

Отчет по лабораторной работе № 9

Шубина София Антоновна

Содержание

1	Цель работы	6
2	Выполнение лабораторной работы	7
3	Выводы	30
	Список литературы	31

Список иллюстраций

2.1	Создание файла и каталога	7
2.2	Ввод текста программы	8
2.3	Создание исполняемого файла и проверка его работы	8
2.4	Создание файла	9
2.5	Трансляция программы при помощи ключа -g	10
2.6	Проверка работы программы	10
2.7	Запуск исполняемого файла	10
2.8	Просмотр кода программы с помощью команды disassemble	11
2.9	Ввод команды disassembly-flavor intel	11
2.10	Ввод режима псевдографики(layout asm)	12
2.11	Ввод режима псевдографики(layout regs)	12
2.12	Проверка установки точки останова по имени метки	13
2.13	Ввод адреса	13
2.14	Просмотр информации о всех установленных точках останова	14
2.15	Выполнение команды stepi(si)	14
2.16	Выполнение команды stepi(si)	15
2.17	Выполнение команды stepi(si)	15
2.18	Выполнение команды stepi(si)	16
2.19	Выполнение команды stepi(si)	16
2.20	Просмотр содержимого регистров	17
2.21	Просмотр значения переменной по имени	17
2.22	Просмотр значения переменной по адресу	17
2.23	Изменение первого символа в переменной msg1 и замена любого символа в переменной msg2	18
2.24	Вывод значения регистра edx	18
2.25	Завершение выполнения работы с помощью команды continue(c)	19
2.26	Копирование файла и создание исполняемого файла	20
2.27	Загрузим исполняемый файл в отладчик,указав аргументы	20
2.28	Установка точки и запуск программы	20
2.29	Содержимое регистра esp	21
2.30	Содержимое регистра esp	21
2.31	Неверно написанный текст программы	23
2.32	Запуск исполняемого файла	24
2.33	Загрузка файла f2.asm	24
2.34	Загрузка файла f2.asm	25
2.35	Просмотр значений регистров	26
2.36	Изменение значения регистра	27

2.37	Изменение регистра программы и завершение программы	28
2.38	Изменение текста программы	29
2.39	Запуск исполняемого файла	29

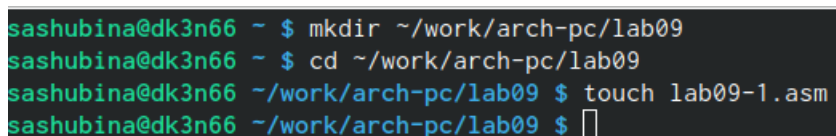
Список таблиц

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

Реализация подпрограмм в NASM 1. Создадим каталог для выполнения лабораторной работы No 9, перейдем в него и создадим файл lab09-1.asm: `mkdir ~/work/arch-pc/lab09 cd ~/work/arch-pc/lab09 touch lab09-1.asm` (рис. 2.1).



```
sashubina@dk3n66 ~ $ mkdir ~/work/arch-pc/lab09
sashubina@dk3n66 ~ $ cd ~/work/arch-pc/lab09
sashubina@dk3n66 ~/work/arch-pc/lab09 $ touch lab09-1.asm
sashubina@dk3n66 ~/work/arch-pc/lab09 $
```

Рис. 2.1: Создание файла и каталога

2. В качестве примера рассмотрим программу вычисления арифметического выражения $\text{X}(\text{X}) = 2\text{X} + 7$ с помощью подпрограммы `_calcul`. В данном примере `X` вводится с клавиатуры, а само выражение вычисляется в подпрограмме. Внимательно изучим текст программы. 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`) После следующей инструкции `call _calcul`, которая передает управление подпрограмме `_calcul`, будут выполнены инструкции подпрограммы 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`).

Введем в файл lab09-1.asm текст программы из листинга. Создадим исполняемый файл и проверим его работу. (рис. 2.2, 2.3).

```
lab09-1.asm      [-M--] 27 L:[ 1+34 35/ 35] *(706 / 706b) <EOF>
%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 aread
mov eax, x
call atoi
call _calcul ; Выход 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
```

Рис. 2.2: Ввод текста программы

```
sashubina@dk3n66 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
sashubina@dk3n66 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
sashubina@dk3n66 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 1
2x+7=9
sashubina@dk3n66 ~/work/arch-pc/lab09 $
```

