

Отчёта по лабораторной работе 10

Понятие подпрограммы. Отладчик GDB.

Герра Гарсия Паола Валентина НКАбд-
05-22

Содержание

1	Цель работы	5
2	Задание	6
3	Теоретическое введение	7
4	Выполнение лабораторной работы	8
5	Выводы	30
	Список литературы	31

Список иллюстраций

4.1	Файл lab10-1.asm	9
4.2	Работа программы lab10-1.asm	10
4.3	Файл lab10-1.asm.....	11
4.4	Работа программы lab10-1.asm	12
4.5	Файл lab10-2.asm.....	13
4.6	Работа программы lab10-2.asm в отладчике	14
4.7	дисассимилированный код	15
4.8	дисассимилированный код в режиме интел	16
4.9	точка остановки.....	17
4.10	изменение регистров	18
4.11	изменение регистров	19
4.12	изменение значения переменной	20
4.13	вывод значения регистра	21
4.14	вывод значения регистра	22
4.15	вывод значения регистра	23
4.16	Файл lab10-4.asm.....	24
4.17	Работа программы lab10-4.asm	25
4.18	код с ошибкой	26
4.19	отладка	27
4.20	код исправлен	28
4.21	проверка работы	29

Список таблиц

1 Цель работы

Целью работы является приобретение навыков написания программ с использованием подпрограмм. Знакомство с методами отладки при помощи GDB и его основными возможностями.

2 Задание

1. Изучите примеры реализации подпрограмм
2. Изучите работу с отладчиком GDB
3. Выполните самостоятельное задание
4. Загрузите файлы на GitHub.

3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. В общем случае его можно разделить на четыре этапа:

- обнаружение ошибки;
- поиск её местонахождения;
- определение причины ошибки;
- исправление ошибки.

4 Выполнение лабораторной работы

1. Создайте каталог для выполнения лабораторной работы № 10, перейдите в него и создайте файл lab10-1.asm:
2. В качестве примера рассмотрим программу вычисления арифметического выражения $f(x) = 2x+7$ с помощью подпрограммы calcul. В данном примере x вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Внимательно изучите текст программы (Листинг 10.1). (рис. 4.1, 4.2)


```
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2x+7=',0
5 SECTION .bss
6 x: RESB 80
7 rez: RESB 80
8 SECTION .text
9 GLOBAL _start
10 _start:
11 ;-----
12 ; Основная программа
13 ;-----
14 mov eax, msg
15 call sprint
16 mov ecx, x
17 mov edx, 80
18 call sread
19 mov eax, x
20 call atoi
21 call _calcul ; Вызов подпрограммы _calcul
22 mov eax, result
23 call sprint
24 mov eax, [rez]
25 call iprintLF
26 call quit
27 ;-----
28 ; Подпрограмма вычисления
29 ; выражения "2x+7"
30 _calcul:
31 mov ebx, 2
32 mul ebx
33 add eax, 7
34 mov [rez], eax
35 ret ; выход из подпрограммы
```

Matlab ▾ Largeur des tabulations : 8 ▾ Lig 34, Col 13 ▾ INS

Рис. 4.1: Файл lab10-1.asm

Рис. 4.2: Работа программы lab10-1.asm

3. Измените текст программы, добавив подпрограмму subcalcul в подпрограмму calcul, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$ (рис. 4.3, 4.4)

```
1 %include 'in_out.asm'
2 SECTION .data
3 msg: DB 'Введите x: ',0
4 result: DB '2(3x-1)+7=',0
5
6 SECTION .bss
7 x: RESB 80
8 rez: RESB 80
9
10 SECTION .text
11 GLOBAL _start
12 _start:
13 mov eax, msg
14 call sprint
15 mov ecx, x
16 mov edx, 80
17 call sread
18 mov eax, x
19 call atoi
20 call _calcul ; Вызов подпрограммы _calcul
21 mov eax, result
22 call sprint
23 mov eax, [rez]
24 call iprintLF
25 call quit
26
27 _calcul:
28 call _subcalcul
29 mov ebx, 2
30 mul ebx
31 add eax, 7
32 mov [rez], eax
33 ret ; выход из подпрограммы
34
35 _subcalcul:
36 mov ebx, 3
37 mul ebx
38 sub eax, 1
39 ret
```

