

Лабораторная работа №9

Архитектура компьютера

Тойчубекова Асель Нурлановна

Содержание

1	Цель работы	5
2	Теоретическое введение	7
2.1	Понятие об отладке	7
2.2	Методы отладки	7
2.3	Основные возможности отладчика GDB	8
2.4	Запуск отладчика GDB; выполнение программы; выход	8
2.5	Дизассемблирование программы	9
2.6	Точки останова	10
2.7	Пошаговая отладка	10
2.8	Работа с данными программы в GDB	10
2.9	Понятие подпрограммы	11
2.10	Инструкция call и инструкция ret	11
3	Выполнение лабораторной работы	12
3.1	Задание для самостоятельной работы	26
4	Выводы	34
	Список литературы	35

Список иллюстраций

3.1	РИС.1 Создание каталога	12
3.2	РИС.2 Редактирование файла	13
3.3	РИС.3 Запуск исполняемого файла	13
3.4	РИС.4 Редактирование файла	14
3.5	РИС.5 Запуск исполняемого файла	14
3.6	РИС.6 Редактирование файла	15
3.7	РИС.7 Запуск исполняемого файла с gdb	16
3.8	РИС.8 Проверка работы файла с помощью команды run	16
3.9	РИС.9 Установка breakpoint и запуск программы	16
3.10	РИС.10 Переключение дисассимилированного кода на intel'овский синтаксис	17
3.11	РИС.11 Включение режима псевдографики	18
3.12	РИС.12 Информация о точке останова	18
3.13	РИС.13 Установка точки останова и просмотр информации о всех точках останова	19
3.14	РИС.14 Выполнение 5 инструкций с помощью команды stepi	19
3.15	РИС.15 Выполнение 5 инструкций с помощью команды stepi	20
3.16	РИС.16 Просмотр значений переменных msg1,msg2	20
3.17	РИС.17 Изменение символа с помощью set	21
3.18	РИС.18 Изменение символа msg2	21
3.19	РИС.19 Вывод значения регистра в разных формах	22
3.20	РИС.20 Изменение значений регистра	23
3.21	РИС.21 Завершение работы GDB	24
3.22	РИС.22 Копирование файла	25
3.23	РИС.23 Создание исполняемого файла	25
3.24	РИС.24 Загрузка файла с аргументами в отладчик	25
3.25	РИС.25 Установка точки останова и запуск программы	25
3.26	РИС.26 Просмотр значений в стеке	26
3.27	РИС.27 Редактирование файла	28
3.28	РИС.28 Запуск программы	28
3.29	РИС.29 Редактирование файла	29
3.30	РИС.30 Запуск программы	29
3.31	РИС.31 Постановка breakpoint	30
3.32	РИС.32 Наблюдения за значениями регистров	31
3.33	РИС.33 Редактирование файла	32
3.34	РИС.34 Запуск программы	33

Список таблиц

1 Цель работы

Целью лабораторной работы №9 является приобретение навыков написания программ с использованием подпрограмм, также знакомство с методами отладки при помощи GDB и его основными возможностями. # Задание 1. Изучить теоретическое введение:

- Понятие отладки
 - Методы отладки
 - Основные возможности отладчика GDB
 - Запуск отладчика GDB;
 - Дизассемблирование программы;
 - Точки останова;
 - Пошаговая отладка;
 - Работа с данными программы в GDB;
 - Понятие подпрограммы;
 - Инструкция call и инструкция ret;
2. Написать программу с использованием вызова подпрограммы
 3. Выполнить отладку программы с помощью GDB
 4. Выполнить обработку аргументов командной строки в GDB
 5. Задание для самостоятельной работы:
 - Преобразуйте программу из лабораторной работы No8 (Задание No1 для самостоятельной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму.
 - В листинге 9.3 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.

2 Теоретическое введение

2.1 Понятие об отладке

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.

Можно выделить следующие типы ошибок:

- синтаксические ошибки — обнаруживаются во время трансляции исходного кода и вызваны нарушением ожидаемой формы или структуры языка;
- семантические ошибки — являются логическими и приводят к тому, что программа запускается, отработывает, но не даёт желаемого результата;
- ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают прерывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль).

2.2 Методы отладки

Наиболее часто применяют следующие методы отладки: • создание точек контроля значений на входе и выходе участка программы (например, вывод промежуточных значений на экран — так называемые диагностические сообщения);

- использование специальных программ-отладчиков.

