Шаблон отчёта по лабораторной работе номер 9

Архитектура компьютеровт

Волчкова Елизавета Дмитриевна

Содержание

Цель работы

Приобретение навыков написания программ с использованием подпрограмм и знакомство с методами отладки при помощи GDB и его основными возможностями.

Задание

1. Преобразуйте программу из лабораторной работы №8 (Задание №1 для самостоятель- ной работы), реализовав вычисление значения функции $f(x)$ как подпрограмму.
2. 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

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. 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, отрабатывает, но не даёт желаемого результата; • ошибки в процессе выполнения — не обнаруживаются при трансляции и вызывают пре- рывание выполнения программы (например, это ошибки, связанные с переполнением или делением на ноль). Более подробно про Unix см. в [@tanenbaum_book_modern-os_ru; @robbins_book_bash_en; @zarrelli_book_mastering-bash_en; @newham_book_learning-bash_en].

Выполнение лабораторной работы

1. Создала каталог для выполнения лабораторной работы № 9, перешла в него и создала файл lab09-1.asm

2. В качестве примера рассмотрела программу вычисления арифметического выражения $f(x) = 2x + 7$ с помощью подпрограммы _calcul. В данном примере x вводилось с клавиатуры, а само выражение вычисляется в подпрограмме. Затем внимательно изучила текст программы (Листинг 9.1). Листинг 9.1. П 1

```
root@liza2006-VirtualBox:~/work/arch-pc/lab09# nasm -f elf32 lab9-1.asm -o obj.o

root@liza2006-VirtualBox:~/work/arch-pc/lab09# ld -m elf_i386 -o lab9-1 obj.o

root@liza2006-VirtualBox:~/work/arch-pc/lab09# ./lab9-1
Введите x: 15
2x+7=37

root@liza2006-VirtualBox:~#
```

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, выполнила инструкции подпрограммы:

```
  call atoi         ; Вычисляем f(g(x))
  call _calcul      ; Выводим результат
  mov eax, result
  call sprint
  mov eax, [res]
  call iprintLF
  call quit
calcul:             ; Вычисляем g(x) = 3x - 1
  call _subcalcul ; Теперь в eax хранится g(x), продолжаем вычисление f(g(x))
  mov ebx, 2
  mul ebx
  add eax, 7
  mov [res], eax
  ret
subcalcul:          ; g(x) = 3x - 1
  mov ebx, 3
  mul ebx
  sub eax, 1
```

Инструкция ret является последней в подпрограмме и ее исполнение приводит к воз- вращению в основную программу к инструкции, следующей за инструкцией call, которая вызвала данную подпрограмму.

```
root@liza2006-VirtualBox:~/work/arch-pc/lab09# touch lab9-2.asm


root@liza2006-VirtualBox:~/work/arch-pc/lab09# nasm -f elf32 lab9-2.asm -o obj.o

root@liza2006-VirtualBox:~/work/arch-pc/lab09# ld -m elf_i386 -o lab9-2 obj.o

root@liza2006-VirtualBox:~/work/arch-pc/lab09# ./lab9-2
Введите x: 15
f(g(x))=95
```

Последние строки программы реализуют вывод сообщения (call sprint), результата вы- числения (call iprintLF) и завершение программы (call quit).

Ввела в файл lab09-1.asm текст программы из листинга 9.1. Создала исполняемый файл и проверила его работу.

