

# **Лабораторная работа №10**

**Понятие подпрограммы. Отладчик GDB.**

Киньябаева Аиша Иделевна

# Содержание

1	Цель работы	4
2	Задание	5
3	Выполнение лабораторной работы	6
4	Выводы	21

## Список иллюстраций

3.1	код . . . . .	6
3.2	lab10-1.asm_вывод . . . . .	7
3.3	код . . . . .	7
3.4	код . . . . .	8
3.5	lab10-1.asm_вывод . . . . .	8
3.6	GDB . . . . .	9
3.7	проверка через run . . . . .	9
3.8	break point . . . . .	10
3.9	Синтаксис Intel . . . . .	10
3.10	layout asm . . . . .	11
3.11	layout rex . . . . .	11
3.12	info breakpoints . . . . .	12
3.13	info registers . . . . .	12
3.14	si 5 . . . . .	13
3.15	x/1sb . . . . .	13
3.16	set . . . . .	14
3.17	значение edx . . . . .	14
3.18	значение ebx . . . . .	15
3.19	аргументы . . . . .	15
3.20	код . . . . .	16
3.21	2(x-1) вывод . . . . .	16
3.22	проверка программы . . . . .	16
3.23	проверка программы . . . . .	17
3.24	проверка программы . . . . .	17
3.25	проверка программы . . . . .	18
3.26	проверка программы . . . . .	18
3.27	проверка программы . . . . .	19
3.28	проверка программы . . . . .	19
3.29	код . . . . .	20
3.30	(3 + 2) * 4 + 5 вывод . . . . .	20

# 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

Научиться отладке программы с помощью отладчика GDB и изучение подпрограмм.

### 3 Выполнение лабораторной работы

Пишу программу с использованием с подпрограммы для вычисления выражения:  $2x + 7$  (рис. 3.1), (рис. 3.2)

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

Рис. 3.1: код

```

aikinjyabaeva@aikinjyabaeva-VirtualBox:~/w
● ork/arch-pc/lab10$ ./lab10-1
Введите x: 7
2x+7=21

```

Рис. 3.2: lab10-1.asm\_вывод

Меняем программу, чтобы она вычисляла:  $f(g(x)) = 2(3x - 1) + 7$  (рис. [fig:fig3]),  
(рис. 3.3), (рис. 3.5)

```

1  %include 'in_out.asm'
2
3  SECTION .data
4  msg: DB 'Введите x: ',0
5  result: DB '2(3x-1)+7=',0
6  SECTION .bss
7  x: RESB 80
8  rezs: RESB 80
9  SECTION .text
10 GLOBAL _start
11 _start:

```

Рис. 3.3: код

```

7 | x: RESB 80
8 | rezs: RESB 80
9 | SECTION .text
10 | GLOBAL _start
11 | _start:
12 | mov eax, msg
13 | call sprint
14 |
15 | mov ecx, x
16 | mov edx, 80
17 | call sread
18 |
19 | mov eax, x
20 | call atoi
21 |
22 | call _calcul
23 | mov eax, result
24 | call sprint
25 | mov eax, [rezs]
26 | call iprintLF
27 |
28 | call quit
29 |
30 | _calcul:
31 | call _subcalcul
32 | mov ebx, 2
33 | mul ebx
34 | add eax, 7
35 | mov [rezs], eax
36 |
37 | ret
38 |
39 | _subcalcul:
40 | mov ebx, 3
41 | mul ebx
42 | dec eax
43 | mov [rezs], eax
44 |
45 | ret

```

Рис. 3.4: код

```

aikinjyabaeva@aikinjyabaeva-VirtualBox:~/work/arc
h-pc/lab10$ ./lab10-1
Введите x: 3
2(3x-1)+7=23
aikinjyabaeva@aikinjyabaeva-VirtualBox:~/work/arc

```

Рис. 3.5: lab10-1.asm\_вывод



Создаем программу для вывода надписи: Hello world, компилируем листинг файл с отладочной информацией, проверяем с помощью дебаггера GDB.(рис. 3.6), (рис. 3.7)

```
aikinjyabaeva@aikinjyabaeva-VirtualBox:~/work/arch-pc/lab10$ ./lab10-2
Hello, world!
aikinjyabaeva@aikinjyabaeva-VirtualBox:~/work/arch-pc/lab10$ gdb lab10-2
GNU gdb (Ubuntu 12.0.90-0ubuntu1) 12.0.90
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 lab10-2...
(No debugging symbols found in lab10-2)
(gdb)
```

Рис. 3.6: GDB

```
(gdb) r
Starting program: /home/aikinjyabaeva/work/arch-pc/lab10/lab10-2
Hello, world!
[Inferior 1 (process 6655) exited normally]
(gdb)
```

Рис. 3.7: проверка через run

Устанавливаем break на метку \_start, смотрим работу программы и далее запускаем диссасилированный код. Переключаемся на Intel'овский синтаксис. Различия синтаксисов заключается в разном отображении аргументов и значений(в Intel без доп. символов), а так же изменен порядок их отображения(рис. 3.8), (рис. 3.9)

```

(gdb) break _start
Breakpoint 1 at 0x08049000
(gdb) r
Starting program: /home/aikinjyabaeva/work/arch-pc/lab10/lab10-2

Breakpoint 1, 0x08049000 in _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.