2.3 Основные возможности отладчика GDB

GDB (GNU Debugger — отладчик проекта GNU) [1] работает на многих UNIX-подобных системах и умеет производить отладку многих языков программирования. GDB предлагает обширные средства для слежения и контроля за выполнением компьютерных программ. Отладчик не содержит собственного графического пользовательского интерфейса и использует стандартный текстовый интерфейс консоли. Однако для GDB существует несколько сторонних графических надстроек, а кроме того, некоторые интегрированные среды разработки используют его в качестве базовой подсистемы отладки. GDB может выполнять следующие действия:

- начать выполнение программы, задав всё, что может повлиять на её поведение;
- остановить программу при указанных условиях;
- исследовать, что случилось, когда программа остановилась;
- изменить программу так, чтобы можно было поэкспериментировать с устранением эффектов одной ошибки и продолжить выявление других.

2.4 Запуск отладчика GDB; выполнение программы; выход

Синтаксис команды для запуска отладчика имеет следующий вид:

`gdb` опции имя файла|ID процесса

После запуска `gdb` выводит текстовое сообщение — так называемое «nice GDB logo». В следующей строке появляется приглашение (`gdb`) для ввода команд.

Далее приведён список некоторых команд GDB.

Команда `run` (сокращённо `r`) — запускает отлаживаемую программу в оболочке GDB.

Если точки останова были заданы, то отладчик останавливается на соответ-

ствующей команде и выдаёт номер точки останова, адрес и дополнительную информацию — текущую строку, имя процедуры, и др.

Команда `kill` (сокращённо `k`) прекращает отладку программы, после чего следует вопрос о прекращении процесса отладки:

Kill the program being debugged? (y or n) `y`

Если в ответ введено `y` (то есть «да»), отладка программы прекращается. Командой `run` её можно начать заново, при этом все точки останова (breakpoints), точки просмотра (watchpoints) и точки отлова (catchpoints) сохраняются. Для выхода из отладчика используется команда `quit` (или сокращённо `q`)

2.5 Дизассемблирование программы

Если есть файл с исходным текстом программы, а в исполняемый файл включена информация о номерах строк исходного кода, то программу можно отлаживать, работая в отладчике непосредственно с её исходным текстом. Чтобы программу можно было отлаживать на уровне строк исходного кода, она должна быть откомпилирована с ключом `-g`. Посмотреть дизассемблированный код программы можно с помощью команды `disassemble` :

(gdb) `disassemble _start`.

Существует два режима отображения синтаксиса машинных команд: режим Intel, используемый в том числе в NASM, и режим ATT (значительно отличающийся внешне). По умолчанию в дизассемблере GDB принят режим ATT. Переключиться на отображение команд с привычным Intel’овским синтаксисом можно, введя команду `set disassembly-flavor intel`.

2.6 Точки останова

Установить точку останова можно командой `break` (кратко `b`). Типичный аргумент этой команды — место установки. Его можно задать как имя метки или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка».

Информацию о всех установленных точках останова можно вывести командой `info breakpoint`.

Для того чтобы сделать неактивной какую-нибудь ненужную точку останова, можно воспользоваться командой `disable`, а активировать командой `enable`.

Если же точка останова в дальнейшем больше не нужна, она может быть удалена с помощью команды `delete`.

2.7 Пошаговая отладка

Для продолжения остановленной программы используется команда `continue` (`c`) (`gdb`) с аргумент. Выполнение программы будет происходить до следующей точки останова. В качестве аргумента может использоваться целое число `N`, которое указывает отладчику проигнорировать `N-1` точку останова (выполнение остановится на `N`-й точке). Команда `stepi` (кратко `si`) позволяет выполнять программу по шагам, т.е. данная команда выполняет ровно одну инструкцию.

2.8 Работа с данными программы в GDB

Отладчик может показывать содержимое ячеек памяти и регистров, а при необходимости позволяет вручную изменять значения регистров и переменных. Посмотреть содержимое регистров можно с помощью команды `info registers`. Для отображения содержимого памяти можно использовать команду `x/NFU`, выдаёт содержимое ячейки памяти по указанному адресу. `NFU` задает формат, в котором выводятся данные.

2.9 Понятие подпрограммы

Подпрограмма — это, как правило, функционально законченный участок кода, который можно многократно вызывать из разных мест программы. В отличие от простых переходов из подпрограмм существует возврат на команду, следующую за вызовом.

2.10 Инструкция call и инструкция ret

Для вызова подпрограммы из основной программы используется инструкция `call`, которая заносит адрес следующей инструкции в стек и загружает в регистр `еір` адрес соответствующей подпрограммы, осуществляя таким образом переход. Подпрограмма завершается инструкцией `ret`, которая извлекает из стека адрес, занесённый туда соответствующей инструкцией `call`, и заносит его в `еір`. Подпрограмма может вызываться как из внешнего файла, так и быть частью основной программы.

