

Отчёт по лабораторной работе №9

дисциплина: Архитектура компьютера

Габралян Георгий Александрович

Содержание

1	Цель работы	6
2	Выполнение лабораторной работы	7
3	Выполнение заданий для самостоятельной работы	22
4	Выводы	34
	Список литературы	35

Список иллюстраций

2.1	Создание файла lab09-1.asm	7
2.2	Текст lab09-1.asm	8
2.3	Создание и проверка работы файла lab09-1	9
2.4	Изменённый текст lab09-1.asm	9
2.5	Создание и запуск изменённого файла lab09-1	10
2.6	Текст lab09-2.asm	10
2.7	Создание исполняемого файла lab09-2 и файла листинга	11
2.8	Отладчик GDB	11
2.9	Запуск программы с помощью run	11
2.10	Установка брейкпоинта на метку _start	11
2.11	Дисассимилированный код программы	12
2.12	Синтаксис Intel	13
2.13	Режим псевдографики	14
2.14	Проверка точки останова	14
2.15	Установка точки останова по адресу	15
2.16	Инструкция stepi(1)	15
2.17	Инструкция stepi(2)	16
2.18	Инструкция stepi(3)	16
2.19	Инструкция stepi(4)	17
2.20	Инструкция stepi(5)	17
2.21	Значение переменной msg1 по имени	18
2.22	Значение переменной msg2 по адресу	18
2.23	Изменение символов в переменных msg1 и msg2	18
2.24	Вывод значения регистра edx	19
2.25	Изменение значения регистра ebx	19
2.26	Завершение выполнения программы	20
2.27	Создание исполняемого файла lab09-3	20
2.28	Загрузка программы lab09-3 в отладчике gdb	20
2.29	Установка точки останова и запуск программы	21
2.30	Содержимое регистра esp	21
2.31	Содержимое стека	21
3.1	Текст f1.asm	23
3.2	Создание и запуск исполняемого файла f1	23
3.3	Запуск исполняемого файла f2	24
3.4	Загрузка файла f2 в отладчик gdb	24
3.5	Дисассимилированный код f2	25

3.6	Режим псевдографики f2	25
3.7	Просмотр значений регистров f2	26
3.8	Точка останова на инструкции add	26
3.9	Просмотр значений регистров	27
3.10	si	28
3.11	Изменение значения регистра eax	29
3.12	si	30
3.13	Результат вычисления	31
3.14	Завершение выполнения программы	32
3.15	Исправленный текст программы в f2.asm	33
3.16	Проверка работы программы func2	33

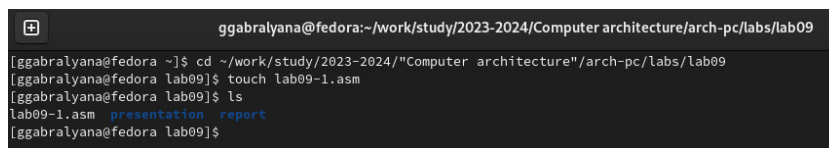
Список таблиц

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

Создаём каталог для выполнения лабораторной работы номер 9, переходим в него и создаём файл `lab09-1.asm`. (рис. 2.1)



```
ggabralyana@fedora:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab09
[ggabralyana@fedora ~]$ cd ~/work/study/2023-2024/"Computer architecture"/arch-pc/labs/lab09
[ggabralyana@fedora lab09]$ touch lab09-1.asm
[ggabralyana@fedora lab09]$ ls
lab09-1.asm  presentation  report
[ggabralyana@fedora lab09]$
```

Рис. 2.1: Создание файла `lab09-1.asm`

Рассматриваем программу вычисления арифметического выражения $f(x) = 2x + 7$. Вводим текст этой программы в файл `lab09-1.asm`. (рис. 2.2)

```

#include 'in_out.asm

SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
res: RESB 80

SECTION .text
GLOBAL _start
_start:

mov eax, msg
call sprint

mov ecx, x
mov edx, 80
call sread

mov eax, x
call atoi

call _calcul ; Вызов подпрограммы _calcul

mov eax, result
call sprint
mov eax, [res]
call iprintLF

call quit

_calcul:
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax

ret ; выход из подпрограммы
[ggabralyana@fedora lab09]$

```

Рис. 2.2: Текст lab09-1.asm

Создаём исполняемый файл и сразу проверяем его работу. (рис. 2.3)


```

[ggabralyana@fedora lab09]$ nasm -f elf lab09-1.asm
lab09-1.asm:1: warning: unterminated string [-w+other]
[ggabralyana@fedora lab09]$ ls
in_out.asm  lab09-1.asm  lab09-1.o  presentation  report
[ggabralyana@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[ggabralyana@fedora lab09]$ ls
in_out.asm  lab09-1  lab09-1.asm  lab09-1.o  presentation  report
[ggabralyana@fedora lab09]$ ./lab09-1
Введите x: 1
2x+7=9
[ggabralyana@fedora lab09]$

```

Рис. 2.3: Создание и проверка работы файла lab09-1

Теперь изменяем текст программы, добавляя подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$. (рис. 2.4)

```

[ggabralyana@fedora ~]$ cat ~/work/study/2023-2024/"Computer architecture"/arch-pc/labs/lab09/lab09-1.asm
#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:
; 2x+7
call _subcalcul
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы f(g(x))

_subcalcul:
; 3x-1
mov ebx, 3
mul ebx
dec eax ; eax=3x-1
ret ; выход из подпрограммы g(x)
[ggabralyana@fedora ~]$ S