Matlab ▾ Largeur des tabulations : 8 ▾ Lig 39, Col 4 ▾ INS

Рис. 4.3: Файл lab10-1.asm

```
[gpgerra@fedora lab10]$ nasm -f elf lab10-1.asm
[gpgerra@fedora lab10]$ ld -m elf_i386 -o lab10-1 lab10-1.o
[gpgerra@fedora lab10]$ ./lab10-1
Введите x: 2
2(3x-1)+7=17
[gpgerra@fedora lab10]$
```

Рис. 4.4: Работа программы lab10-1.asm

4. Создайте файл lab10-2.asm с текстом программы из Листинга 10.2. (Программа печати сообщения Hello world!): (рис. 4.5)



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

Рис. 4.5: Файл lab10-2.asm

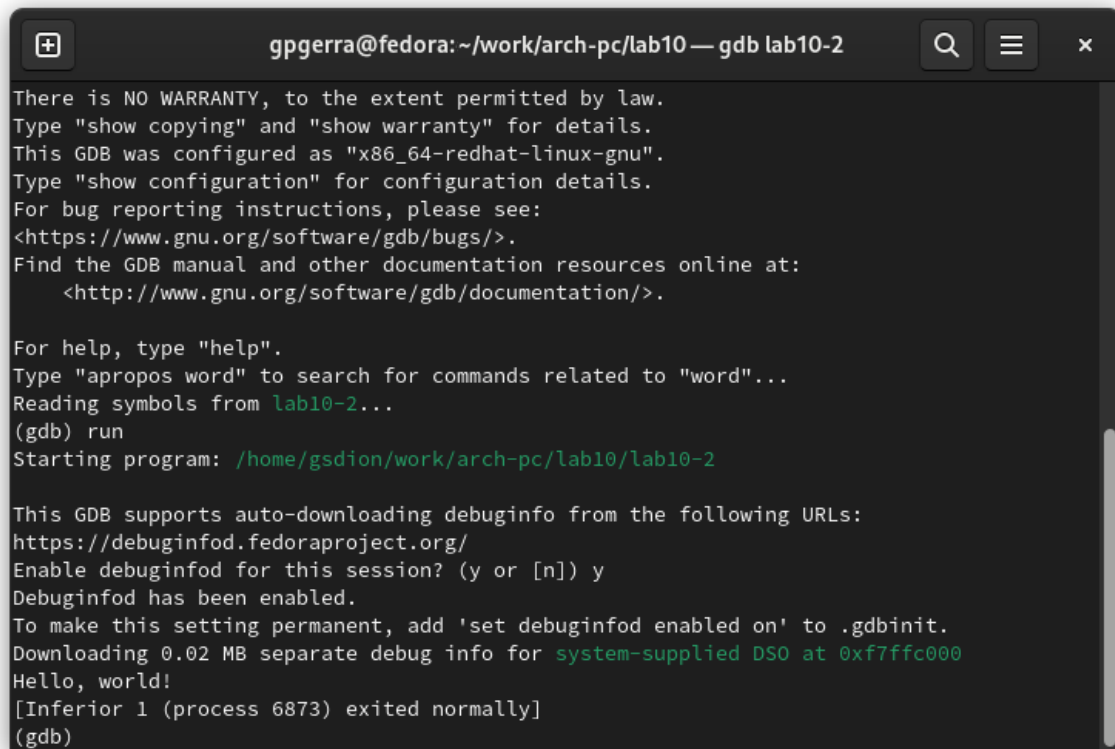
Получите исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом '-g'. Загрузите исполняемый файл в отладчик gdb: Проверьте работу программы, запустив ее в оболочке GDB с помощью команды `run` (сокращённо `r`):(рис. 4.6)

```
GNU gdb (GDB) Fedora Linux 12.1-6.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 lab10-2...
(gdb)
```

Рис. 4.6: Работа программы lab10-2.asm в отладчике

Для более подробного анализа программы установите брейкпоинт на метку start, с которой начинается выполнение любой ассемблерной программы, и запустите её. Посмотрите дисассимилированный код программы (рис. 4.7, 4.8)