3 Выполнение лабораторной работы

Создадим каталог для выполнения лабораторной работы №9, перейдем в него и создадим файл lab09-1.asm.(РИС.1)

```
antoychubekova@dk8n54 ~ $ mkdir ~/work/arch-pc/lab09  
antoychubekova@dk8n54 ~ $ cd ~/work/arch-pc/lab09  
antoychubekova@dk8n54 ~/work/arch-pc/lab09 $ touch lab09-1.asm
```

Рис. 3.1: РИС.1 Создание каталога

Введем в файл lab09-1.asm текст программы с использованием вызова подпрограммы.(РИС.2)

```

lab09-1.asm      [----]  0 L: [ 1+ 0  1/ 37] *(0  / 666b) 0037 0x025
%include "in_out.asm"
SECTION .data
msg: DB "Введите x: ",0
result: DB "2x+7=",0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
;-----
; Основная программа
;-----
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax, result
call sprint
mov eax, [res]
call iprintLF
call quit

; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы

```

Рис. 3.2: РИС.2 Редактирование файла

Создадим исполняемый файл и запустим его. Мы видим, что программа правильно работает и выдает правильно значение.(РИС.3)

```

antoyjchubekova@dk8n51 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
antoyjchubekova@dk8n51 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
antoyjchubekova@dk8n51 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 2
2x+7=11

```

Рис. 3.3: РИС.3 Запуск исполняемого файла

Изменим 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)

```

lab09-1.asm      [----]  3 L: [ 2+40  42/ 42] *(723 / 723b) <EOF>
SECTION .data
msg: DB 'Введите x: ',0
result: DB '2x+7=',0
SECTION .bss
x: RESB 80
res: RESB 80
SECTION .text
GLOBAL _start
_start:
;-----
; Основная программа
;-----
mov eax, msg
call sprint
mov ecx, x
mov edx, 80
call sread
mov eax, x
call atoi
call _calcul ; Вызов подпрограммы _calcul
mov eax, result
call sprint
mov eax, [res]
call iprintLF
call quit

; Подпрограмма вычисления
; выражения "2x+7"
_calcul:
call _subcalcul
mov ebx, 2
mul ebx
add eax, 7
mov [res], eax
ret ; выход из подпрограммы

_subcalcul:
mov ebx, 3
mul ebx
dec eax
ret

```

Рис. 3.4: РИС.4 Редактирование файла

Создадим исполняемый файл и запустим его. Проверим работу программы введя число 2 и видим, что все работает правильно.(РИС.5)

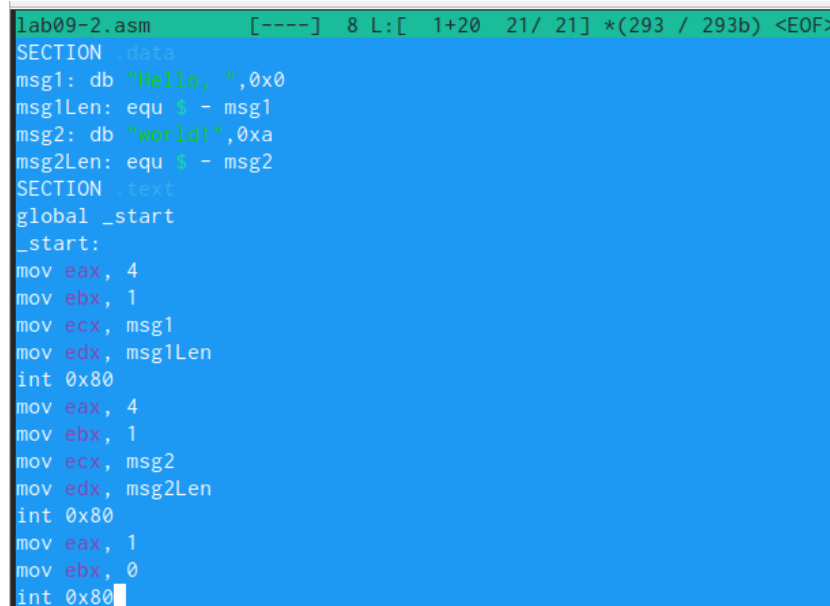
```

antoychubekova@dk8n54 ~/work/arch-pc/lab09 $ nasm -f elf lab09-1.asm
antoychubekova@dk8n54 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-1 lab09-1.o
antoychubekova@dk8n54 ~/work/arch-pc/lab09 $ ./lab09-1
Введите x: 2
2x+7=17

