x804911a add BYTE PTR [eax],al
0x804911c add BYTE PTR [eax],al
0x804911e add BYTE PTR [eax],al

native process 24813 In: _start
(gdb) i r
eax            0x0            0
ecx            0x0            0
edx            0x0            0
ebx            0x0            0
esp            0xffffd120     0xffffd120
ebp            0x0            0x0
esi            0x0            0
edi            0x0            0
eip            0x80490e8     0x80490e8 <_start>
eflags        0x202          [ IF ]
cs             0x23          35
ss             0x2b          43
ds             0x2b          43
es             0x2b          43
fs             0x0            0
gs             0x0            0
(gdb) █
```

Рис. 3.8: Просмотр регистров

С помощью команды `break` установим точку останова по адресу на инструкции `add ebx,eax` (рис. 3.9).


```
B+> 0x80490e8 <_start> mov ebx,0x3
0x80490ed <_start+5> mov eax,0x2
b+ 0x80490f2 <_start+10> add ebx,eax
0x80490f4 <_start+12> mov ecx,0x4
0x80490f9 <_start+17> mul ecx
0x80490fb <_start+19> add ebx,0x5
0x80490fe <_start+22> mov edi,ebx
0x8049100 <_start+24> mov eax,0x804a000
0x8049105 <_start+29> call 0x804900f <sprintf>
0x804910a <_start+34> mov eax,edi
0x804910c <_start+36> call 0x8049086 <iprintf>
0x8049111 <_start+41> call 0x80490db <quit>
0x8049116 add BYTE PTR [eax],al
0x8049118 add BYTE PTR [eax],al
0x804911a add BYTE PTR [eax],al
0x804911c add BYTE PTR [eax],al
0x804911e add BYTE PTR [eax],al

native process 24813 In: _start
ebp 0x0 0x0
esi 0x0 0
edi 0x0 0
eip 0x80490e8 0x80490e8 <_start>
eflags 0x202 [ IF ]
cs 0x23 35
ss 0x2b 43
ds 0x2b 43
es 0x2b 43
fs 0x0 0
gs 0x0 0
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x80490e8 func2.asm:8
breakpoint already hit 1 time
(gdb) b *0x80490f2
Breakpoint 2 at 0x80490f2: file func2.asm, line 10.
(gdb) 
```

Рис. 3.9: Точка останова на инструкции add

Посмотрим значения регистров на этом этапе с помощью команды i r (рис. 3.10).

```
B+> 0x80490e8 <_start> mov ebx,0x3
0x80490ed <_start+5> mov eax,0x2
b+ 0x80490f2 <_start+10> add ebx,eax
0x80490f4 <_start+12> mov ecx,0x4
0x80490f9 <_start+17> mul ecx
0x80490fb <_start+19> add ebx,0x5
0x80490fe <_start+22> mov edi,ebx
0x8049100 <_start+24> mov eax,0x804a000
0x8049105 <_start+29> call 0x804900f <sprint>
0x804910a <_start+34> mov eax,edi
0x804910c <_start+36> call 0x8049086 <iprintLF>
0x8049111 <_start+41> call 0x80490db <quit>
0x8049116 add BYTE PTR [eax],al
0x8049118 add BYTE PTR [eax],al
0x804911a add BYTE PTR [eax],al
0x804911c add BYTE PTR [eax],al
0x804911e add BYTE PTR [eax],al

native process 24813 In: _start
(gdb) i r
eax            0x0            0
ecx            0x0            0
edx            0x0            0
ebx            0x0            0
esp            0xffffd120     0xffffd120
ebp            0x0            0x0
esi            0x0            0
edi            0x0            0
eip            0x80490e8     0x80490e8 <_start>
eflags         0x202         [ IF ]
cs             0x23          35
ss             0x2b          43
ds             0x2b          43
es             0x2b          43
fs             0x0            0
gs             0x0            0
(gdb)
```

Рис. 3.10: Значения регистров

Далее с помощью команды `si` перейдём к следующей инструкции и проследим за изменением значений регистров (рис. 3.11).