```


Рис. 2.4: Изменённый текст lab09-1.asm

Создаём исполняемый файл, проверяем его работу. (рис. 2.5)

```
[ggabralyana@fedora lab09]$ nasm -f elf lab09-1.asm
lab09-1.asm:1: warning: unterminated string [-w+other]
[ggabralyana@fedora lab09]$ ld -m elf_i386 -o lab09-1 lab09-1.o
[ggabralyana@fedora lab09]$ ./lab09-1
Введите x: 1
2(3x-1)+7=11
[ggabralyana@fedora lab09]$ ./lab09-1
Введите x: 5
2(3x-1)+7=35
[ggabralyana@fedora lab09]$ ./lab09-1
Введите x: 9
2(3x-1)+7=59
[ggabralyana@fedora lab09]$
```

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

Создаём файл lab09-2.asm, заполняем его текстом программы печати сообщения *Hello world!* (рис. 2.6)



```
*lab09-2.asm
~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab09

1 SECTION .data
2
3 msg1: db "Hello, ",0x0
4 msg1Len: equ $ - msg1
5
6 msg2: db "world!",0xa
7 msg2Len: equ $ - msg2
8
9 SECTION .text
10 global _start
11
12 _start:
13 mov eax, 4
14 mov ebx, 1
15 mov ecx, msg1
16 mov edx, msg1Len
17 int 0x80
18
19 mov eax, 4
20 mov ebx, 1
21 mov ecx, msg2
22 mov edx, msg2Len
23 int 0x80
24
25 mov eax, 1
26 mov ebx, 0
27 int 0x80
```

Рис. 2.6: Текст lab09-2.asm

Получаем исполняемый файл. Для работы с GDB в исполняемый файл нужно добавить отладочную информацию, для этого используем ключ `-g`. (рис. 2.7)

```

[ggabralyana@fedora lab09]$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
[ggabralyana@fedora lab09]$ ld -m elf_1386 -o lab09-2 lab09-2.o
[ggabralyana@fedora lab09]$ ls
in_out.asm lab09-1 lab09-1.asm lab09-1.o lab09-2 lab09-2.asm lab09-2.lst lab09-2.o presentation report
[ggabralyana@fedora lab09]$

```

Рис. 2.7: Создание исполняемого файла lab09-2 и файла листинга

Загружаем исполняемый файл в отладчик GDB. (рис. 2.8)

```

[ggabralyana@fedora lab09]$ gdb lab09-2
GNU gdb (GDB) Fedora 12.1-4.fc37
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-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

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

```

Рис. 2.8: Отладчик GDB

Проверяем работу программы, запуская её в GDB с помощью команды run (рис. 2.9).

```

(gdb) run
Starting program: /home/ggabralyana/work/study/2023-2024/Computer_architecture/arch-pc/labs/lab09/lab09-2
Hello, world!
[Inferior 1 (process 9916) exited normally]

```

Рис. 2.9: Запуск программы с помощью run

Для более детального анализа программы устанавливаем брейкпоинт на метку _start, запускаем её. (рис. 2.10)

```

(gdb) break _start
Breakpoint 1 at 0x8040000: file lab09-2.asm, line 13.
(gdb) run
Starting program: /home/ggabralyana/work/study/2023-2024/Computer_architecture/arch-pc/labs/lab09/lab09-2
Breakpoint 1, _start () at lab09-2.asm:13
13      mov eax, 4

```

Рис. 2.10: Установка брейкпоинта на метку _start

Далее смотрим дисассимилированный код программы с помощью команды `disassemble` начиная с `_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

Переключаемся на отображение команд с синтаксисом 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

Как можно увидеть, в синтаксисе Intel после команды выводится регистр, куда помещается значение, и затем адрес помещаемого значения или число. В свою очередь, в синтаксисе АТТ сначала выводится ссылка на заносимое в регистр значение, и только после неё регистр с символом %.

Включаем режим псевдографики для более удобного анализа программы с помощью `layout asm` и `layout regs` (рис. 2.13)

```
ggabralyana@fedora:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab09 — gdb lab09-2
[ Register Values Unavailable ]

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,0x804a000
0x8049025 <_start+37> mov edx,0x7
0x804902a <_start+42> int 0x80
0x804902c <_start+44> mov eax,0x1
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 10000 In: _start L13 PC: 0x8049000
(gdb) layout regs
(gdb)
```

Рис. 2.13: Режим псевдографики

Ранее была установлена точка останова по имени метки `_start`. Проверяем это с помощью команды `info breakpoints` (рис. 2.14)

```
0x804903a add BYTE PTR [eax],al
0x804903c add BYTE PTR [eax],al
0x804903e add BYTE PTR [eax],al
native process 10000 In: _start
(gdb) layout regs
(gdb) info breakpoints
Num   Type             Disp Enb Address      What
1     breakpoint       keep y  0x08049000 lab09-2.asm:13
      breakpoint already hit 1 time
(gdb)
```