Рис. 2.3: Создание исполняемого файла и проверка его работы

Изменим текст программы, добавив подпрограмму `_subcalcul` в подпрограмму `_calcul`, для вычисления выражения $\text{in}(\text{in}(\text{in}))$, где `in` вводится с клавиатуры, $\text{in}(\text{in}) =$

$2x + 7$, $x(x) = 3x - 1$. Т.е. x передается в подпрограмму `_calcul` из нее в подпрограмму `_subcalcul`, где вычисляется выражение $x(x)$, результат возвращается в `_calcul` и вычисляется выражение $x(x(x))$. Результат возвращается в основную программу для вывода результата на экран. тладка программ с помощью GDB (рис. ??, ??, ??).

```

lab09-1.asm [----] 12 L: [ 1+ 6 7/ 41] *(129 / 722b) 0010 0x00A
%include "io.inc.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:
;-----
; 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
_calcul:
;2x+7
call _subcalcul
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы

_subcalcul:
;3x-1
mov ebx, 3
mul ebx
dec eax ; eax=3x-1
ret ; выход из подпрограммы g(x)

sashubina@dk2n24 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
sashubina@dk2n24 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
sashubina@dk2n24 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 2
2(3x-1)+7=17

```

Создадим файл `lab09-2.asm` с текстом программы из Листинга. (Программа печати сообщения `Hello world!`) (рис. 2.4).

```

sashubina@dk2n24 ~/work/arch-pc/lab09 $ touch lab09-2.asm

```

Рис. 2.4: Создание файла

Получим исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавить отладочную информацию, для этого трансляцию программ необходимо проводить с ключом `'-g'`. `nasm -f elf -g -l lab09-2.lst lab09-2.asm ld -m elf_i386 -o lab09-2 lab09-2.o` Загрузим исполняемый файл в отладчик `gdb`: `user@dk4n31:~$ gdb lab09-2` (рис. 2.5).

```

sashubina@dk2n24 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
sashubina@dk2n24 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
sashubina@dk2n24 ~/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 lab9-1.asm
sashubina@dk2n24 ~/work/arch-pc/lab09 $ gdb lab09-2
GNU gdb (Gentoo 13.2 vanilla) 13.2
Copyright (C) 2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...

```

Рис. 2.5: Трансляция программы при помощи ключа -g

Проверим работу программы, запустив ее в оболочке GDB с помощью команды run (со- кращённо r): (gdb) run (рис. 2.6).

```

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from lab09-2...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/s/a/sashubina/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 6064) exited normally]
(gdb)

```

Рис. 2.6: Проверка работы программы

Starting program: ~/work/arch-pc/lab09/lab09-2 Hello, world! [Inferior 1 (process 10220) exited normally] (gdb) Для более подробного анализа программы установите брейкпоинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запустите её. (gdb) break _start (рис. 2.7).

```

[Inferior 1 (process 6064) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb)

```

Рис. 2.7: Запуск исполняемого файла

Breakpoint 1 at 0x8049000: file lab09-2.asm, line 12. (gdb) run Starting program: ~/work/arch-pc/lab09/lab09-2 Breakpoint 1, _start () at lab09-2.asm:12 12 mov eax, 4

Посмотрим дисассимилированный код программы с помощью команды disassemble начиная с метки _start (gdb) disassemble _start (рис. 2.8).

```

(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. 2.8: Просмотр кода программы с помощью команды disassemble

Переключимся на отображение команд с Intel'овским синтаксисом, введя команду `set disassembly-flavor intel` (gdb) `set disassembly-flavor intel` (gdb) `disassemble _start` (рис. 2.9).

```

(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. 2.9: Ввод команды disassembly-flavor intel

Перечислим различия отображения синтаксиса машинных команд в режимах АТТ и Intel. - В синтаксисе Intel вводится регистр после команды с помещенным значением, после вводится адрес помещаемого значения или число. В синтаксисе АТТ сначала выводится ссылка на заносимое в регистр значение с помощью символа "\$", а после нее регистр с символом.

Включите режим псевдографики для более удобного анализа программы: (gdb) layout asm (gdb) layout regs (рис. 2.10, 2.11).

```
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

exec No process In:
(gdb) █
```

Рис. 2.10: Ввод режима псевдографики(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

exec No process In:
(gdb) layout regs
(gdb) █
```

Рис. 2.11: Ввод режима псевдографики(layout regs)

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. бавление точек останова 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`). Проверим это с помощью команды `info breakpoints` (кратко `i b`): `(gdb) info breakpoints` (рис. 2.12).