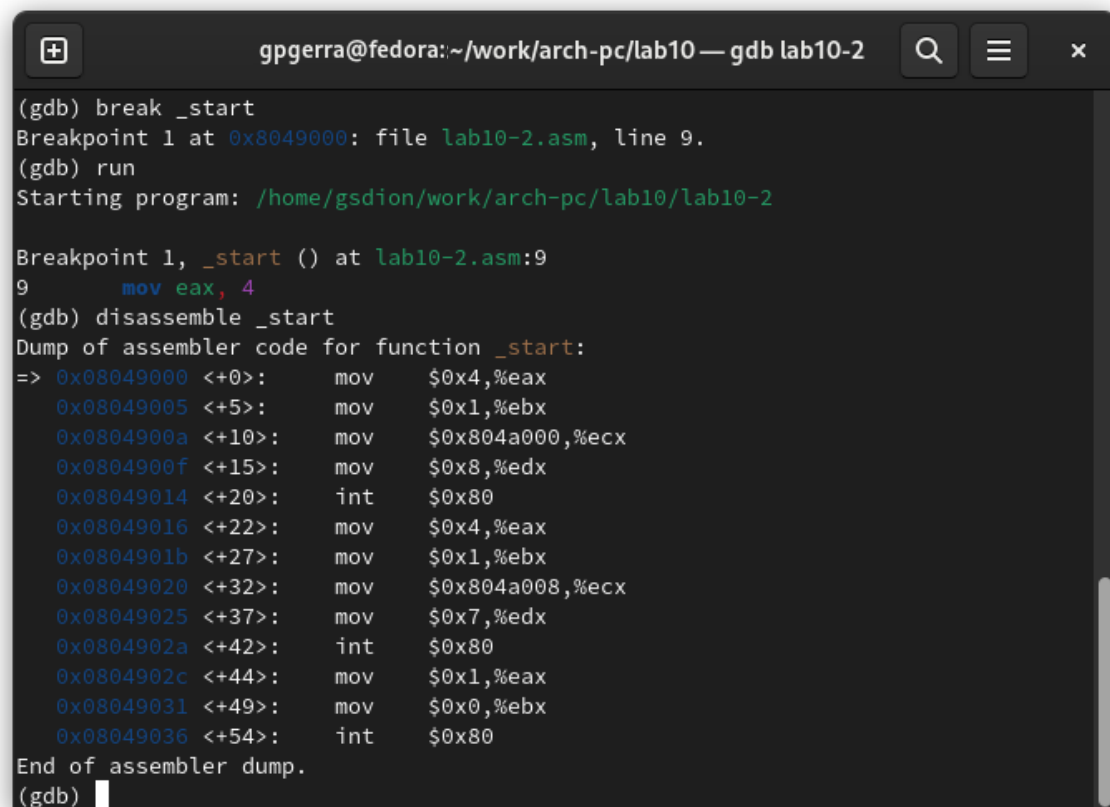
The image shows a terminal window with a dark background. The title bar at the top reads "gpgerra@fedora: ~/work/arch-pc/lab10 — gdb lab10-2". The terminal output shows the GDB startup sequence, including warranty information, configuration details, and the execution of the "run" command. The program being executed is located at "/home/gsdion/work/arch-pc/lab10/lab10-2". The output of the program is "Hello, world!". The terminal also shows the GDB interface prompts "(gdb)" and "[Inferior 1 (process 6873) exited normally]".

```
gpgerra@fedora: ~/work/arch-pc/lab10 — gdb lab10-2
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 lab10-2...
(gdb) run
Starting program: /home/gsdion/work/arch-pc/lab10/lab10-2

This GDB supports auto-downloading debuginfo from the following URLs:
https://debuginfod.fedoraproject.org/
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Downloading 0.02 MB separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 6873) exited normally]
(gdb)
```

Рис. 4.7: дисассимилированный код



```
(gdb) break _start
Breakpoint 1 at 0x08049000: file lab10-2.asm, line 9.
(gdb) run
Starting program: /home/gsdion/work/arch-pc/lab10/lab10-2

Breakpoint 1, _start () at lab10-2.asm:9
9      mov eax, 4
(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. 4.8: дисассимилированный код в режиме интел

На предыдущих шагах была установлена точка останова по имени метки (`_start`). Проверьте это с помощью команды `info breakpoints` (кратко `i b`) 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.(рис. 4.9)

The screenshot shows a GDB terminal window with the title bar "gpgerra@fedora:~/work/arch-pc/lab10 — gdb lab10-2". The window contains the following text:

```
[ Register Values Unavailable ]
```

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

```
native process 7162 In: _start L9 PC: 0x8049000
(gdb) layout regs
(gdb) 
```

Рис. 4.9: точка остановки

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Выполните 5 инструкций с помощью команды `stepi` (или `si`) и проследите за изменением значений регистров. (рис. 4.11 4.12)