Рис. 2.14: Проверка точки останова

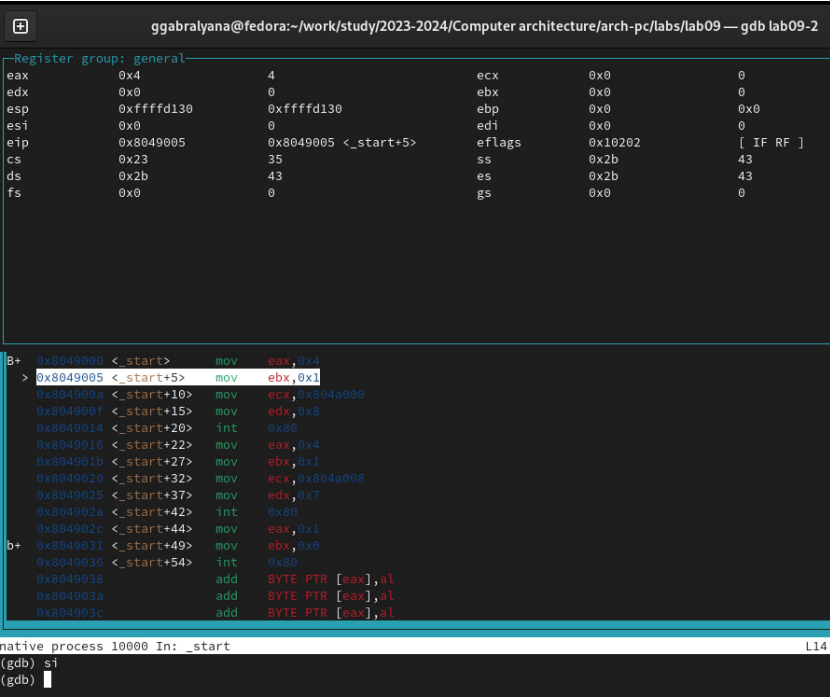
Устанавливаем еще одну точку останова по адресу инструкции `mov ebx, 0x0`. В

нашем случае это адрес: 0x8049031. Затем смотрим информацию о всех точках останова (рис. 2.15).

```
(gdb) disassemble _start
(gdb) b *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 26.
(gdb) i b
Num      Type             Disp Enb Address      What
1        breakpoint       keep y 0x08049000 lab09-2.asm:13
          breakpoint already hit 1 time
2        breakpoint       keep y 0x08049031 lab09-2.asm:26
(gdb)
```

Рис. 2.15: Установка точки останова по адресу

Выполняем 5 инструкций с помощью команды si и следим за изменениями в значениях регистров (рис. 2.16 - 2.20).



The screenshot shows the GDB interface with the following content:

Register group: general

Register	Value	Register	Value	Register	Value
eax	0x4	ecx	0x0	0	
edx	0x0	ebx	0x0	0	
esp	0xffffd130	ebp	0x0	0x0	
esi	0x0	edi	0x0	0	
eip	0x8049005	eip	0x8049005	<_start+5>	
cs	0x23	ss	0x2b	43	
ds	0x2b	es	0x2b	43	
fs	0x0	gs	0x0	0	

Assembly instructions:

```
B+ 0x8049005 <_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
```

native process 1000 In: _start L14

(gdb) si

(gdb)

Рис. 2.16: Инструкция stepi(1)

```

ggabralyana@fedora:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab09 — gdb lab09-2
Register group: general
eax    0x4      4      ecx    0x0      0
edx    0x0      0      ebx    0x1      1
esp    0xffffd130 0xffffd130 ebp    0x0      0x0
esi    0x0      0      edi    0x0      0
eip    0x804900a 0x804900a <_start+10> eflags 0x10202 [ IF RF ]
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,0x0
0x8049014 <_start+20> int    0x80
0x8049016 <_start+22> mov    eax,0x4
0x804901b <_start+27> mov    ebx,0x1
0x8049020 <_start+32> mov    ecx,0x804a000
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

native process 10000 In: _start L15 PC: 0x804900a
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.17: Инструкция stepi(2)

```

ggabralyana@fedora:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab09 — gdb lab09-2
Register group: general
eax    0x4      4      ecx    0x804a000 134520832
edx    0x0      0      ebx    0x1      1
esp    0xffffd130 0xffffd130 ebp    0x0      0x0
esi    0x0      0      edi    0x0      0
eip    0x804900f 0x804900f <_start+15> eflags 0x10202 [ IF RF ]
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,0x0
0x8049011 <_start+20> int    0x80
0x8049016 <_start+22> mov    eax,0x4
0x804901b <_start+27> mov    ebx,0x1
0x8049020 <_start+32> mov    ecx,0x804a000
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

native process 10000 In: _start L16 PC: 0x804900f
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.18: Инструкция stepi(3)


```

ggabralyana@fedora:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab09 — gdb lab09-2

--Register group: general--
eax      0x4      4      ecx      0x804a000      134520832
edx      0x8      8      ebx      0x1      1
esp      0xffffd130      0xffffd130      ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049014      0x8049014 <_start+20>      eflags      0x10202      [ IF RF ]
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,0x804a000
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

native process 10000 In: _start      L17      PC: 0x8049014
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.19: Инструкция stepi(4)

```

ggabralyana@fedora:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab09 — gdb lab09-2

--Register group: general--
eax      0x8      8      ecx      0x804a000      134520832
edx      0x8      8      ebx      0x1      1
esp      0xffffd130      0xffffd130      ebp      0x0      0x0
esi      0x0      0      edi      0x0      0
eip      0x8049016      0x8049016 <_start+22>      eflags      0x202      [ 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
> 0x8049016 <_start+22>     mov     eax,0x4
0x804901b <_start+27>     mov     ebx,0x1
0x8049020 <_start+32>     mov     ecx,0x804a000
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

native process 10000 In: _start      L19      PC: 0x8049016
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.20: Инструкция stepi(5)

Как видно на верхней панели, изменились значения регистров eax, ecx, ebx и

edx.

Теперь смотрим значение переменной `msg1` по имени (рис. 2.21).

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

Рис. 2.21: Значение переменной `msg1` по имени

Выводится строка “Hello,”.