Потом изменила текст программы, добавив подпрограмму _subcalcul в подпрограмму _calcul, для вычисления выражения $f(g(x))$, где x вводится с клавиатуры, $f(x) = 2x + 7$, $g(x) = 3x - 1$. Т.е. x передается в подпрограмму _calcul из нее в подпрограмму _subcalcul, где вычисляется выражение $g(x)$, результат возвращается в _calcul и вычисляется выражение $f(g(x))$. Результат возвращается в основную программу для вывода результата на экран. 9.4.2. Отладка программам с помощью GDB Сначала создала файл lab09-2.asm с текстом программы из Листинга 9.2. (Программа печати сообщения Hello world!):

```
/root/work/arch-pc/lab09/lab09-2.asm   [----]  0 L:[  1+21  22/ 22] *(294 / 294b) <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
```

```
/root/work/arch-pc/lab09/lab09-2.lst
    1                                  SECTION .data
    2 00000000 48656C6C6F2C2000        msg1: db "Hello, ",0x0
    3                                  msg1Len: equ $ - msg1
    4 00000008 776F726C64210A          msg2: db "world!",0xa
    5                                  msg2Len: equ $ - msg2
    6                                  SECTION .text
    7                                  global _start
    8                                  _start:
    9 00000000 B804000000              mov eax, 4
   10 00000005 BB01000000              mov ebx, 1
   11 0000000A B9[00000000]            mov ecx, msg1
   12 0000000F BA08000000              mov edx, msg1Len
   13 00000014 CD80                    int 0x80
   14 00000016 B804000000              mov eax, 4
   15 0000001B BB01000000              mov ebx, 1
   16 00000020 B9[08000000]            mov ecx, msg2
   17 00000025 BA07000000              mov edx, msg2Len
   18 0000002A CD80                    int 0x80
   19 0000002C B801000000              mov eax, 1
   20 00000031 BB00000000              mov ebx, 0
   21 00000036 CD80                    int 0x80
```

```
root@liza2006-VirtualBox:~/work/arch-pc/lab09# touch lab09-2.asm

root@liza2006-VirtualBox:~/work/arch-pc/lab09# nasm -f elf -g -l lab09-2.lst lab09-2.asm -o obj9.o

root@liza2006-VirtualBox:~/work/arch-pc/lab09# ld -m elf_i386 -o lab09-2 obj9.o

root@liza2006-VirtualBox:~/work/arch-pc/lab09# ./lab09-2
Hello, world!
```

Получила исполняемый файл. Для работы с GDB в исполняемый файл необходимо добавила отладочную информацию, для этого трансляцию программ необходимо проводить с ключом ‘-g’.

```
root@liza2006-VirtualBox:~/work/arch-pc/lab09# gdb lab09-2
GNU gdb (Ubuntu 15.0.50.20240403-0ubuntu1) 15.0.50.20240403-git
Copyright (C) 2024 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 lab09-2...
(gdb)
```

```
(gdb) run
Starting program: /root/work/arch-pc/lab09/lab09-2

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.ubuntu.com>
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 separate debug info for system-supplied DSO at 0xf7ffc000
Hello, world!
[Inferior 1 (process 14040) exited normally]
(gdb)
```

Загрузила исполняемый файл в отладчик gdb: user@dk4n31:~$ gdb lab09-2 Проверила работу программы, запустив ее в оболочке GDB с помощью команды run (сокращённо r): (

Hello, world! [Inferior 1 (process 10220) exited normally] (gdb) Для более подробного анализа программы установила брейкпоинт на метку _start, с которой начинается выполнение любой ассемблерной программы, и запустила её. (gdb) break _start 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