```
gpgerra@fedora:~/work/arch-pc/lab10 — gdb lab10-2

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x0      0
esp      0xffffd180 0xffffd180

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

native process 7162 In: _start L10 PC: 0x8049005
(gdb) layout regs
(gdb) si
(gdb) 
```

Рис. 4.10: изменение регистров

```
gpgerra@fedora:~/work/arch-pc/lab10 — gdb lab10-2

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd180 0xffffd180

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

native process 7162 In: _start L14 PC: 0x8049016
(gdb) layout regs
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
```

Рис. 4.11: изменение регистров

Посмотрите значение переменной msg1 по имени Посмотрите значение переменной msg2 по адресу Изменить значение для регистра или ячейки памяти можно с помощью команды set, задав ей в качестве аргумента имя регистра или адрес. Измените первый символ переменной msg1 Замените любой символ во второй переменной msg2. (рис. 4.12)

The screenshot shows a GDB terminal window with the title bar "gpgerra@fedora: ~/work/arch-pc/lab10 — gdb lab10-2". The window is divided into three main sections. The top section, titled "Register group: general", displays the values of several registers:

eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x1	1
esp	0xffffd180	0xffffd180
ebp	0x0	0x0
esi	0x0	0

. The middle section shows a list of assembly instructions with their addresses and disassembled forms:

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

. The bottom section shows the current state of the process: "native process 7162 In: _start" with "L14" and "PC: 0x8049016". Below this, several GDB commands and their outputs are shown:

```
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "world!\n\034"
(gdb) set {char}0x804a008='L'
(gdb) x/1sb 0x804a008
0x804a008 <msg2>: "Lorld!\n\034"
(gdb) x/1sb &msg1
0x804a000 <msg1>: "Hello, "
```

Рис. 4.12: изменение значения переменной

Выведете в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра `edx`. С помощью команды `set` измените значение регистра `ebx`: (рис. 4.13)

```
gpgerra@fedora:~/work/arch-pc/lab10 — gdb lab10-2

Register group: general
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffd180 0xffffd180
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049016 0x8049016 <_start+22>

0x8049040 add BYTE PTR [eax],al
0x8049042 add BYTE PTR [eax],al
0x8049044 add BYTE PTR [eax],al
0x8049046 add BYTE PTR [eax],al
0x8049048 add BYTE PTR [eax],al
0x804904a add BYTE PTR [eax],al
0x804904c add BYTE PTR [eax],al
0x804904e add BYTE PTR [eax],al
0x8049050 add BYTE PTR [eax],al