```
(gdb) info breakpoints
Num   Type             Disp Enb Address      What
1      breakpoint      keep y   0x08049000 lab09-2.asm:9
(gdb)
```

Рис. 2.12: Проверка установки точки останова по имени метки

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. `(gdb) break *` Посмотрите информацию о всех установленных точках останова: `(gdb) i b` (рис. 2.13, 2.14).

```
(gdb) b *0x08049000
Note: breakpoint 1 also set at pc 0x08049000.
Breakpoint 2 at 0x08049000: file lab09-2.asm, line 9.
(gdb)
```

Рис. 2.13: Ввод адреса

```
(gdb) info breakpoints
Num   Type             Disp Enb Address      What
1     breakpoint      keep y   0x08049000 lab09-2.asm:9
(gdb) b *0x8049000
Note: breakpoint 1 also set at pc 0x8049000.
Breakpoint 2 at 0x8049000: file lab09-2.asm, line 9.
(gdb)
```

Рис. 2.14: Просмотр информации о всех установленных точках останова

Работа с данными программы в GDB Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных.

Выполним 5 инструкций с помощью команды `stepi` (или `si`) и проследим за изменением значений регистров. Значения каких регистров изменяются? Изменились значения регистров: `eax`, `ecx`, `edx`, `ebx`, значения остальных регистров не изменились. (рис. 2.15, 2.16, ??, 2.18, 2.19).

```

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

```

```

native process 2686 In: _start
0x08049025 <+37>: mov    edx,0x7
0x0804902a <+42>: int    0x80
0x0804902c <+44>: mov    eax,0x1
0x08049031 <+49>: mov    ebx,0x0
0x08049036 <+54>: int    0x80
--Type <RET> for more, q to quit, c to continue without paging--End of assembler dump.
(gdb) i b
Num   Type             Disp Enb Address      What
1     breakpoint      keep y   0x08049000 lab09-2.asm:9
      breakpoint already hit 1 time
(gdb) b *0x8049000
Note: breakpoint 1 also set at pc 0x8049000.
Breakpoint 2 at 0x8049000: file lab09-2.asm, line 9.
(gdb) si
(gdb)
```

Рис. 2.15: Выполнение команды `stepi`(`si`)

```

lab09-2.asm
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

native process 2686 In: _start
0x0804902a <+42>: int 0x80
0x0804902c <+44>: mov eax,0x1
0x08049031 <+49>: mov ebx,0x0
0x08049036 <+54>: int 0x80
--Type <RET> for more, q to quit, c to continue without paging--End of assembler dump.
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb) b *0x08049000
Note: breakpoint 1 also set at pc 0x08049000.
Breakpoint 2 at 0x08049000: file lab09-2.asm, line 9.
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.16: Выполнение команды stepi(si)

```

lab09-2.asm
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

native process 2686 In: _start
0x0804902c <+44>: mov eax,0x1
0x08049031 <+49>: mov ebx,0x0
0x08049036 <+54>: int 0x80
--Type <RET> for more, q to quit, c to continue without paging--End of assembler dump.
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb) b *0x08049000
Note: breakpoint 1 also set at pc 0x08049000.
Breakpoint 2 at 0x08049000: file lab09-2.asm, line 9.
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.17: Выполнение команды stepi(si)

```

lab09-2.asm
7 global _start
8 _start:
B+ 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

native process 2686 In: _start
0x08049031 <+49>: mov ebx,0x0
0x08049036 <+54>: int 0x80
--Type <RET> for more, q to quit, c to continue without paging--End of assembler dump.
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb) b *0x08049000
Note: breakpoint 1 also set at pc 0x08049000.
Breakpoint 2 at 0x08049000: file lab09-2.asm, line 9.
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.18: Выполнение команды stepi(si)

```

lab09-2.asm
7 global _start
8 _start:
B+ 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

native process 2686 In: _start
0x08049036 <+54>: int 0x80
--Type <RET> for more, q to quit, c to continue without paging--End of assembler dump.
(gdb) i b
Num Type Disp Enb Address What
1 breakpoint keep y 0x08049000 lab09-2.asm:9
breakpoint already hit 1 time
(gdb) b *0x08049000
Note: breakpoint 1 also set at pc 0x08049000.
Breakpoint 2 at 0x08049000: file lab09-2.asm, line 9.
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb) si
(gdb)

```

Рис. 2.19: Выполнение команды stepi(si)

Посмотреть содержимое регистров также можно с помощью команды info registers (или i r). (gdb) info registers (рис. 2.20).


```
native process 2686 In: _start
eax      0x8      8
ecx      0x804a000 134520832
edx      0x8      8
ebx      0x1      1
esp      0xffffc320 0xffffc320
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
--Type <RET> for more, q to quit, c to continue without paging--
```

Рис. 2.20: Просмотр содержимого регистров

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.