```

Рис. 3.5: РИС.5 Запуск исполняемого файла

Создадим файл lab09-2.asm и введем в него текст программы вывода сообщения Hello world!(РИС.6)



```
lab09-2.asm [----] 8 L: [ 1+20 21/ 21] *(293 / 293b) <EOF>
SECTION .data
msg1: db "Hello, ",0x0
msg1Len: equ $ - msg1
msg2: db "world!",0xa
msg2Len: equ $ - msg2
SECTION .text
global _start
_start:
mov eax, 4
mov ebx, 1
mov ecx, msg1
mov edx, msg1Len
int 0x80
mov eax, 4
mov ebx, 1
mov ecx, msg2
mov edx, msg2Len
int 0x80
mov eax, 1
mov ebx, 0
int 0x80
```

Рис. 3.6: РИС.6 Редактирование файла

Создадим исполняемый файл для работы с GDB с помощью команд:

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

Запустим исполняемый файл в отладчик gdb.(РИС.7)

```

antoyjchubekova@dk8n54 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-2.lst lab09-2.asm
antoyjchubekova@dk8n54 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-2 lab09-2.o
antoyjchubekova@dk8n54 ~/work/arch-pc/lab09 $ gdb lab09-2
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 lab09-2...
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/n/antoyjchubekova/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 34631) exited normally]
(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.

```

Рис. 3.7: РИС.7 Запуск исполняемого файла с gdb

Проверим работу программы, запустив ее в оболочке GDB с помощью команды run(РИС.8)

```

(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/n/antoyjchubekova/work/arch-pc/lab09/lab09-2
Hello, world!
[Inferior 1 (process 3498) exited normally]
(gdb)

```

Рис. 3.8: РИС.8 Проверка работы файла с помощью команды run

Для более подробного анализа программы установим breakpoint на метку _start и запустим ее.(РИС.9)

```

(gdb) break _start
Breakpoint 1 at 0x8049000: file lab09-2.asm, line 9.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/n/antoyjchubekova/work/arch-pc/lab09/lab09-2

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

```

Рис. 3.9: РИС.9 Установка breakpoint и запуск программы

Просмотрим дисассимилированный код программы с помощью команды disassemble начиная с метки _start. Далее переключимся на отображение команд с Intel'овским синтаксисом, введя команду set disassembly-flavor intel(РИС.10)


```

(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) 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.

```

Рис. 3.10: РИС.10 Переключение дисассимилированного кода на intel'овский синтаксис

В пежимк АТТ имена регистров начинаются с символа %, а имена операндов с \$, в то время как в intel используется привычный нам синтаксис.

Включим режим перфографики для более удобного анализа программы.(РИС.11)

```

[ 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 3531 In: _start
(gdb) layout regs
(gdb)

```

Рис. 3.11: РИС.11 Включение режима псевдологафики

Проверим установленный breakpoint по имени метки `_start` с помощью команды `info breakpoints`. (РИС.12)

```

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

```

Рис. 3.12: РИС.12 Информация о точке останова

Установим еще одну точку основа по адресу инструкции `mov ebx,0x0`. Просмотрим информацию о всех установленных точках останова введя `i b`. (РИС.13)

```

(gdb) break *0x8049031
Breakpoint 2 at 0x8049031: file lab09-2.asm, line 20.
(gdb) i b
Num      Type           Disp Enb Address      What
1        breakpoint     keep y   0x08049000 lab09-2.asm:9
          breakpoint already hit 1 time
2        breakpoint     keep y   0x08049031 lab09-2.asm:20
(gdb) █

```

Рис. 3.13: РИС.13 Установка точки останова и просмотр информации о всех точках останова

Выполним 5 инструкций с помощью команды `stepi` и наблюдаем за изменением значений регистров. (РИС.14) Мы видим, что изменились значение регистров `eax`, `ecx`, `edx`, `ebx` (РИС.15)

```

--Register group: general--
eax      0x4          4
ecx      0x0          0
edx      0x0          0
ebx      0x0          0
esp      0xffffc2d0   0xffffc2d0
ebp      0x0          0x0
esi      0x0          0
edi      0x0          0
eip      0x8049005     0x8049005 <_start+5>
eflags   0x202        [ IF ]
cs       0x23         35
ss       0x2b         43

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

native process 3531 In: _start L10 F
ebx      0x0          0
esp      0xffffc2d0   0xffffc2d0
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
--Type <RET> for more, q to quit, c to continue without paging--fs 0x0
gs       0x0          0
(gdb) si
(gdb) █