native process 7162 In: _start L14 PC: 0x8049016
(gdb) p/t $edx
$4 = 1000
(gdb) p/x $edx
$5 = 0x8
(gdb) 
```

Рис. 4.13: вывод значения регистра

С помощью команды set измените значение регистра ebx:(рис. 4.14)

The screenshot shows a GDB terminal window with the title bar "gpgerra@fedora: ~/work/arch-pc/lab10 — gdb lab10-2". The window is divided into two main sections. The top section, titled "Register group: general", lists the values of several registers:

Register	Value 1	Value 2
eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x2	2
esp	0xffffd180	0xffffd180
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x8049016	0x8049016 <_start+22>

. The bottom section displays a list of assembly instructions starting from address 0x8049040:

Address	Instruction
0x8049040	add BYTE PTR [eax],al
0x8049042	add BYTE PTR [eax],al
0x8049044	add BYTE PTR [eax],al
0x8049046	add BYTE PTR [eax],al
0x8049048	add BYTE PTR [eax],al
0x804904a	add BYTE PTR [eax],al
0x804904c	add BYTE PTR [eax],al
0x804904e	add BYTE PTR [eax],al
0x8049050	add BYTE PTR [eax],al

. Below the assembly list, the status bar shows "native process 7162 In: _start L14 PC: 0x8049016". The command history at the bottom shows: (gdb) p/t \$edx, \$4 = 1000, (gdb) p/x \$edx, \$5 = 0x8, (gdb) set \$ebx='2', (gdb) p/s \$ebx, \$6 = 50, (gdb) set \$ebx=2, (gdb) p/s \$ebx, \$7 = 2, and (gdb).

Рис. 4.14: вывод значения регистра

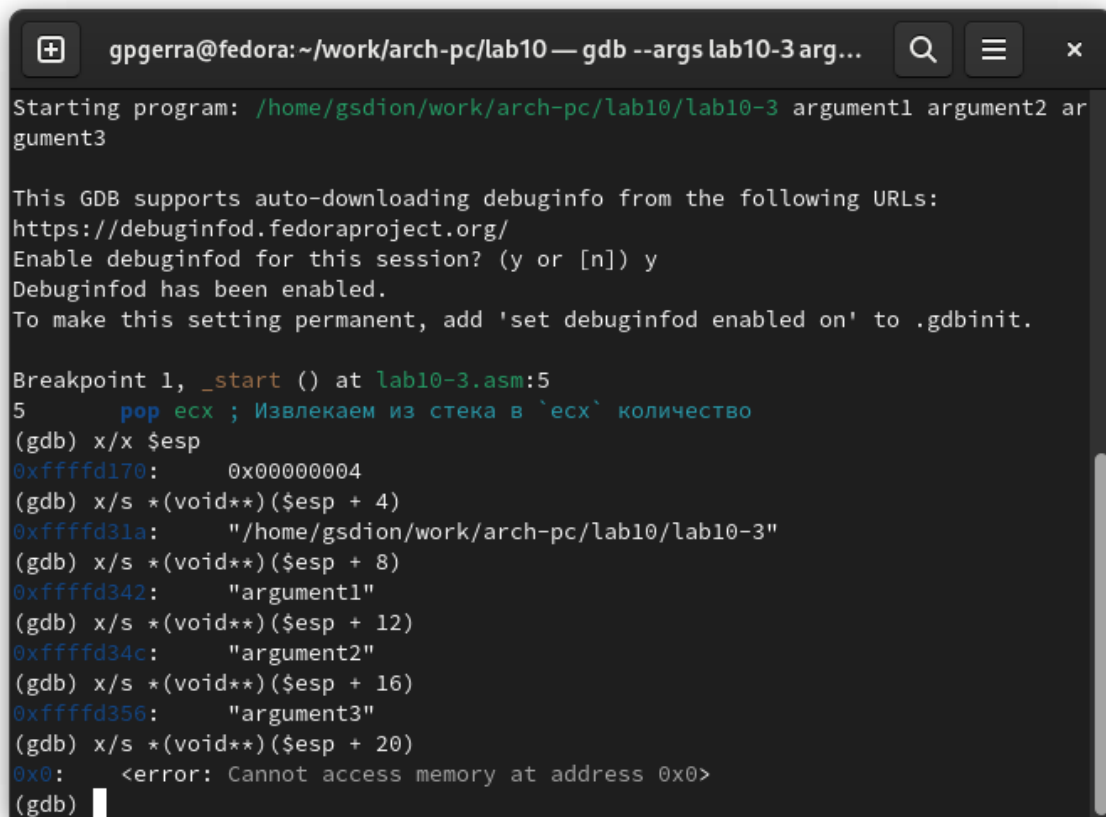
5. Скопируйте файл lab9-2.asm, созданный при выполнении лабораторной работы №9, с программой выводящей на экран аргументы командной строки. Создайте исполняемый файл. Для загрузки в gdb программы с аргументами необходимо использовать ключ `-args`. Загрузите исполняемый файл в отладчик, указав аргументы

Для начала установим точку останова перед первой инструкцией в программе

и запустим ее.

Адрес вершины стека храниться в регистре `esp` и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы): Как видно, число аргументов равно 5 – это имя программы `lab10-3` 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. 4.15)



```
gpgerra@fedora: ~/work/arch-pc/lab10 — gdb --args lab10-3 arg...
Starting program: /home/gsdion/work/arch-pc/lab10/lab10-3 argument1 argument2 argument3

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.