Посмотрим значение переменной `msg1` по имени (gdb) `x/1sb &msg1 0x804a000`: “Hello,” (рис. 2.21).

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

Рис. 2.21: Просмотр значения переменной 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. 2.22).

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

Рис. 2.22: Просмотр значения переменной по адресу

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).

Изменим первый символ переменной `msg1`: `(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).

```
(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 первого символа в переменной `msg1` и замена любого символа в переменной `msg2`

Чтобы посмотреть значения регистров используется команда `print /F` (перед именем регистра обязательно ставится префикс `$`): `p/F $` (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`

Выведем в различных форматах (в шестнадцатеричном формате, в двоичном

формате и в символьном виде) значение регистра edx. С помощью команды set изменим значение регистра ebx: (gdb) set \$ebx='2' (gdb) p/s \$ebx \$3 = 50 (gdb) set \$ebx=2 (gdb) p/s \$ebx \$4 = 2 (gdb) (рис. ??).

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

Объясните разницу

вывода команд p/s \$ebx. Мы занесли в регистр ebx символ “2”, после значения регистра видим код символа “2”. Если занести в регистр изначально число “2”, а не символ, то команда вывела значение “2”.

Завершим выполнение программы с помощью команды continue (сокращенно c) или stepi (сокращенно si) и выйдем из GDB с помощью команды quit (сокращенно q). (рис. 2.25).

```
(gdb) c
Continuing.
World!
[Inferior 1 (process 2686) exited normally]
(gdb) 
```

Рис. 2.25: Завершение выполнения работы с помощью команды continue(c)

Обработка аргументов командной строки в GDB Скопируем файл lab8-2.asm, созданный при выполнении лабораторной работы No8, с программой вывода-

щей на экран аргументы командной строки 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. `nasm -f elf -g -l lab09-3.lst lab09-3.asm` `ld -m elf_i386 -o lab09-3 lab09-3.o` (рис. 2.26).

```
(gdb) layout regs
sashubina@dk8n67: ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
sashubina@dk8n67: ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
sashubina@dk8n67: ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
sashubina@dk8n67: ~/work/arch-pc/lab09 $
```

Рис. 2.26: Копирование файла и создание исполняемого файла

Для загрузки в gdb программы с аргументами необходимо использовать ключ `-args`. Загрузим исполняемый файл в отладчик, указав аргументы: `gdb -args lab09-3 аргумент1 аргумент2 'аргумент3'` (рис. 2.27).

```
sashubina@dk8n67: ~/work/arch-pc/lab09 $ gdb --args lab09-3 аргумент1 аргумент2 'аргумент3'
GNU gdb (Gentoo 12.1 vanilla) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

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

Рис. 2.27: Загрузим исполняемый файл в отладчик,указав аргументы

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. `(gdb) b _start` `(gdb) run` (рис. 2.28).

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 8.
(gdb) run
Starting program: /afs/dk.sc1.pfu.edu.ru/home/s/a/sashubina/work/arch-pc/lab09/lab09-3 аргумент1 аргумент2 аргумент\ 3

Breakpoint 1, _start () at lab09-3.asm:8
8      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb)
```

Рис. 2.28: Установка точки и запуск программы

Адрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы): (gdb) x/x \$esp 0xffffd200: 0x05 (рис. 2.29).

```
(gdb) x/x $esp
0xfffffc2d0:      0x00000005
(gdb) █
```

Рис. 2.29: Содержимое регистра esp

Как видно, число аргументов равно 5 – это имя программы lab09-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’. Посмотрив 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. 2.30).