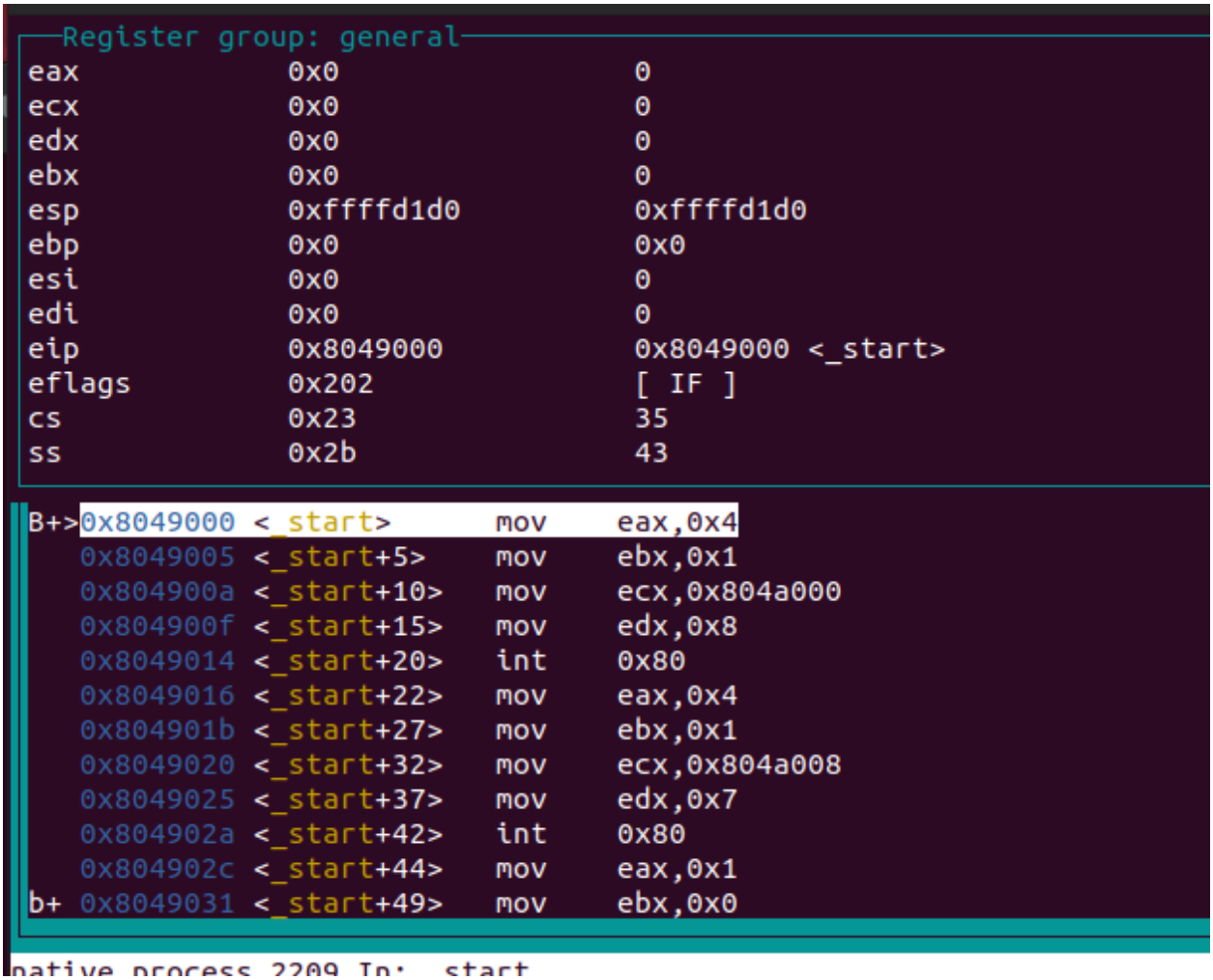
Посмотрела дисассимилированный код программы с помощью команды disassemble начиная с метки _start (gdb) disassemble _start Переключилась на отображение команд с Intel'овским синтаксисом, введя команду set disassembly-flavor intel (gdb) set disassembly-flavor intel (gdb) disassemble _start Перечислила различия отображения синтаксиса машинных команд в режимах ATT и Intel. Далее включила режим псевдографики для более удобного анализа программы (рис. 9.2): (gdb) layout asm (gdb) layout regs