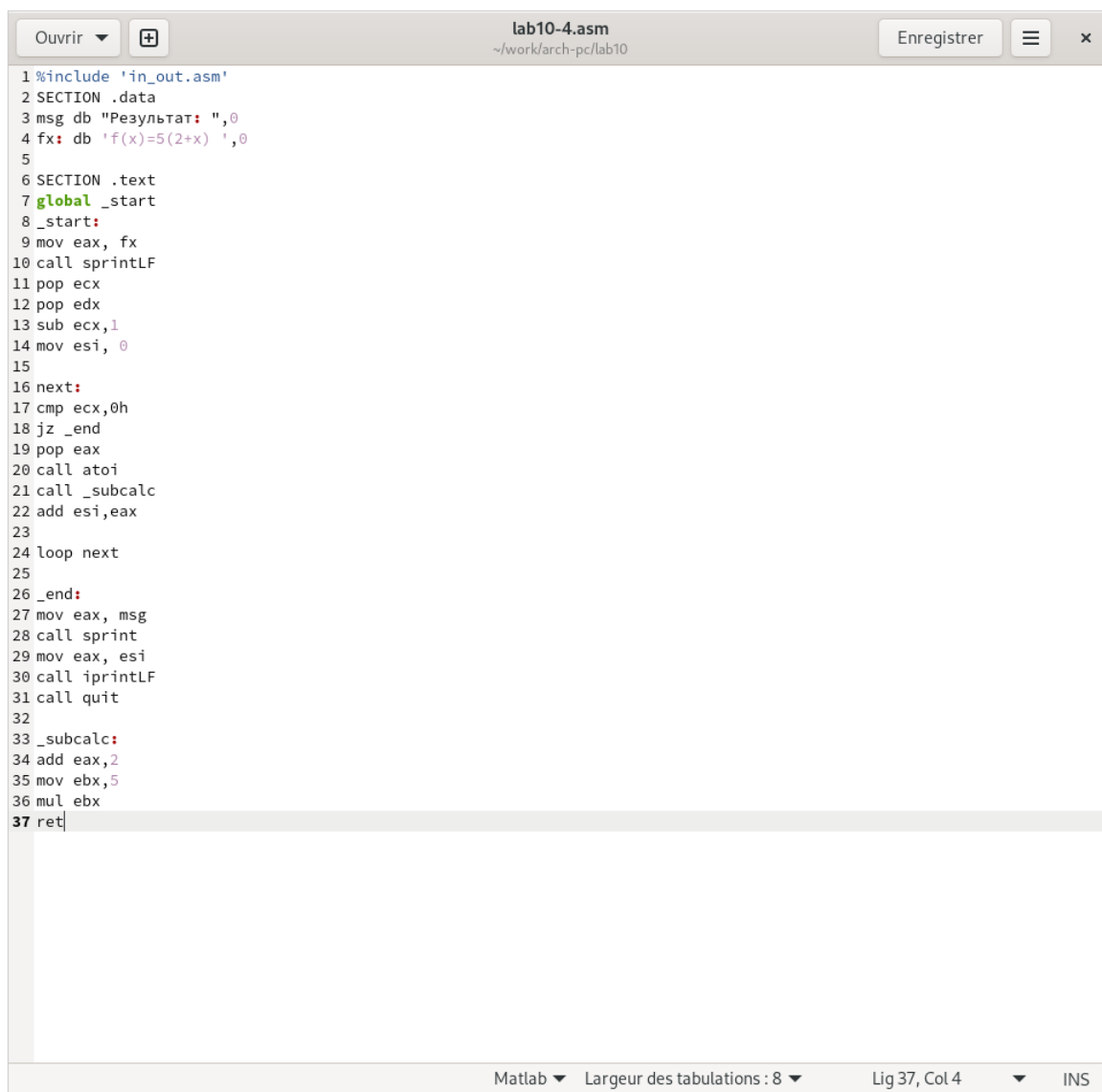
Breakpoint 1, _start () at lab10-3.asm:5
5      pop ecx ; Извлекаем из стека в `ecx` количество
(gdb) x/x $esp
0xffffd170:      0x00000004
(gdb) x/s *(void**)( $esp + 4)
0xffffd31a:      "/home/gsdion/work/arch-pc/lab10/lab10-3"
(gdb) x/s *(void**)( $esp + 8)
0xffffd342:      "argument1"
(gdb) x/s *(void**)( $esp + 12)
0xffffd34c:      "argument2"
(gdb) x/s *(void**)( $esp + 16)
0xffffd356:      "argument3"
(gdb) x/s *(void**)( $esp + 20)
0x0:      <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 4.15: вывод значения регистра

Объясните, почему шаг изменения адреса равен 4 (`[esp+4]`, `[esp+8]`, `[esp+12]`) - шаг равен размеру переменной - 4 байтам.