```

Рис. 3.8: break point

```

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

```

Рис. 3.9: Синтаксис Intel

Два режима псевдографики(рис. 3.10), (рис. 3.11)

```

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
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
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 7404 In: \_start L?? PC: 0x8049000

Рис. 3.10: layout asm

[ 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,0x804a008
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

```

native process 7404 In: \_start L?? PC: 0x8049000

Рис. 3.11: layout rex

В режиме дебаггинга проверяем точки останова, и видим, что можем установить еще одну, используя адрес инструкции (рис. 3.12)

```

(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y  0x08049000 <_start>
          breakpoint already hit 1 time
(gdb) break *0x8049031
Breakpoint 2 at 0x8049031
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y  0x08049000 <_start>
          breakpoint already hit 1 time
2        breakpoint      keep y  0x08049031 <_start+49>
(gdb)

```

Рис. 3.12: info breakpoints

Проверяем значения регистров(рис. 3.13)

```

(gdb) i r
eax      0x0             0
ecx      0x0             0
edx      0x0             0
ebx      0x0             0
esp      0xffffd200      0xffffd200
ebp      0x0             0x0
esi      0x0             0
edi      0x0             0
eip      0x8049000      0x8049000 <_start>
eflags   0x202           [ IF ]
cs       0x23            35
ss       0x2b            43
ds       0x2b            43
es       0x2b            43
fs       0x0             0
gs       0x0             0

```

Рис. 3.13: info registers

Провела 5 инструкций с помощью команды si, меняются значения регистров: eax, ebx, ecx, edx. (рис. 3.14)

Register group: general		
eax	0x8	8
ecx	0x804a000	134520832
edx	0x8	8
ebx	0x1	1
esp	0xffffd200	0xffffd200
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

Рис. 3.14: si 5

Видим, что с помощью различных команд можно смотреть содержимое переменных(в конце скриншота просмотрено значение переменной msg2) (рис. 3.15)

```
(gdb) x/1sb &msg1
0x804a000: "Hello, "
(gdb) x/1sb 0x804a008
0x804a008: "world!\n"
(gdb) █
```

Рис. 3.15: x/1sb

А с помощью команды set можно менять значение регистра(в конце скриншота изменен символ второй переменной msg2)(рис. 3.16)

```

(gdb) set {char}&msg1='h'
(gdb) set {char}0x804a001='h'
(gdb) x/1sb &msg1
0x804a000:      "hhlllo, "
(gdb) set {char}0x804a001='e'
(gdb) x/1sb &msg1
0x804a000:      "hello, "
(gdb) set {char}0x804a008='L'
(gdb) set {char}0x804a00b=' '
(gdb) x/1sb &msg2
0x804a008:      "Lor d!\n"
(gdb) set {char}&msg2='w'
(gdb) x/1sb &msg2
0x804a008:      "wor d!\n"
(gdb)

```

Рис. 3.16: set

Изучаем использование команды print /F, которое смотрит значение регистров(можно регулировать формат отображения). Вывела в различных форматах значение регистра edx(16ричных, 2чный и символный вид соответственно)(рис. 3.17)

```

(gdb) p/x $edx
$5 = 0x8
(gdb) p/t $edx
$6 = 1000
(gdb) p/s $edx
$7 = 8
(gdb)

```

Рис. 3.17: значение edx

С помощью команды set, меняем значение регистра ebx (рис. 3.18)

```

(gdb) set $ebx='2'
(gdb) p/s $ebx
$8 = 50
(gdb) set $ebx=2
(gdb) p/s $ebx
$9 = 2
(gdb)

```

Рис. 3.18: значение ebx

Указываем аргументы файла в отладчик и далее смотрим позиции аргумента в стеке. Заметим, что  $[esp + 24]$  не выводит значения, так как у нас нет бго аргумента. Шаг изменения адреса равен 4, потому что память имеет 16ричный вид, и  $2^4=16$ .(рис. 3.19)

```

Breakpoint 1, _start () at lab10-3.asm:7
7      pop ecx
(gdb) x/x $esp
0xffffd1d0: 0x00000005
(gdb) x/s *(void**)($esp + 4)
0xffffd382: "/home/aikinjyabaeva/work/arch-pc/lab10/lab10-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd3b1: "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xffffd3c3: "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xffffd3d4: "2"
(gdb) x/s *(void**)($esp + 20)
0xffffd3d6: "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 3.19: аргументы

## САМОСТОЯТЕЛЬНАЯ РАБОТА

1. Меняю программу из 9 лаб. работы, которая вычисляла  $2(x-1)$  с использованием подпрограмм (рис. 3.20), (рис. 3.21)