```

Register group: general
eax      0x2      2      ecx      0x0      0
ebx      0x5      5      esp      0xffffd120  0xffffd120
esi      0x0      0      edi      0x0      0
eflags   0x206    [ PF IF ]  cs      0x23    35
ds       0x2b     43      es      0x2b     43
gs       0x0      0

B+ 0x80490e6 <_start> mov ebx,0x3
0x80490ed <_start+5> mov eax,0x2
B+ 0x80490f2 <_start+10> add ebx,eax
> 0x80490f4 <_start+12> mov ecx,0x4
0x80490f9 <_start+17> mul ecx
0x80490fb <_start+19> add ebx,0x5
0x80490fe <_start+22> mov edi,ebx
0x8049100 <_start+24> mov eax,0x804a000
0x8049105 <_start+29> call 0x804900f <sprint>
0x804910a <_start+34> mov eax,edi
0x804910c <_start+36> call 0x8049086 <iprintf>
0x8049111 <_start+41> call 0x80490db <quit>
0x8049116 add BYTE PTR [eax],al
0x8049118 add BYTE PTR [eax],al
0x804911a add BYTE PTR [eax],al
0x804911c add BYTE PTR [eax],al
0x804911e add BYTE PTR [eax],al

native process 24813 In: _start
esp      0xffffd120  0xffffd120
ebp      0x0        0x0
esi      0x0        0
edi      0x0        0
eip      0x80490e8  0x80490e8 <_start>
eflags   0x202      [ IF ]
cs       0x23       35
ss       0x2b       43
ds       0x2b       43
es       0x2b       43
fs       0x0        0
gs       0x0        0
(gdb) si

```

Рис. 3.11: Команда si

Результат суммы чисел 2 и 3 записался в регистр ebx. Это могло послужить проблемой для дальнейшего вычисления произведения. Исправим это, изменив значение регистра eax и занеся в него значение 5 с помощью ко-манды set (рис. 3.12).

```

Register group: general
eax      0x5      5      ecx      0x4      4
ebx      0x5      5      esp      0xffffd120  0xffffd120
esi      0x0      0      edi      0x0      0
eflags   0x206    [ PF IF ]  cs      0x23     35
ds       0x2b     43      es      0x2b     43
gs       0x0      0

B+ 0x80490e8 <_start> mov ebx,0x3
0x80490ed <_start+5> mov eax,0x2
B+ 0x80490f2 <_start+10> add ebx,eax
0x80490f4 <_start+12> mov ecx,0x4
> 0x80490f9 <_start+17> mul ecx
0x80490fb <_start+19> add ebx,0x5
0x80490fe <_start+22> mov edi,ebx
0x8049100 <_start+24> mov eax,0x804a000
0x8049105 <_start+29> call 0x804900f <sprint>
0x804910a <_start+34> mov eax,edi
0x804910c <_start+36> call 0x8049086 <lprintf>
0x8049111 <_start+41> call 0x80490db <quit>
0x8049116 add BYTE PTR [eax],al
0x8049118 add BYTE PTR [eax],al
0x804911a add BYTE PTR [eax],al
0x804911c add BYTE PTR [eax],al
0x804911e add BYTE PTR [eax],al

native process 24813 In: _start
eip      0x80490e8 0x80490e8 <_start>
eflags   0x202 [ IF ]
cs       0x23 35
ss       0x2b 43
ds       0x2b 43
es       0x2b 43
fs       0x0 0
gs       0x0 0
(gdb) si
(gdb) si

Breakpoint 2, _start () at func2.asm:10
(gdb) si
(gdb) set $eax=5
(gdb) p/s $eax
$1 = 5
(gdb) si
(gdb)

```

Рис. 3.12: Изменение значения регистра eax

Двигаемся дальше. Мы поместили в регистр ecx значение 4 для вычисления произведения. После произведения значения регистров будут следующими (рис. 3.13):

```

Register group: general
eax      0x14      20      ecx      0x4      4
ebx      0x5      5      esp      0xffffd120  0xffffd120
esi      0x0      0      edi      0x0      0
eflags   0x206    [ PF IF ]  cs      0x23     35
ds       0x2b     43      es      0x2b     43
gs       0x0      0