Теперь смотрим значение переменной `msg2` по адресу. Адрес переменной определяем по дисассемблированной инструкции. (рис. 2.22).

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

Рис. 2.22: Значение переменной `msg2` по адресу

Далее изменяем первый символ в переменных `msg1` и `msg2` (рис. 2.23).

```
(gdb) set {char}&msg1='h'
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hello, "
(gdb) set {char}&msg2='W'
(gdb) x/1sb &msg2
0x804a008 <msg2>:      "World!\n\034"
(gdb)
```

Рис. 2.23: Изменение символов в переменных `msg1` и `msg2`

Выводим в различных форматах значение регистра `edx` (рис. 2.24).

```

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

```

Рис. 2.24: Вывод значения регистра edx

И теперь, используя команду `set`, изменяем значение регистра `ebx` (рис. 2.25).

```

(gdb) set $ebx='2'
(gdb) p/s $ebx
$5 = 50
(gdb) set $ebx=2
(gdb) p/s ebx
No symbol "ebx" in current context.
(gdb) p/s $ebx
$6 = 2

```

Рис. 2.25: Изменение значения регистра ebx

Сначала мы занесли в регистр `ebx` символ '2', вот почему после запроса `p/s` на вывод значения регистра на экране мы увидели код символа "2", то есть 50. А затем, когда число 2 изначально было занесено в регистр, команда `p/s $ebx` вывела значение 2.

Завершаем выполнение программы с помощью `c` (рис. 2.26).

```
(gdb) c
Continuing.
World!

Breakpoint 2, _start () at lab09-2.asm:26
(gdb) █
```

Рис. 2.26: Завершение выполнения программы

И сразу после этого с помощью команды `quit` выходим из отладчика `gdb`.

Копируем файл `lab8-2.asm`, который был создан при выполнении лабораторной работы 8, в файл с именем `lab09-3.asm` и создаём исполняемый файл с ключом `-g` и файлом листинга (рис. 2.27).

```
[ggabralyana@fedora lab09]$ nasm -f elf -g -l lab09-3.lst lab09-3.asm
[ggabralyana@fedora lab09]$ ld -m elf_i386 -o lab09-3 lab09-3.o
[ggabralyana@fedora lab09]$ ls
in_out.asm lab09-1.asm lab09-2 lab09-2.lst lab09-3 lab09-3.lst presentation
lab09-1 lab09-1.o lab09-2.asm lab09-2.o lab09-3.asm lab09-3.o report
[ggabralyana@fedora lab09]$ █
```

Рис. 2.27: Создание исполняемого файла `lab09-3`

Для того чтобы загрузить в GDB программы с аргументами нужно использовать ключ `--args`. Загружаем файл в отладчик, указав аргументы (рис. 2.28).

```
[ggabralyana@fedora lab09]$ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
GNU gdb (GDB) Fedora 12.1-4.fc37
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-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

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

Рис. 2.28: Загрузка программы `lab09-3` в отладчике `gdb`

Сначала устанавливаем точку останова перед первой инструкцией в программе, запускаем её (рис. 2.29).

```
(gdb) b _start
Breakpoint 1 at 0x00400000: file lab09-3.asm, line 7.
(gdb) run
Starting program: /home/ggabrallyana/work/study/2023-2024/Computer architecture/arch-pc/labs/lab09/lab09-3 аргумент1 аргумент 2 а
ргумент\ 3
This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Breakpoint 1, _start () at lab09-3.asm:7
7      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb)
```

Рис. 2.29: Установка точки останова и запуск программы

Адрес вершины стека хранится в регистре `esp` и по нему располагается число, которое равно количеству аргументов командной строки (рис. 2.30).

```
(gdb) x/x $esp
0xffffd0e0:      0x00000005
(gdb)
```

Рис. 2.30: Содержимое регистра `esp`

Число аргументов равно 5, то есть имя программы `lab09-3` и сами аргументы: `аргумент1`, `аргумент, 2` и `аргумент 3`.

Просматриваем остальные позиции (рис. 2.31).

```
(gdb) x/s *(void**)($esp + 4)
0xffffd28e:      "/home/ggabrallyana/work/study/2023-2024/Computer architecture/arch-pc/labs/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd2ae:      "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffd2f8:      "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffd308:      "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd308:      "аргумент 3"
(gdb) x/s *(void**)($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 так далее. Как можно заметить, шаг изменения равен 4. Шаг имеет такое значение, так как при добавлении значения каждого аргумента в стек, значение регистра `esp` увеличивается на 4.

3 Выполнение заданий для самостоятельной работы

Задание №1

Преобразовываем программу из лабораторной работы 8, реализовывая вычисление значения функции $f(x)$ как подпрограмму.

В лабораторной работе 8 мы реализовывали вычисление в цикле следующей функции:

$$f(x) = 6x + 13$$

Теперь для вычисления значения этой функции в цикле мы вызываем вспомогательную подпрограмму. Создаём файл `f1.asm` и вписываем в него текст программы (рис. 3.1).

```

%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 ;используем esi для хранения промежуточных сумм f(x)

;6x+13
next:
    cmp ecx,0h ;проверяем, есть ли ещё аргументы
    jz _end ;если их нет, выходим из цикла
    pop eax
    call _calcul ;вызов подпрограммы вычисления функции
    add esi,eax ;помещаем значение f(x) из eax в esi
    loop next

_end:
    mov eax,msg
    call sprint
    mov eax,esi
    call iprintLF ;печать результата (суммы)
    call quit

_calcul:
    call atoi ;преобразуем символ в число
    mov ebx,6
    mul ebx;умножаем след.аргумент на 6 'eax=eax*6'
    add eax,13;eax=eax+13

```

Рис. 3.1: Текст f1.asm

Создаём исполняемый файл, проверяем его работу с теми же аргументами (рис. 3.2).

```

[ggabralyana@fedora lab09]$ nasm -f elf f1.asm
[ggabralyana@fedora lab09]$ ld -m elf_i386 -o f1 f1.o
[ggabralyana@fedora lab09]$ ./f1 4 6 8
Результат: 147
[ggabralyana@fedora lab09]$ ./f1 3 0 7 4
Результат: 136
[ggabralyana@fedora lab09]$ ./f1 1 4 6 8 2
Результат: 191
[ggabralyana@fedora lab09]$

```

Рис. 3.2: Создание и запуск исполняемого файла f1

Как видно, программа работает исправно.