```

Рис. 3.14: РИС.14 Выполнение 5 инструкций с помощью команды `stepi`

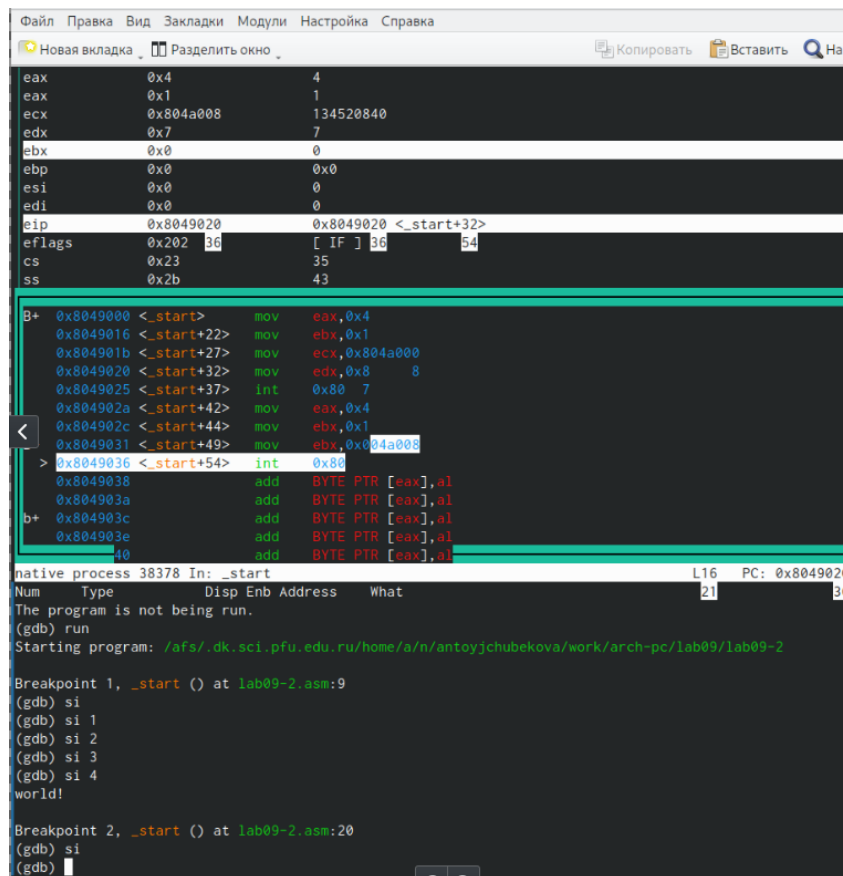


Рис. 3.15: РИС.15 Выполнение 5 инструкций с помощью команды stepi

Просмотрим значение переменной msg1 по имени используя команду x/1sb &msg1, также посмотрим значение переменной msg2 по ее адресу.(РИС.16)

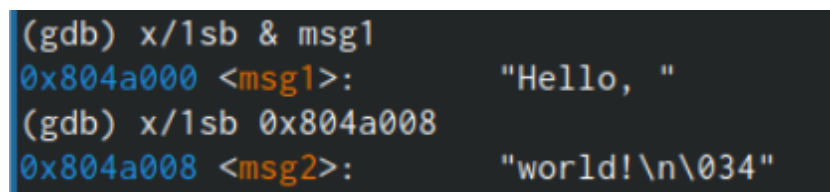


Рис. 3.16: РИС.16 Просмотр значений переменных msg1,msg2

Используя команду set изменим второй символ переменной msg1 на h.(РИС.17)

```
(gdb) set {char}&msg1='h'  
(gdb) set {char}0x804a001='h'  
(gdb) x/1sb &msg1  
0x804a000 <msg1>:      "hhlllo, "  
(gdb) █
```

Рис. 3.17: РИС.17 Изменение символа с помощью set

Заменим первый символ в переменной msg2 на f(РИС.18)

```
(gdb) set {char}&msg2='f'  
(gdb) x/1sb &msg2  
0x804a008 <msg2>:      "for1d!\n\034"
```

Рис. 3.18: РИС.18 Изменение символа msg2

Выведем значение регистра edx в шестнадцатиричном, двоичном формах и символьном виде.(РИС.19)

```

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x2      2
esp      0xffffc2d0 0xffffc2d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

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

native process 3531 In: _start
(gdb) x/1sb &msg1
0x804a000 <msg1>:      "hhlllo, "
(gdb) p/x $edx
$1 = 0x0
(gdb) p/t $edx
$2 = 0
(gdb) p/c $edx
$3 = 0 '\000'