```

ASM mywork.asm
1  %include 'in_out.asm'
2
3  SECTION .data
4  | msg: DB 'Введите x: ',0
5  | result: DB '2(x-1)=',0
6  SECTION .bss
7  | x: RESB 80
8  | rezs: RESB 80
9  SECTION .text
10 GLOBAL _start
11 | _start:
12  mov eax, msg
13  call sprint
14
15  mov ecx, x
16  mov edx, 80
17  call sread
18
19  mov eax, x
20  call atoi
21
22  call _calcul
23  mov eax, result
24  call sprint
25  mov eax, [rezs]
26  call iprintLF
27
28  call quit
29
30 _calcul:
31  dec eax
32  mov ebx, 2
33  mul ebx
34  mov [rezs], eax
35  ret

```

Рис. 3.20: код

```

aikinjuryabaeva@aikinjuryabaeva-VirtualBox:~/work/arch-pc/lab10$ ./mywork
Введите x: 7
2(x-1)=12
aikinjuryabaeva@aikinjuryabaeva-VirtualBox:~/work/arch-pc/lab10$ S

```

Рис. 3.21:  $2(x-1)$  вывод

2.Находим ошибка в листинге, для этого используем GDB, смотрим значение регистров. Просматриваю каждый шаг программы с помощью si (рис. 3.22), (рис. 3.23), (рис. 3.24), (рис. 3.25), (рис. 3.26), (рис. 3.27), (рис. 3.28)

```

mywork-1.asm
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

```

native process 4272 In: \_start L8 PC: 0x80490e8

Рис. 3.22: проверка программы



Register group: general		
eax	0x0	0
ecx	0x0	0
edx	0x0	0
ebx	0x3	3
esp	0xffffd210	0xffffd210
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x80490ed	0x80490ed <_start+5>
eflags	0x202	[ IF ]

Рис. 3.23: проверка программы

Register group: general		
eax	0x2	2
ecx	0x0	0
edx	0x0	0
ebx	0x3	3
esp	0xffffd210	0xffffd210
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x80490f2	0x80490f2 <_start+10>
eflags	0x202	[ IF ]

  

mywork-1.asm		
B+	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,ebx
	15	; ---- Вывод результата на экран
	16	mov eax,div
	17	call sprint

  

native process 4272 In: _start		
--------------------------------	--	--

Рис. 3.24: проверка программы

```

Register group: general
eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x5      5
esp      0xffffd210 0xffffd210
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490f4 0x80490f4 <_start+12>
eflags   0x206    [ PF IF ]

mywork-1.asm
B+      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,ebx
       15  ; ---- Вывод результата на экран
       16  mov eax,div
       17  call sprint

native process 4272 In: _start

```

Рис. 3.25: проверка программы

```

Register group: general
eax      0x2      2
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd210 0xffffd210
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490f9 0x80490f9 <_start+17>
eflags   0x206    [ PF IF ]

mywork-1.asm
B+      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,ebx
       15  ; ---- Вывод результата на экран
       16  mov eax,div
       17  call sprint

native process 4272 In: _start

```

Рис. 3.26: проверка программы

```

Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0x5      5
esp      0xffffd210 0xffffd210
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490fb 0x80490fb <_start+19>
eflags   0x202    [ IF ]

mywork-1.asm
B+      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,ebx
       15  ; ---- Вывод результата на экран
       16  mov eax,div
       17  call sprint

native process 4272 In: _start

```

Рис. 3.27: проверка программы

```

Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffd210 0xffffd210
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490fe 0x80490fe <_start+22>
eflags   0x206    [ PF IF ]

mywork-1.asm
B+      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,ebx
       15  ; ---- Вывод результата на экран
       16  mov eax,div
       17  call sprint

native process 4272 In: _start

```

Рис. 3.28: проверка программы

Видим, что ошибка в том, что программа умножает на 4 двойку из eax. Исправ-

ляем программу в редакторе. (рис. 3.29), (рис. 3.30)

```
asm mywork-1.asm
1  %include 'in_out.asm'
2  SECTION .data
3  msg: DB 'Результат: ',0
4  SECTION .text
5  GLOBAL _start
6  _start:
7  mov ebx,3
8  mov eax,2
9  add ebx,eax
10 mov eax,ebx
11 mov ecx,4
12 mul ecx
13 mov ebx,eax
14 add ebx,5
15 mov edi,ebx
16
17 mov eax,msg
18 call sprintf
19
20 mov eax,edi
21 call iprintLF
22 call quit
```

Рис. 3.29: код

```
aikinjyabaeva@aikinjyabaeva-VirtualBox:~/work/arch-pc/lab10$ ./mywork-1
Результат: 25
aikinjyabaeva@aikinjyabaeva-VirtualBox:~/work/arch-pc/lab10$
```

Рис. 3.30:  $(3 + 2) * 4 + 5$  вывод

Загрузка всех файлов на Git.

Далее создается отчет по 10й лабораторной работе с помощью Markdown.

## 4 Выводы

В ходе данной лабораторной работы были изучены подпрограммы, освоены дебагинг с помощью GDB.