Задание №2

Создаём файл `f2.asm` и вписываем в него текст программы вычисления выражения $(3 + 2) * 4 + 5$. Затем создаём исполняемый файл и запускаем его (рис. 3.3).

```
[ggabralyana@fedora lab09]$ nasm -f elf -g -l f2.lst f2.asm
[ggabralyana@fedora lab09]$ ld -m elf_i386 -o f2 f2.o
[ggabralyana@fedora lab09]$ ./f2
Результат: 10
[ggabralyana@fedora lab09]$
```

Рис. 3.3: Запуск исполняемого файла `f2`

Как можно увидеть, при запуске программа действительно даёт неверный результат.

С помощью отладчика GDB анализируем изменения значений регистров, определяем ошибку и исправляем её.

Для начала загрузим исполняемый файл в отладчик (рис. 3.4).

```
[ggabralyana@fedora lab09]$ gdb f2
GNU gdb (GDB) Fedora 12.1-4.fc37
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-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from f2...
(gdb)
```

Рис. 3.4: Закрузка файла `f2` в отладчик `gdb`

Смотрим дисассимилированный код программы, начиная с метки `_start` (рис. 3.5).


```

(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    0x804900f <sprint>
   0x0804910a <+34>:     mov     %edi,%eax
   0x0804910c <+36>:     call    0x8049086 <iprintf>
   0x08049111 <+41>:     call    0x80490db <quit>
End of assembler dump.
(gdb)

```

Рис. 3.5: Дисассимилированный код f2

Переходим в режим псевдографики (рис. 3.6).

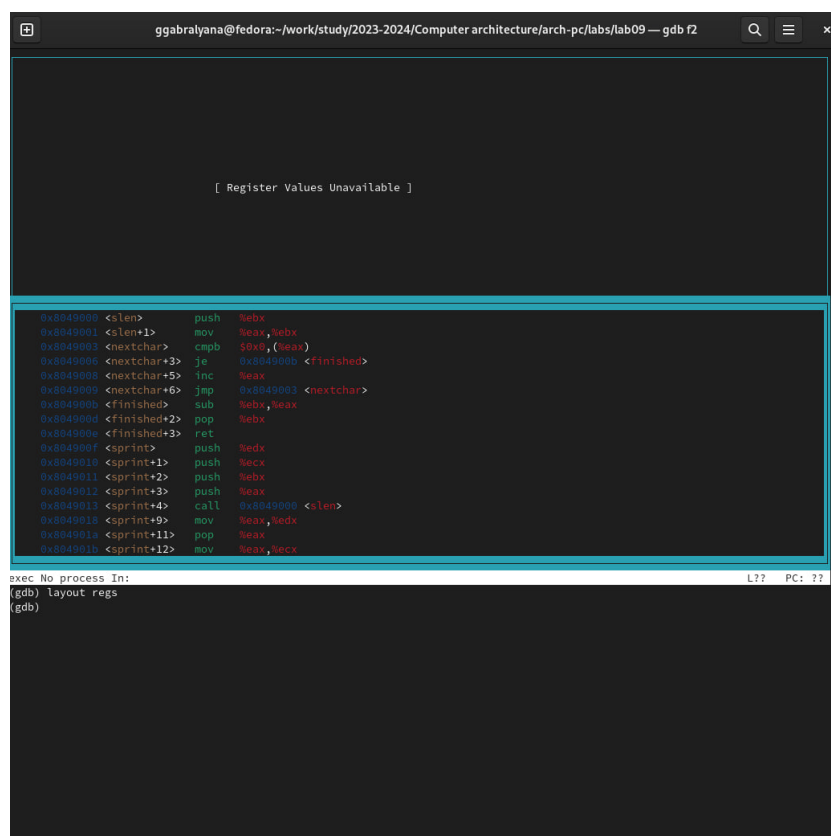
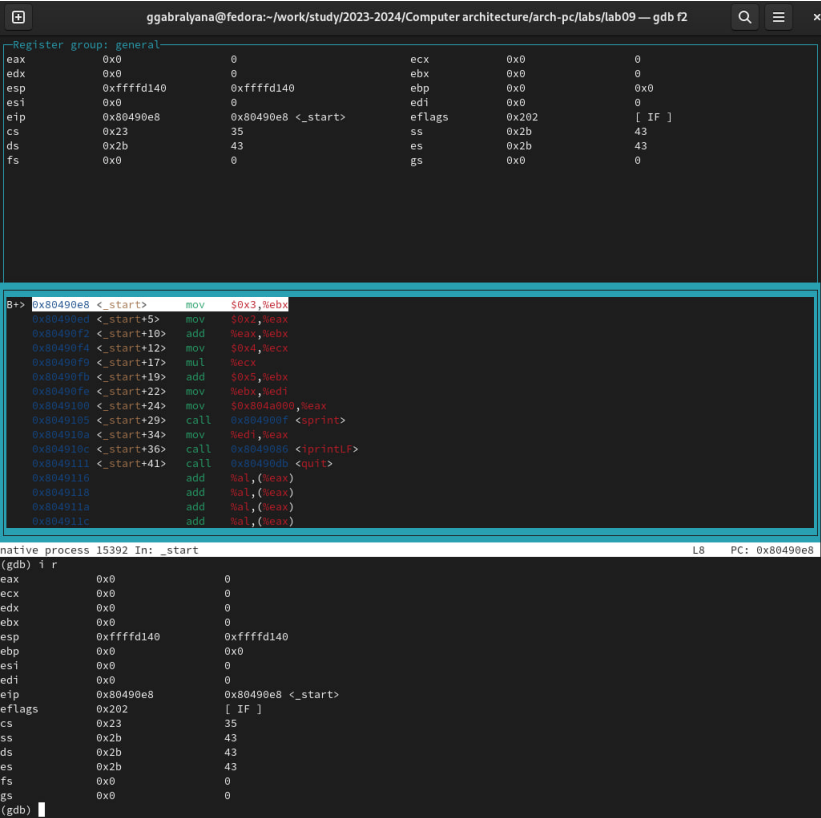