```

Рис. 3.19: РИС.19 Вывод значения регистра в разных формах

С помощью команды set изменим значение регистра ebx в соответствии с заданием.(РИС.20)

```

Register group: general
eax      0x4      4
ecx      0x0      0
edx      0x0      0
ebx      0x2      2
esp      0xffffc2d0 0xffffc2d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x8049005 0x8049005 <_start+5>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

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

native process 3531 In: _start
(gdb) x/1sb &msg1
0x804a000 <msg1>: "hhllo, "
(gdb) p/x $edx
$1 = 0x0
(gdb) p/t $edx
$2 = 0
(gdb) p/c $edx
$3 = 0 '\000'
(gdb) set $ebx='2'
(gdb) p/c $ebx
$4 = 50 '2'
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb)

```

Рис. 3.20: РИС.20 Изменение значений регистра

Разница вывода команд p/s \$ebx отличается тем, что в первом случае мы переводим символ в его строковой вид, а во втором случае число в строковом виде не изменяется.

Завершим выполнение программы с помощью команды continue и выйдем из GDB с помощью команды quit.(РИС.21)

eax	0x4	4
eax	0x1	1
ecx	0x804a008	134520840
edx	0x7	7
esp	0xffffc2d0	ffffc2d0
ebp	0x0	0x0
esi	0x0	0
edi	0x0	0
eip	0x8049005	0x8049005 <_start+5>
eip	0x8049031	0x8049031 <_start+49>
cs	0x23	35
ss	0x2b	43


```

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
B+> 0x8049031 <_start+49> mov     ebx,0x0
0x8049036 <_start+54> int     0x80
0x8049038          add     BYTE PTR [eax],al
0x804903a          add     BYTE PTR [eax],al
0x804903c          add     BYTE PTR [eax],al
0x804903e          add     BYTE PTR [eax],al
40          add     BYTE PTR [eax],al

```

```

native process 3531 In: _start
(gdb) p/x $edx
$4 = 50 '2'
(gdb) set $ebx=2
(gdb) p/s $ebx
$5 = 2
(gdb) c
Continuing.
hhllo, forld!

Breakpoint 2, _start () at lab09-2.asm:20
(gdb) q
A debugging session is active.

    Inferior 1 [process 3531] will be killed.

Quit anyway? (y or n)

```

Рис. 3.21: РИС.21 Завершение работы GDB

Скопируем файл lab8-2.asm, созданный при выполнении лабораторной работы № 8 в файл с именем lab09-3.asm.(РИС.22)


```
antoyjchubekova@dk6n55 ~/work/arch-pc/lab09 $ cp ~/work/arch-pc/lab08/lab8-2.asm ~/work/arch-pc/lab09/lab09-3.asm
```

Рис. 3.22: РИС.22 Копирование файла

Создадим исполняемый файл.(РИС.23)

```
antoyjchubekova@dk6n55 ~/work/arch-pc/lab09 $ nasm -f elf -g -l lab09-3.lst lab09-3.asm
antoyjchubekova@dk6n55 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-3 lab09-3.o
```

Рис. 3.23: РИС.23 Создание исполняемого файла

Загрузим исполняемый файл в отладчик gdb, указав аргументы с использованием ключа `--args`(РИС.24)

```
antoyjchubekova@dk6n55 ~/work/arch-pc/lab09 $ gdb --args lab09-3 аргумент1 аргумент 2 'аргумент 3'
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. 3.24: РИС.24 Загрузка файла с аргументами в отладчик

Установим точку останова перед первой инструкцией в программе и запустим ее.(РИС.25)

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab09-3.asm, line 5.
(gdb) run
Starting program: /afs/.dk.sci.pfu.edu.ru/home/a/n/antoyjchubekova/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2
Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) █
```

Рис. 3.25: РИС.25 Установка точки останова и запуск программы

Посмотрим вершину стека и остальные позиции стека по их адресам. Шаг изменения адреса равен 4, потому что количество аргументов командной строки равно 4.(РИС.26)

```

(gdb) run
Starting program: /afs/.dk.scl.pfu.edu.ru/home/a/n/antoyjchubekova/work/arch-pc/lab09/lab09-3 аргумент1 аргумент 2 аргумент\ 3
Breakpoint 1, _start () at lab09-3.asm:5
5      pop ecx ; Извлекаем из стека в 'ecx' количество
(gdb) x/x $esp
0xfffffc280: 0x00000005
(gdb) x/s *(void**)(esp + 4)
0xfffffc518: "/afs/.dk.scl.pfu.edu.ru/home/a/n/antoyjchubekova/work/arch-pc/lab09/lab09-3"
(gdb) x/s *(void**)(esp + 8)
0xfffffc564: "аргумент1"
(gdb) x/s *(void**)(esp + 12)
0xfffffc576: "аргумент"
(gdb) x/s *(void**)(esp + 16)
0xfffffc587: "2"
(gdb) x/s *(void**)(esp + 20)
0xfffffc589: "аргумент 3"
(gdb) x/s *(void**)(esp + 24)
0x0: <error: Cannot access memory at address 0x0>
(gdb)