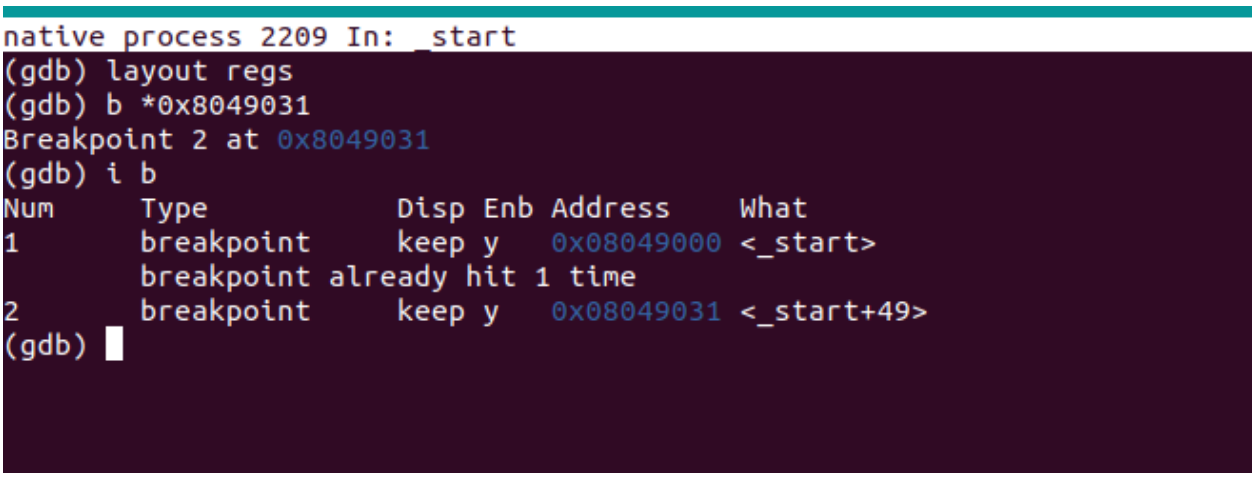
В этом режиме есть три окна:

• В верхней части видны названия регистров и их текущие значения; • В средней части виден результат дисассимилирования программы; • Нижняя часть доступна для ввода команд.

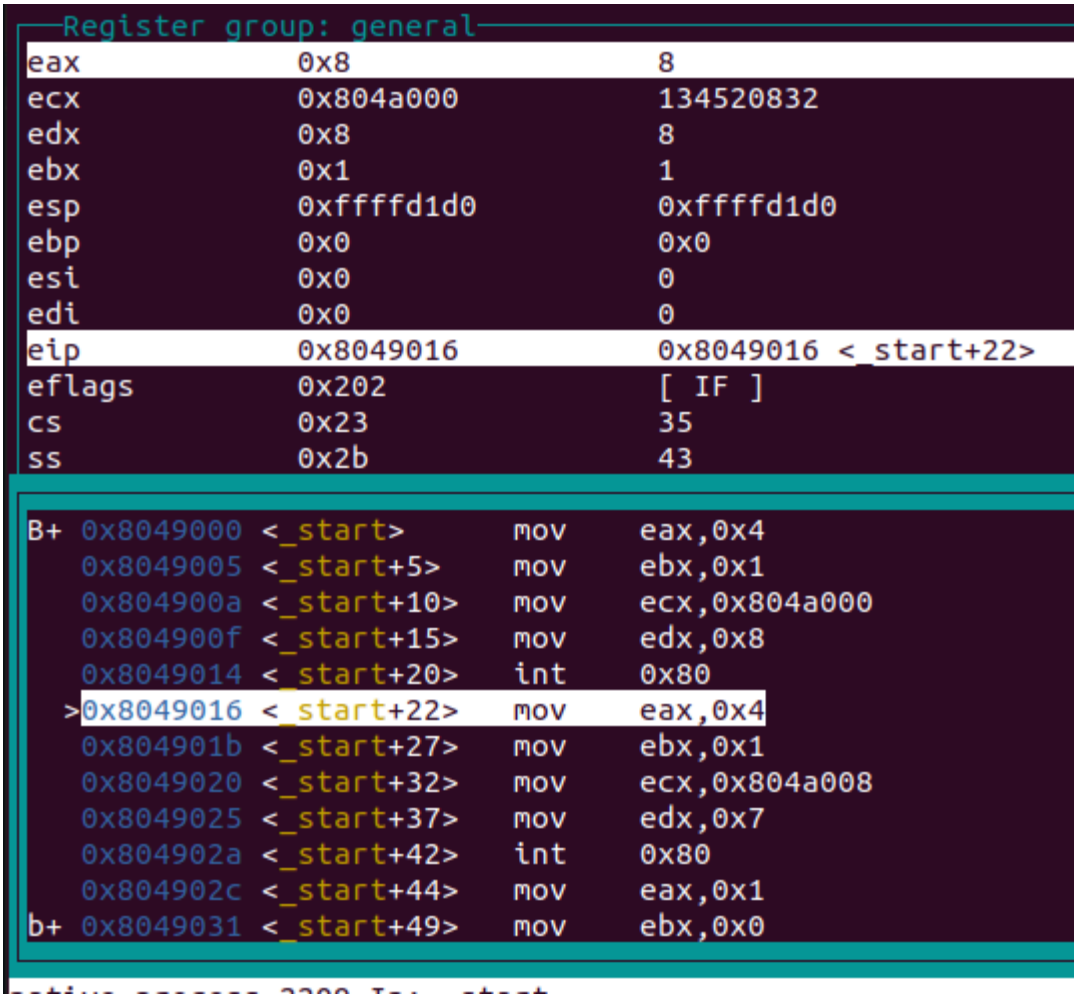
9.4.2.1. Добавление точек останова Установить точку останова можно командой break (кратко b). Типичный аргумент этой команды — место установки. Его можно задать или как номер строки программы (имеет смысл, если есть исходный файл, а программа компилировалась с информацией об отладке), или как имя метки, или как адрес. Чтобы не было путаницы с номерами, перед адресом ставится «звёздочка»:

На предыдущих шагах была установлена точка останова по имениметки (_start). Про- верьте это с помощью команды info breakpoints (кратко i b):

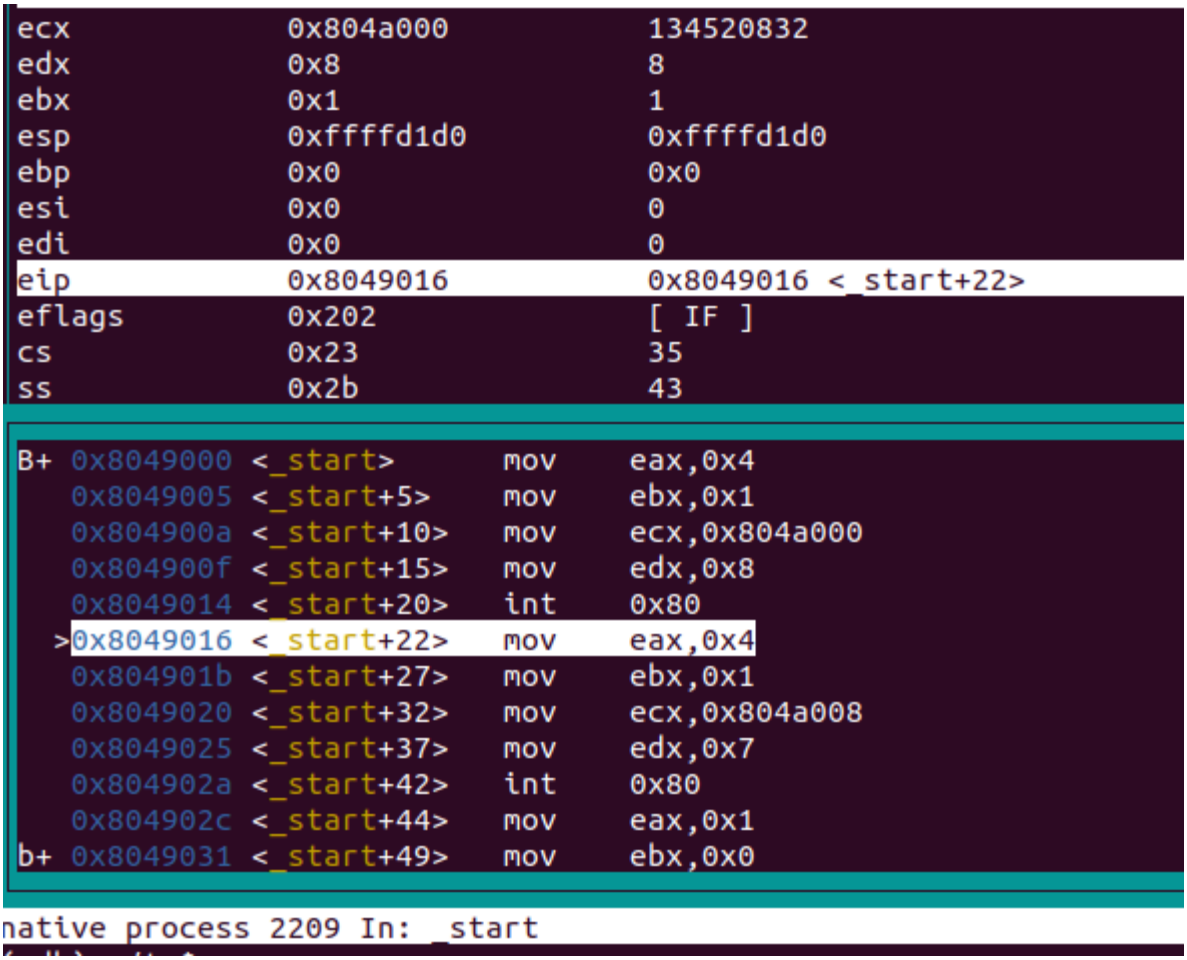


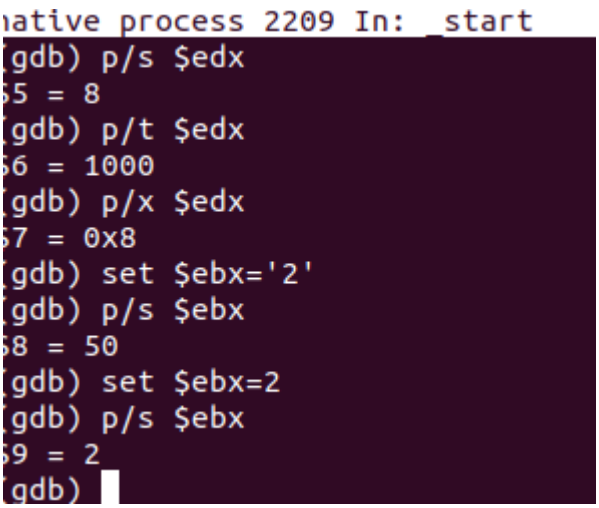


Посмотрела значение переменной msg1 по имени и посмотрела значение переменной msg2 по адресу.



Вывел в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx.





Скопировал файл lab8-2.asm, созданный при выполнении лабораторной работы №8, с программой выводящей на экран аргументы командной строки. Создал исполняемый файл. Для загрузки в gdb программы с аргументами необходимо использовать ключ –args. Загрузил исполняемый файл в отладчик, указав аргументы.

```
%include 'in_out.asm'
SECTION .text
global _start
_start:
pop ecx ; Извлекаем из стека в `ecx` количество
; аргументов (первое значение в стеке)
pop edx ; Извлекаем из стека в `edx` имя программ
; (второе значение в стеке)
sub ecx, 1 ; Уменьшаем `ecx` на 1 (количество
; аргументов без названия программы)
next:
cmp ecx, 0 ; проверяем, есть ли еще аргументы
jz _end ; если аргументов нет выходим из цикла
; (переход на метку `_end`)
pop eax ; иначе извлекаем аргумент из стека
call sprintLF ; вызываем функцию печати
loop next ; переход к обработке следующего
; аргумента (переход на метку `next`)
_end:
call quit
```

```
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://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 lab9-3...
(gdb) b _start
Breakpoint 1 at 0x80490e8
(gdb) run
Starting program: /home/baranovjora/work/arch-pc/lab09/lab9-3 argument 1 argument 2 argument\ 3

Breakpoint 1, 0x080490e8 in _start ()
(gdb) x/x $esp
0xffffd190:     0x00000006
(gdb)
0xffffd194:     0xffffd354
(gdb) x/s *(void**)($esp + 4)
0xffffd354:     "/home/baranovjora/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)($esp + 8)
0xffffd380:     "argument"
(gdb) x/s *(void**)($esp + 12)
0xffffd389:     "1"
(gdb) x/s *(void**)($esp + 16)
0xffffd38b:     "argument"
(gdb) x/s *(void**)($esp + 20)
0xffffd394:     "2"
(gdb) x/s *(void**)($esp + 24)
```

Затем для начала установила точку останова перед первой инструкцией в программе, запустила ее.