Рис. 3.6: Режим псевдографики f2


Просматриваем значения регистров (рис. 3.7).



The screenshot shows the GDB interface. The top panel displays the 'Register group: general' with values for registers: eax (0x0), ebx (0x0), ecx (0x0), edx (0x0), esp (0xffffd140), ebp (0xffffd140), esi (0x0), edi (0x0), eip (0x80490e8), eflags (0x202), cs (0x23), ss (0x2b), ds (0x2b), and fs (0x0). The middle panel shows assembly code starting at address 0x80490e8, with instructions like 'mov \$0x3,%ebx', 'mov \$0x2,%eax', 'add %eax,%ebx', etc. The bottom panel shows the 'native process 15392 In: _start' with a list of registers and their values, including 'eax: 0x0', 'ecx: 0x0', 'edx: 0x0', 'ebx: 0x0', 'esp: 0xffffd140', 'ebp: 0x0', 'esi: 0x0', 'edi: 0x0', 'eip: 0x80490e8', 'eflags: 0x202', 'cs: 0x23', 'ss: 0x2b', 'ds: 0x2b', 'fs: 0x0', and 'gs: 0x0'.

Рис. 3.7: Просмотр значений регистров f2

Воспользовавшись командой `break` устанавливаем точку останова по адресу на инструкции `add ebx, eax` (рис. 3.8).



The screenshot shows the GDB command prompt with the command `(gdb) b *0x80490f2` and the response `Breakpoint 2 at 0x80490f2: file f2.asm, line 10.` followed by `(gdb)`.

Рис. 3.8: Точка останова на инструкции `add`

Смотрим значения регистров с помощью команды `i r` (рис. 3.9).

The screenshot shows the GDB interface with the following content:

Register group: general

eax	0x2	2	ecx	0x0	0
edx	0x0	0	ebx	0x3	3
esp	0xffffd140	0xffffd140	ebp	0x0	0x0
esi	0x0	0	edi	0x0	0
ebp	0x80490f2	0x80490f2 <_start+10>	eiflags	0x202	[IF]
cs	0x23	35	ss	0x2b	43
ds	0x2b	43	es	0x2b	43
fs	0x0	0	gs	0x0	0

Assembly code:

```

B+ 0x80490e8 <_start> mov $0x1,%ebx
0x80490e9 <_start+5> mov $0x2,%ecx
B+ 0x80490f2 <_start+10> add %ebx,%ebx
0x80490f4 <_start+12> mov %ecx,%ecx
0x80490f5 <_start+17> mul %ecx
0x80490f6 <_start+19> add $0x5,%ebx
0x80490fe <_start+22> mov %ebx,%edi
0x8049100 <_start+24> mov $0x804a00b,%eax
0x8049105 <_start+29> call 0x804900f <sprintf>
0x804910a <_start+34> mov %edi,%eax
0x804910c <_start+36> call 0x8049006 <printf>
0x8049111 <_start+41> call 0x804900b <quit>
0x8049116 add %al,(%eax)
0x804911a add %al,(%eax)
0x804911c add %al,(%eax)
  
```

Native process 15392 In: _start L10 PC: 0x80490f2

Breakpoint 2, _start () at f2.asm:10

(gdb) i r

eax	0x2	2
ecx	0x0	0
edx	0x0	0
ebx	0x3	3
esp	0xffffd140	0xffffd140
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
ebp	0x80490f2	0x80490f2 <_start+10>
eiflags	0x202	[IF]
cs	0x23	35
ss	0x2b	43
ds	0x2b	43
es	0x2b	43
fs	0x0	0
gs	0x0	0

(gdb)