```
0xfffffc2d0:      0x00000005
(gdb) x/s *(void**)($esp + 4)
0xfffffc56b:      "/afs/.dk.sci.pfu.edu.ru/home/s/a/sashubina/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)($esp + 8)
0xfffffc5b1:      "аргумент1"
(gdb) x/s *(void**)($esp + 12)
0xfffffc5c3:      "аргумент"
(gdb) x/s *(void**)($esp + 16)
0xfffffc5d4:      "2"
(gdb) x/s *(void**)($esp + 20)
0xfffffc5d6:      "аргумент 3"
(gdb) x/s *(void**)($esp + 24)
0x0:      <error: Cannot access memory at address 0x0>
(gdb) █
```

Рис. 2.30: Содержимое регистра esp

Объясните, почему шаг изменения адреса равен 4 ([esp+4], [esp+8], [esp+12] и т.д.). Шаг равен 4, потому что при прибавлении значений аргументов в стеке значение регистра esp увеличивается на 4. #Задание для самостоятельной работы 1. Преобразуем программу из лабораторной работы №8 (Задание No1 для самостоятельной работы), реализовав вычисление значения функции $\boxtimes(\boxtimes)$ как подпрограмму. (рис. ??).

```

#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

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:

```

```

mov ebx,4
mul ebx
mov ebx,3
sub eax,ebx
ret

```

```

sashubina@dk3n37 ~/work/arch-pc/lab09 $ gedit func.asm

sashubina@dk3n37 ~/work/arch-pc/lab09 $ nasm -f elf func.asm
sashubina@dk3n37 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o func func.o
sashubina@dk3n37 ~/work/arch-pc/lab09 $ ./func
Результат: 0
sashubina@dk3n37 ~/work/arch-pc/lab09 $ ./func 1 2 3 4
Результат: 28
sashubina@dk3n37 ~/work/arch-pc/lab09 $

```

{#fig:044

width=70%

2. В листинге 9.3 приведена программа вычисления выражения $(3 + 2) \times 4 + 5$. При запуске данная программа дает неверный результат. Проверим это. С помощью отладчика GDB, анализируя изменения значений регистров, определим ошибку и исправим ее. Создадим файл f2.asm и впишем в него текст программы вычисления выражения. (рис. 2.31).

```

f2.asm [----] 9 L: [ 1+19 20/ 20] *(348 / 348b) <EOF>
#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. 2.31: Неверно написанный текст программы

Создадим исполняемый файл и запустим его (рис. 2.32).

```
sashubina@dk4n70 ~/work/arch-pc/lab09 $ nasm -f elf f2.asm
sashubina@dk4n70 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o f2 f2.o
ld: невозможно найти f2.o: Нет такого файла или каталога
sashubina@dk4n70 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o f2 f2.o
sashubina@dk4n70 ~/work/arch-pc/lab09 $ ./f2
Результат: 10
sashubina@dk4n70 ~/work/arch-pc/lab09 $
```

Рис. 2.32: Запуск исполняемого файла

При запуске программа дает неверный результат. С помощью отладчика GDB проанализируем изменения значений регистров, исправим ошибку. Загрузим исполняемый файл в отладчик. (рис. 2.33).

```
sashubina@dk4n70 ~/work/arch-pc/lab09 $ gdb f2
GNU gdb (Gentoo 12.1 vanilla) 12.1
Copyright (C) 2022 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://bugs.gentoo.org/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from f2...
(No debugging symbols found in f2)
(gdb)
```

Рис. 2.33: Загрузка файла f2.asm

Просмотр дисассемблированного кода программы (рис. 2.34).


```

Breakpoint 1, _start () at lab09-2.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

```

Рис. 2.34: Загрузка файла f2.asm

Включим режим псевдографики, с помощью команды break установим точку основа на инструкции add,ebx,eax. С помощью команды si перейдем к следующей инструкции и проследим за изменением значений регистров. (рис. 2.35).

```
eax      0x5      5      ecx      0x0
ebx      0x5      5      esp      0xff
esi      0x0      0      edi      0x0
eflags   0x206    [ PF IF ]  cs      0x23
ds       0x2b     43     es      0x2b
gs       0x0      0

B+ 0x80490e8 <_start> mov $0x3,%ebx
0x80490ed <_start+5> mov $0x2,%eax
0x80490f2 <_start+10> add %eax,%ebx
> 0x80490f4 <_start+12> mov $0x4,%ecx
0x80490f9 <_start+17> mul %ecx
0x80490fb <_start+19> add $0x5,%ebx
0x80490fe <_start+22> mov %ebx,%edi
0x8049100 <_start+24> mov $0x804a000,%eax
0x8049105 <_start+29> call 0x804900f <sprint>
0x804910a <_start+34> mov %edi,%eax
0x804910c <_start+36> call 0x8049086 <iprintf>
0x8049111 <_start+41> call 0x80490db <quit>
0x8049116 <_start+46> add %al,(%eax)

native process 4952 In: _start
The program is not being run.
(gdb) r
Starting program: /afs/.dk.sci.pfu.edu.ru/home/s/a/sashubina/work/arch-pc/lab09/f2

Breakpoint 1, 0x080490e8 in _start ()
(gdb) si
0x080490ed in _start ()
(gdb) si
0x080490f2 in _start ()
(gdb) si
0x080490f4 in _start ()
(gdb) set $eax=5
$1 = 5
(gdb) p/s $eax
$1 = 5
(gdb) |
```