```

Рис. 3.26: РИС.26 Просмотр значений в стеке

3.1 Задание для самостоятельной работы

Скопировав в файл lab08-4.asm в новый файл lab09-4.asm преобразуем программу в этом файле так, чтобы вычисление значения функции реализовывалась как подпрограмма. (РИС.27) Сама программа выглядит следующим образом:

```
%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 podprogramma

    loop next
_end:

    mov eax, msg
call sprint
mov eax, esi
call iprintLF
call quit
    podprogramma:
inc eax
mov ebx,7
mul ebx
add esi,eax
ret.
```

```

lab09-4.asm      [-M--]  0 L: [ 1+40  41/ 41] *(359 / 359b) <EOF>
#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 podprogramma

loop next
_end:

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

podprogramma:
inc eax
mov ebx,7
mul ebx
add esi,eax
ret

```

Рис. 3.27: РИС.27 Редактирование файла

Создадим исполняемый файл и запустим его. Введя значения 1 2 3 4, мы видим, что программа работает правильно.(РИС.28)

```

antoychubekova@dk2n26 ~/work/arch-pc/lab09 $ nasm -f elf lab09-4.asm
antoychubekova@dk2n26 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-4 lab09-4.o
antoychubekova@dk2n26 ~/work/arch-pc/lab09 $ ./lab09-4 1 2 3 4
Результат: 98

```

Рис. 3.28: РИС.28 Запуск программы

Создаю файл lab09-5.asm и ввожу в него текст программы вычисления выражения $(3 + 2) * 4 + 5$ (РИС.29)

```
lab09-5.asm [-M--] 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. 3.29: РИС.29 Редактирование файла

Создадим исполняемый файл и запускаем ее. Мы видим, что программа работает неправильно, так как она выводит 10 вместо 25.(РИС.30)

```
antoyjchubekova@dk2n26 ~/work/arch-pc/lab09 $ nasm -f elf lab09-5.asm
antoyjchubekova@dk2n26 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-5 lab09-5.o
antoyjchubekova@dk2n26 ~/work/arch-pc/lab09 $ ./lab09-5
Результат: 10
```

Рис. 3.30: РИС.30 Запуск программы

Получим исполняемый файл для работы с GDB, запустим ее и ставим breakpoint во всех инструкциях, в которых происходит вычисления. С помощью команды `continue` проверяю каждую точку останова следя за значением регистров. При выполнении инструкции `mul ecx`, значение регистра `eax` должно было равняться 20, но она равнялась 8, это происходило потому что после выполнения инструкции `add ebx,eax` результат сложения был сохранен в регистр `ebx`, а не в регистр `eax`. В итоге после последней инструкции у нас получилось 10 так как программа прибавляет 5 как раз таки к регистру `ebx` которое равно 5.(РИС.31) и (РИС.32)

```
--Register group: general
eax      0x2      2
ecx      0x0      0
edx      0x0      0
ebx      0x3      3
esp      0xffffc2d0 0xffffc2d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490f2 0x80490f2 <_start+10>
eflags   0x202    [ IF ]
cs       0x23     35
ss       0x2b     43

B+ 0x80490e8 <_start>    mov     ebx,0x3
b+ 0x80490ed <_start+5>  mov     eax,0x2
B+> 0x80490f2 <_start+10> add     ebx,eax
b+ 0x80490f4 <_start+12>  mov     ecx,0x4
b+ 0x80490f9 <_start+17>  mul     ecx
b+ 0x80490fb <_start+19>  add     ebx,0x5
b+ 0x80490fe <_start+22>  mov     edi,ebx
0x8049100 <_start+24>    mov     eax,0x804a000
0x8049105 <_start+29>    call    0x804900f <sprintf>
0x804910a <_start+34>    mov     eax,edi
0x804910c <_start+36>    call    0x8049086 <fprintf>
0x8049111 <_start+41>    call    0x80490db <quit>
0x8049116 <_start+46>    add     BYTE PTR [eax],al

native process 3639 In: _start          L10  PC: 0x80490f2
(gdb) b *0x80490f2
Breakpoint 3 at 0x80490f2: file lab09-5.asm, line 10.
(gdb) b *0x80490f4
Breakpoint 4 at 0x80490f4: file lab09-5.asm, line 11.
(gdb) b *0x80490f9
Breakpoint 5 at 0x80490f9: file lab09-5.asm, line 12.
(gdb) b *0x80490fb
Breakpoint 6 at 0x80490fb: file lab09-5.asm, line 13.
(gdb) b *0x80490fe
```

Рис. 3.31: РИС.31 Постановка breakpoint