Рис. 3.9: Просмотр значений регистров

Затем с помощью `si` переходим к следующей инструкции и следим за изменениями значений регистров (рис. 3.10).

```

ggabralyana@fedora:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab09 — gdb f2

Register group: general
eax 0x2 2 ecx 0x0 0
edx 0x0 0 ebx 0x5 5
esp 0xffffd140 0xffffd140 ebp 0x0 0
esi 0x0 0 edi 0x0 0
eip 0x80490f4 0x80490f4 <_start+12> eflags 0x10206 [ PF IF RF ]
cs 0x23 35 ss 0x2b 43
ds 0x2b 43 es 0x2b 43
fs 0x0 0 gs 0x0 0

B+ 0x80490e8 <_start> mov $0x3, %ebx
0x80490ed <_start+5> mov $0x7, %eax
B+ 0x80490f2 <_start+10> add %eax, %ebx
> 0x80490f4 <_start+12> mov $0x4, %ecx
0x80490f9 <_start+17> mul %ecx
0x80490fb <_start+19> add $0x9, %ebx
0x80490fe <_start+22> mov %ebx, %edi
0x8049108 <_start+24> mov $0x804a009, %eax
0x8049105 <_start+29> call 0x80490ef <sprint>
0x804910a <_start+34> mov %edi, %eax
0x804910c <_start+36> call 0x8049086 <fprintf>
0x8049111 <_start+41> call 0x80490db <quit>
0x8049116 add $0x1, (%eax)
0x8049118 add $0x1, (%eax)
0x804911a add $0x1, (%eax)
0x804911c add $0x1, (%eax)

native process 15392 In: _start L11 PC: 0x80490f4
(gdb) i r
eax 0x2 2
ecx 0x0 0
edx 0x0 0
ebx 0x3 3
esp 0xffffd140 0xffffd140
ebp 0x0 0
esi 0x0 0
edi 0x0 0
eip 0x80490f2 0x80490f2 <_start+10>
eflags 0x202 [ IF ]
cs 0x23 35
ss 0x2b 43
ds 0x2b 43
es 0x2b 43
fs 0x0 0
gs 0x0 0
(gdb) si
(gdb)

```

Рис. 3.10: si

Результат суммы чисел 2 и 3 записался в регистр ebx. Это и могло послужить проблемой для дальнейшего вычисления произведения. Исправляем это, изменяя значение регистра eax и занеся в него значение 5 с помощью set (рис. 3.11).

```
Обзор Терминал C6, 9 дек
ggabralyana@fedora: ~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab09 — gdb f2

Register group: general
eax    0x5      5      ecx    0x0      0
edx    0x0      0      ebx    0x5      5
esp    0xffffd140 0xffffd140 ebp    0x0      0x0
esi    0x0      0      edi    0x0      0
eip    0x80490f4 0x80490f4 <_start+12> eflags 0x10206 [ PF IF RF ]
cs     0x23     35     ss     0x2b     43
ds     0x2b     43     es     0x2b     43
fs     0x0      0      gs     0x0      0

B+ 0x80490e0 <_start> mov $0x3,%ebx
0x80490e2 <_start+2> mov $0x2,%eax
B+ 0x80490f2 <_start+10> add $0x5,%ebx
> 0x80490f4 <_start+12> mov $0x4,%ecx
0x80490f6 <_start+14> mul %ecx
0x80490f8 <_start+16> add $0x5,%ebx
0x80490fa <_start+18> mov %ebx,%edi
0x80490fc <_start+20> mov $0x804900,%eax
0x80490fe <_start+22> call 0x804900f <sprint>
0x8049100 <_start+24> mov %edi,%eax
0x8049102 <_start+26> call 0x8049080 <printf>
0x8049104 <_start+28> call 0x8049080 <printf>
0x8049106 <_start+30> add %al,(%eax)
0x8049108 <_start+32> add %al,(%eax)
0x804910a <_start+34> add %al,(%eax)
0x804910c <_start+36> add %al,(%eax)
0x804910e <_start+38> add %al,(%eax)
0x8049110 <_start+40> add %al,(%eax)
0x8049112 <_start+42> add %al,(%eax)
0x8049114 <_start+44> add %al,(%eax)
0x8049116 <_start+46> add %al,(%eax)
0x8049118 <_start+48> add %al,(%eax)
0x804911a <_start+50> add %al,(%eax)
0x804911c <_start+52> add %al,(%eax)

native process 15392 In: _start L11 PC: 0x80490f4
edx    0x0      0
ebx    0x3      3
esp    0xffffd140 0xffffd140
ebp    0x0      0
esi    0x0      0
edi    0x0      0
eip    0x80490f2 0x80490f2 <_start+10>
eflags 0x202     [ IF ]
cs     0x23     35
ss     0x2b     43
ds     0x2b     43
es     0x2b     43
fs     0x0      0
gs     0x0      0
(gdb) s1
(gdb) set $eax=5
(gdb) p/s $eax
$1 = 5
(gdb)
```

Рис. 3.11: Изменение значения регистра eax

Далее переходим к следующей инструкции (рис. 3.12).

```

ggabralyana@fedora:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab09 — gdb f2

-Register group: general-
eax    0x5      5      ecx    0x4      4
edx    0x0      0      ebx    0x5      5
esp    0xffffd140 0xffffd140  ebp    0x0      0x0
esi    0x0      0      edi    0x0      0
ebp    0x80490f9 0x80490f9 <_start+17>  eflags 0x10206   [ PF IF RF ]
cs     0x23     35     ss     0x2b     43
ds     0x2b     43     es     0x2b     43
fs     0x0      0      gs     0x0      0

B+ 0x80490e8 <_start>   mov     $0x3,%ebx
0x80490ed <_start+5>   mov     $0x2,%eax
B+ 0x80490f2 <_start+10> add     %eax,%ebx
0x80490f7 <_start+12>   mov     $0x1,%ecx
> 0x80490f9 <_start+17> mul     %ecx
0x80490fd <_start+19>   add     $0x1,%ebx
0x80490fe <_start+22>   mov     %ebx,%edi
0x8049100 <_start+24>   mov     $0x040800,%eax
0x8049105 <_start+29>   call    0x80490ef <_sprintf>
0x804910a <_start+34>   mov     %edi,%eax
0x804910c <_start+36>   call    0x8049086 <_printf>
0x8049111 <_start+41>   call    0x80490db <_quit>
0x8049116             add     %al,(%eax)
0x8049118             add     %al,(%eax)
0x804911a             add     %al,(%eax)
0x804911c             add     %al,(%eax)

native process 15392 In: _start L12 PC: 0x80490f9
ebx     0x3      3
esp     0xffffd140 0xffffd140
ebp     0x0      0x0
esi     0x0      0
edi     0x0      0
eip     0x80490f2 0x80490f2 <_start+10>
eflags  0x202    [ IF ]
cs      0x23     35
ss      0x2b     43
ds      0x2b     43
es      0x2b     43
fs      0x0      0
gs      0x0      0
(gdb) si
(gdb) set $eax=5
(gdb) p/s $eax
$1 = 5
(gdb) si
(gdb)