Рис. 2.35: Просмотр значений регистров

Чтобы исправить программу, занесем в `eax` с помощью команды `set` значение 5. (рис. 2.36).

```

eax      0x5      5      ecx      0x4
ebx      0x5      5      esp      0xff
esi      0x0      0      edi      0x0
eflags   0x206    [ PF IF ]  cs      0x23
ds       0x2b     43      es      0x2b
gs       0x0      0

B+ 0x80490e8 <_start>    mov     $0x3,%ebx
0x80490ed <_start+5>    mov     $0x2,%eax
0x80490f2 <_start+10>   add     %eax,%ebx
0x80490f4 <_start+12>   mov     $0x4,%ecx
> 0x80490f9 <_start+17>  mul     %ecx
0x80490fb <_start+19>   add     $0x5,%ebx
0x80490fe <_start+22>   mov     %ebx,%edi
0x8049100 <_start+24>   mov     $0x804a000,%eax
0x8049105 <_start+29>   call    0x804900f <sprint>
0x804910a <_start+34>   mov     %edi,%eax
0x804910c <_start+36>   call    0x8049086 <iprintf>
0x8049111 <_start+41>   call    0x80490db <quit>
0x8049116              add     %al,(%eax)

native process 4952 In: _start
Starting program: /afs/.dk.sc1.pfu.edu.ru/home/s/a/sashubina/work/arch-pc/lab09/f2

Breakpoint 1, 0x080490e8 in _start ()
(gdb) si
0x080490ed in _start ()
(gdb) si
0x080490f2 in _start ()
(gdb) si
0x080490f4 in _start ()
(gdb) set $eax=5
(gdb) p/s $eax
$1 = 5
(gdb) si
0x080490f9 in _start ()
(gdb)

```

Рис. 2.36: Изменение значения регистра

За результат программы отвечает регистр `ebx`. В него посместим значение $20+5$ и запустим программу.
(рис. 2.37).

```
ebx      0x19      25      esp      0xff
esi      0x0       0       edi      0x0
eflags   0x206     [ PF IF ]  cs      0x23
ds       0x2b      43     es       0x2b
gs       0x0       0

B+ 0x080490e8 <_start> mov $0x3,%ebx
5>

>

, %ebx

0,%eax [ No Assem
nt>

<iprintLF>

native process 4952 In: _start
(gdb) sNo process In:
$1 = 5
(gdb) si
0x080490f9 in _start ()
(gdb) si
0x080490fb in _start ()
(gdb) si
0x080490fe in _start ()
(gdb) set $ebx=25
(gdb) p/s $ebx
$2 = 25
(gdb) c
Continuing.
Результат: 25
[Inferior 1 (process 4952) exited normally]
(gdb)
```

Рис. 2.37: Изменение регистра программы и завершение программы

Изменим текст программы. (рис. 2.38).

```

f2.asm [----] 11 L: [ 1+17 18/ 20] *(225 / 249b) 0010 0x00A
#include "io.h"
SECTION .data
div: DB "Результат: ",0
SECTION .text
GLOBAL _start
_start:
mov ebx,3
mov eax,2
add eax,ebx
mov ecx,4
mul ecx
mov ebx,eax
add ebx,5
mov edi,ebx

mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit

```

Рис. 2.38: Изменение текста программы

Запустим исполняемый файл, программа выдает верный результат. (рис. 2.39).

```

sashubina@dk4n70 ~/work/arch-pc/lab09 $ nasm -f elf f2.asm
sashubina@dk4n70 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o f2 f2.o
sashubina@dk4n70 ~/work/arch-pc/lab09 $ ./f2
Результат: 25
sashubina@dk4n70 ~/work/arch-pc/lab09 $

```

Рис. 2.39: Запуск исполняемого файла

3 Выводы

Я приобрела навыки написания программ с использованием подпрограмм. Ознакомилась 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).