6. Преобразуйте программу из лабораторной работы №9 (Задание №1 для

самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму. (рис. 4.16 4.17)



```
1 %include 'in_out.asm'
2 SECTION .data
3 msg db "Результат: ",0
4 fx: db 'f(x)=5(2+x) ',0
5
6 SECTION .text
7 global _start
8 _start:
9 mov eax, fx
10 call sprintf
11 pop ecx
12 pop edx
13 sub ecx,1
14 mov esi, 0
15
16 next:
17 cmp ecx,0h
18 jz _end
19 pop eax
20 call atoi
21 call _subcalc
22 add esi,eax
23
24 loop next
25
26 _end:
27 mov eax, msg
28 call sprintf
29 mov eax, esi
30 call iprintLF
31 call quit
32
33 _subcalc:
34 add eax,2
35 mov ebx,5
36 mul ebx
37 ret
```

Matlab ▾ Largeur des tabulations : 8 ▾ Lig 37, Col 4 ▾ INS

Рис. 4.16: Файл lab10-4.asm

Рис. 4.17: Работа программы lab10-4.asm

7. В листинге приведена программа вычисления выражения $(3+2)^* 4+5$. При запуске данная программа дает неверный результат. Проверьте это. С помощью отладчика GDB, анализируя изменения значений регистров, определите ошибку и исправьте ее.(рис. 4.18 4.19 4.20 4.21)

```
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ---- Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add ebx,eax
11 mov ecx,4
12 mul ecx
13 add ebx,5
14 mov edi,eax
15 ; ----| Вывод результата на экран
16 mov eax,div
17 call sprint
18 mov eax,edi
19 call iprintLF
20 call quit
21
22
```

Matlab ▾ Largeur des tabulations : 8 ▾ Lig 15, Col 7 ▾ INS

Рис. 4.18: код с ошибкой

```
1 %include 'in_out.asm'
2 SECTION .data
3 div: DB 'Результат: ',0
4 SECTION .text
5 GLOBAL _start
6 _start:
7 ; ---- Вычисление выражения (3+2)*4+5
8 mov ebx,3
9 mov eax,2
10 add eax,ebx
11 mov ecx,4
12 mul ecx
13 add eax,5
14 mov edi,eax
15 ; ---- Вывод результата на экран
16 mov eax,div
17 call sprint
18 mov eax,edi
19 call iprintLF
20 call quit
21
22
```

Enregistrement du fichier « /home/gsdion/work/arch-pc/la... Matlab ▾ Largeur des tabulations : 8 ▾ Lig 13, Col 7 ▾ INS

Рис. 4.19: отладка

Отметим, что перепутан порядок аргументов у инструкции `add` и что по окончании работы в `edi` отправляется `ebx` вместо `eax`

```
eax      0x2      2
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd150  0xffffd150
ebp      0x0      0x0
esi      0x0      0

B+ 0x80490e8 <_start>    mov     ebx,0x3
   0x80490f9 <_start+17> mul     ecx,0x5
   0x80490fe <_start+22> mov     edi,ebx
   0x8049100 <_start+24> mov     eax,0x804a000

X 0x8049105 <_start+29> call    0x804900f <sprint>
   0x804910a <_start+34> mov     eax,edi
   0x804910c <_start+36> call    0x8049086 <iprintLF>
   0x8049111 <_start+41> call    0x80490db <quit>

native process 9662 In: _start L12 PC: 0x80490f9
to makeNo process In: , add 'set debuginfod enabled off' to .gdbinit. L?? PC: ??
breakpoint 1, _start () at lab10-5.asm:8
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) cont
continuing.
результат: 10
Inferior 1 (process 9662) exited normally]
```

Рис. 4.20: код исправлен

```

eax      0x19      25
ecx      0x4       4
edx      0x0       0
ebx      0x3       3
esp      0xffffd150 0xffffd150
ebp      0x0       0x0
esi      0x0       0

0x80490f2 <_start+10> add    eax,ebx
0x80490fe <_start+22> mov    edi,eax04a000
0x8049105 <_start+29> call   0x804900f <sprint>
0x804910a <_start+34> mov    eax,edi

0x804910c <_start+36> call   0x8049086 <iprintLF>
0x8049111 <_start+41> call   0x80490db <quit>

native process 9742 In: _start L14 PC: 0x80490fe
Breakpoint 1: No process In: 10-5.asm:8 L?? PC: ??
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) cont
Continuing.
Результат: 25
[Inferior 1 (process 9742) exited normally]
(gdb)

```

Рис. 4.21: проверка работы

5 Выводы

Освоили работу с подпрограммами и отладчиком.

Список литературы

1. Расширенный ассемблер: NASM
2. MASM, TASM, FASM, NASM под Windows и Linux