При условии, что дрес вершины стека храниться в регистре esp и по этому адресу располагается число равное количеству аргументов командной строки (включая имя программы). Число аргументов равно 5 – это имя программы lab9-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’.

Посмотрела остальные позиции стека – по адесу [esp+4] располагается адрес в памяти где находиться имя программы, по адесу [esp+8] храниться адрес первого аргумента, по аресу [esp+12] – второго и т.д.

```
fx: db  'f(x)= 2x+15 ',0

SECTION .text
global _start
_start:
mov eax, fx
call sprintLF
pop ecx
pop edx
sub ecx,1
mov esi, 0

next:
cmp ecx,0h
jz _end
pop eax
call atoi
call proc
add esi,eax

loop next

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

proc:
mov ebx,2
mul ebx
```

Приведенный ниже листинг программы вычисляет выражение (3+2) * 4+5 .

Программа дает неверный результат., проверив это я использовала отладчик GDB для анализа изменений значений регистров и определения ошибки.

```
 GNU nano 4.8
%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
```

```
 GNU nano 4.8
%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
```

Узнала, что орядок аргументов в инструкции add был перепутан и что при завершении работы, вместо eax, значение отправлялось в edi. Поэтому вот исправленнный код в файле task-2.asm

```
  GNU nano 4.8
%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
```

```
esi            0x0                0
edi            0x0                0
eip            0x80490fe          0x80490fe <_start+22>
eflags         0x202              [ IF ]
cs             0x23               35
ss             0x2b               43

B+ 0x80490e8 <_start>      mov    ebx,0x3
B+ 0x80490e8 <_start>5>    mov    ebx,0x3
   0x80490ed <_start+5>    mov    eax,0x2
   0x80490f2 <_start+10>   add    eax,ebx
   0x80490f4 <_start+12>   mov    ecx,0x4
   0x80490f9 <_start+17>   mul    ecx,0x5
  >0x80490fb <_start+19>   add    eax,0x5
   0x80490fe <_start+22>   mov    edi,eax
   0x8049100 <_start+24>   mov    eax,0x804a000rint>
   0x8049105 <_start+29>   call   0x804900f <sprint>
   0x804910a <_start+34>   mov    eax,edi86 <iprintLF>
   0x804910c <_start+36>   call   0x8049086 <iprintLF>
   0x8049111 <_start+41>   call   0x80490db <quit>
native process 2299 In: _start
(gdb) sNo process In:
(gdb) si
0x080490f4 in _start ()
(gdb) si
0x080490f9 in _start ()
(gdb) si
0x080490fb in _start ()
(gdb) si
0x080490fe in _start ()
(gdb) c
```

Установила точку останова перед первой инструкцией в программе и запустила 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 lab9-3 и непосредственно аргументы: аргумент1, аргумент, 2 и ‘аргумент 3’.

Посмотрела остальные позиции стека – по адесу [esp+4] располагается адрес в памяти где находиться имя программы, по адесу [esp+8] храниться адрес первого аргумента, по

Выводы

Целью работы было - приобретение навыков написания программ с использованием подпрограмм и знакомство с методами отладки при помощи GDB и его основными возможностями, сделав данные задания, я познакомилась с написанием программ при помощи GDB.

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. Новожилов О. П. Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM. — 2021. — URL: https://www.opennet.ru/docs/RUS/nasm/.
13. Робачевский А., Немнюгин С., Стесик О. Операционная система UNIX. — 2-е изд. — БХВ- Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. 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.

15. Таненбаум Э. Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
16. Таненбаум Э., Бос Х. Современные операционные системы. — 4-е изд. — СПб. : Питер,
17. — 1120 с. — (Классика Computer Science) ::: {#refs} :::