```

Рис. 3.12: si

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

```

ggabralyana@fedora:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab09 — gdb f2

Register group: general
eax    0x14    20    ecx    0x4    4
edx    0x0    0    ebx    0x5    5
esp    0xffffd140 0xffffd140  ebp    0x0    0x0
esi    0x0    0    edi    0x0    0
eip    0x80490fb 0x80490fb <_start+19>  eflags  0x10206  [ PF IF RF ]
cs     0x23    43    ss     0x2b    43
ds     0x2b    43    es     0x2b    43
fs     0x0    0    gs     0x0    0

0x80490e8 <_start>    mov    $0x3,%ebx
0x80490ed <_start+5>    mov    $0x2,%eax
0x80490f2 <_start+10>   add    %eax,%ebx
0x80490fa <_start+12>   mov    $0x4,%ecx
0x80490fb <_start+17>   mul    %ecx
> 0x80490fb <_start+19> add    $0x5,%ebx
0x80490fe <_start+22>   mov    %ebx,%edi
0x8049100 <_start+24>   mov    $0x040000,%eax
0x8049105 <_start+29>   call  0x804900f <sprintf>
0x804910a <_start+34>   mov    %edi,%eax
0x804910c <_start+36>   call  0x8049006 <printf>
0x8049111 <_start+41>   call  0x80490db <_quit>
0x8049116    add    %al,(%eax)
0x8049118    add    %al,(%eax)
0x804911a    add    %al,(%eax)
0x804911c    add    %al,(%eax)

native process 15392 In: _start L13 PC: 0x80490fb
esp    0xffffd140 0xffffd140
ebp    0x0    0x0
esi    0x0    0
edi    0x0    0
eip    0x80490f2 0x80490f2 <_start+10>
eflags 0x202    [ IF ]
cs     0x23    35
ss     0x2b    43
ds     0x2b    43
es     0x2b    43
fs     0x0    0
gs     0x0    0
(gdb) si
(gdb) set $eax=5
(gdb) p/s $eax
$1 = 5
(gdb) si
(gdb) si
(gdb)

```

Рис. 3.13: Результат вычисления

В результате в регистр `eax` было помещено значение произведения 20.

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).

```
ggabralyana@fedora:~/work/study/2023-2024/Computer architecture/arch-pc/labs/lab09 — gdb f2

eax    0x14    20      ecx    0x4      4
edx    0x0      0      ebx    0x19    25
esp    0xffffd140 0xffffd140  ebp    0x0      0
esi    0x0      0      edi    0x0      0
eip    0x80490fe 0x80490fe <_start+22>  eflags 0x10206    [ PF IF RF ]
cs     0x23     35      ss     0x2b     43
ds     0x2b     43      es     0x2b     43
fs     0x0      0      gs     0x0      0

B+ 0x80490e8 <_start> mov    $0x1,%ebx
0x80490fe <_start+22> mov    %ebx,%edi,%eax
0x8049105 <_start+29> call  0x804900f <sprint>
0x804910a <_start+34> mov    %edi,%eax
0x804910c <_start+36> call  0x8049006 <iprintf>
0x8049111 <_start+41> call  0x80490db <quit>
0x8049116      add    %al,(%eax)

native process 15392 In: _start
cs     No process in: 35      L14 PC: 0x80490fe
ds     0x2b     43      L2? PC: ??
ss     0x2b     43
es     0x2b     43
fs     0x0      0
gs     0x0      0
(gdb) si
(gdb) set $eax=5
(gdb) p/s $eax
$1 = 5
(gdb) si
(gdb) si
(gdb) set $ebx=25
(gdb) p/s $ebx
$2 = 25
(gdb) c
Continuing.
Результат: 25
Inferior 1 (process 15392) exited normally]
(gdb)
```

Рис. 3.14: Завершение выполнения программы

Как видно, с учётом всех изменений программа выдаёт верный результат. Теперь изменяем код программы в файле `f2.asm` (рис. 3.15).


```

#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 ;4*5
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: Исправленный текст программы в f2.asm

Теперь создаём исполняемый файл и проверяем корректность работы программы (рис. 3.16).

```

[ggabralyana@fedora lab09]$ nasm -f elf -g -l f2.lst f2.asm
[ggabralyana@fedora lab09]$ ld -m elf_i386 -o f2 f2.o
[ggabralyana@fedora lab09]$ ./f2
Результат: 25
[ggabralyana@fedora lab09]$

```

Рис. 3.16: Проверка работы программы func2

Программа выдаёт верный результат.

4 Выводы

В ходе выполнения лабораторной работы мы приобрели навыки написания программ с использованием подпрограмм. Также мы познакомились 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/Learningbash-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.