```

Register group: general
eax      0x8      8
ecx      0x4      4
edx      0x0      0
ebx      0xa      10
esp      0xffffc2d0 0xffffc2d0
ebp      0x0      0x0
esi      0x0      0
edi      0x0      0
eip      0x80490fe 0x80490fe <_start+22>
eflags   0x206    [ PF IF ]
cs       0x23     35
ss       0x2b     43

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

native process 3639 In: _start

Breakpoint 4, _start () at lab09-5.asm:11
(gdb) c
Continuing.

Breakpoint 5, _start () at lab09-5.asm:12
(gdb) c
Continuing.

Breakpoint 6, _start () at lab09-5.asm:13
(gdb) c
Continuing.

Breakpoint 7, _start () at lab09-5.asm:14
(gdb) 

```

Рис. 3.32: РИС.32 Наблюдения за значениями регистров

Исправляем ошибку, добавляя после add ebx,eax mov eax,ebx и заменяя ebx на eax в инструкциях add ebx,5 и mov edi,ebx.(РИС.33) Правильный код выглядит следующим образом:

```

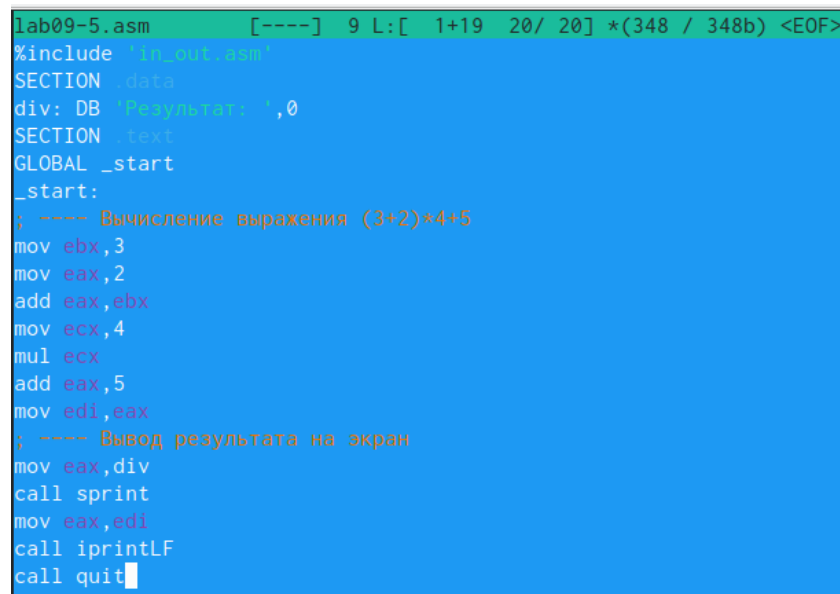
include 'in_out.asm'
SECTION .data
div: DB 'Результат:',0
SECTION .text
GLOBAL _start

```

```

_start:
; -- Вычисление выражения (3+2)*4+5
mov ebx,3
mov eax,2
add eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
; -- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit.

```



```

lab09-5.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 eax,ebx
mov ecx,4
mul ecx
add eax,5
mov edi,eax
; ---- Вывод результата на экран
mov eax,div
call sprint
mov eax,edi
call iprintLF
call quit

```

Рис. 3.33: РИС.33 Редактирование файла

Создадим исполняемый файл и запустим его. Мы видим, что программа рабо-

тает правильно и выводит 25.(РИС.34)

```
antoychubekova@dk8n51 ~/work/arch-pc/lab09 $ nasm -f elf lab09-5.asm
antoychubekova@dk8n51 ~/work/arch-pc/lab09 $ ld -m elf_i386 -o lab09-5 lab09-5.o
antoychubekova@dk8n51 ~/work/arch-pc/lab09 $ ./lab09-5
Результат: 25
```

Рис. 3.34: РИС.34 Запуск программы

4 Выводы

При выполнении данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм. Также познакомилась с методом отладки при помощи GDB и его основными возможностями. Используя полученные навыки я отредактировала программу, написанную мной при выполнении лабораторной работы №8 так, чтобы она работала с помощью подпрограмм. Вместе с тем я нашла ошибку в программе используя отладчик GDB.

Список литературы

-<https://esystem.rudn.ru/course/view.php?id=4975>.