B+ 0x80490e8 <_start> mov ebx,0x3
0x80490ed <_start+5> mov eax,0x2
B+ 0x80490f2 <_start+10> add ebx,eax
0x80490f4 <_start+12> mov ecx,0x4
0x80490f9 <_start+17> mul ecx
> 0x80490fb <_start+19> add ebx,0x5
0x80490fe <_start+22> mov edi,ebx
0x8049100 <_start+24> mov eax,0x804a000
0x8049105 <_start+29> call 0x804900f <sprint>
0x804910a <_start+34> mov eax,edi
0x804910c <_start+36> call 0x8049086 <lprintf>
0x8049111 <_start+41> call 0x80490db <quit>
0x8049116 add BYTE PTR [eax],al
0x8049118 add BYTE PTR [eax],al
0x804911a add BYTE PTR [eax],al
0x804911c add BYTE PTR [eax],al
0x804911e add BYTE PTR [eax],al

native process 24813 In: _start
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43
ds       0x2b     43
es       0x2b     43
fs       0x0      0
gs       0x0      0
(gdb) si
(gdb) si

Breakpoint 2, _start () at func2.asm:10
(gdb) si
(gdb) set $eax=5
(gdb) p/s $eax
$1 = 5
(gdb) si
(gdb) si
(gdb)

```

Рис. 3.13: Результат вычисления произведения

В результате в регистр `eax` было помещено значение произведения 20. Далее к этому значению нужно прибавить 5. В программе за результат отвечает регистр `ebx`. Поместим в него значение $20 + 5 = 25$ и запустим программу на вывод конечного результата (рис. 3.14).

```
eax    0x14    20    ecx    0x4    4
ebx    0x19    25    esp    0xffffd120    0xffffd120
esi    0x0    0    edi    0x0    0
eflags 0x206    [ PF IF ]    cs    0x23    35
ds     0x2b    43    es     0x2b    43
gs     0x0    0

B+ 0x80490e8 <_start>    mov    ebx,0x3
0x80490fe <_start+22>    mov    edi,ebx04a000
0x8049105 <_start+29>    call   0x804900f <sprint>
0x804910a <_start+34>    mov    eax,edi
0x804910c <_start+36>    call   0x8049086 <_printf@plt>
0x8049111 <_start+41>    call   0x80490db <_exit@plt>
0x8049116 <_start+46>    add    BYTE PTR [eax],al

native process 24813 In: _start
gs     No process In:    0
(gdb) si

Breakpoint 2, _start () at func2.asm:10
(gdb) si
(gdb) set $eax=5
(gdb) p/s $eax
$1 = 5
(gdb) si
(gdb) si
(gdb) si
(gdb) set $ebx=25
(gdb) p/s $ebx
$2 = 25
(gdb) c
Continuing.
Результат: 25
[Inferior 1 (process 24813) exited normally]
(gdb)
```

Рис. 3.14: Завершение выполнения программы

Как мы видим, с учётом всех изменений программа выдаёт верный результат. Теперь изменим код программы в файле func2.asm (рис. 3.15).

```
GNU nano 6.2
%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
mov ebx,eax
add ebx,5
mov edi,ebx

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

Рис. 3.15: Измененный текст программы в файле func2

Теперь создадим исполняемый файл и проверим корректность работы программы (рис. 3.16).

```
ramikhalova@ramikhalova:~/work/arch-pc/lab09$ nasm -f elf -g -l func2.lst func2.asm
ramikhalova@ramikhalova:~/work/arch-pc/lab09$ ld -m elf_i386 -o func2 func2.o
ramikhalova@ramikhalova:~/work/arch-pc/lab09$ ./func2
Результат: 25
ramikhalova@ramikhalova:~/work/arch-pc/lab09$
```

Рис. 3.16: Проверка работы программы

Теперь программа работает правильно.

4 Выводы

В ходе выполнения лабораторной работы я приобрела навыки написания программ с использованием подпрограмм. Познакомилась с методами отладки при помощи GDB и его основными возможностями.

Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.
5. Newham C. Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learning-bash-Shell-Programming-Nutshell/dp/0596009658>.
6. Robbins A. Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
8. Zarrelli G. Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. Колдаев В. Д., Лупин С. А. Архитектура ЭВМ. — М. : Форум, 2018.
10. Куляс О. Л., Никитин К. А. Курс программирования на ASSEMBLER. — М. : Солон-Пресс,
- 11.
12. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
13. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.
14. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВ- Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
15. Столяров А. Программирование на языке ассемблера NASM для ОС Unix. — 2-

- е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.
16. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
17. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер,
18. — 1120 с. — (Классика Computer Science).