Отчёт по лабораторной работе №9

Понятие подпрограммы. Отладчик GDB.

Малкина Дарья Александровна

Содержание

Список иллюстраций

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

1. Сначала создадим каталог lab09 и файл lab9-1.asm. Рассмотрим программу, вычисляющую f(x)=2x+7 с помощью подпрограммы _calcul. Добавим в файл lab9-1.asm текст программы из листинга 9.1. Создадим исполняемый файл и проверим его работу:

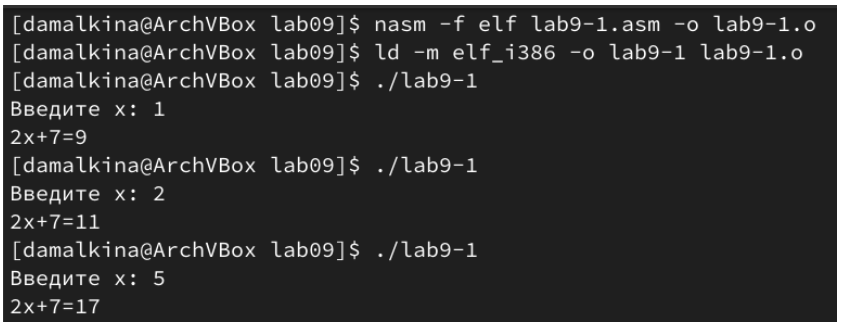

Рис. 2.1: Программа lab9-1 - f(x)=2x+7

После этого добавим в программу подпрограмму _subcalcul внутри _calcul, чтобы _subcalcul вычисляла g(x) = 3x-1, а после передавала значение x в _calcul, где будет вычислено f(g(x)):

```
;-----------------------------------------
; Подпрограмма вычисления выражения "2x+7"
;-----------------------------------------
_calcul:

call _subcalcul ; Вызов подпрограммы _subcalcul

mov ebx,2
mul ebx
add eax,7
mov [res],eax

ret             ; возвращению в основную программу

;-----------------------------------------
; Подпрограмма вычисления выражения "3x-1"
;-----------------------------------------
_subcalcul:
mov ebx,3
mul ebx
sub eax,1
mov [res],eax

ret             ; возвращению в основную программу
```

Рис. 2.2: Добавление подпрограммы _subcalcul

Создадим исполняемый файл и проверим его работу:

```
[damalkina@ArchVBox lab09]$ nasm -f elf lab9-1.asm -o lab9-1.o
[damalkina@ArchVBox lab09]$ ld -m elf_i386 -o lab9-1 lab9-1.o
[damalkina@ArchVBox lab09]$ ./lab9-1
Введите x: 1
f(g)=2(3x-1)+7=11
[damalkina@ArchVBox lab09]$ ./lab9-1
Введите x: 2
f(g)=2(3x-1)+7=17
[damalkina@ArchVBox lab09]$ ./lab9-1
Введите x: 5
f(g)=2(3x-1)+7=35
[damalkina@ArchVBox lab09]$
```

Рис. 2.3: Программа lab9-1 - f((g)x)

2.1 Отладка программам с помощью GDB

2. Создадим файл lab9-2.asm с программой из листинга 9.2 и соберем исполняемый файл. Добавим отладочную информацию, используя ключ -g:

```
[damalkina@ArchVBox lab09]$ nasm -f elf -g -l lab09-2.lst lab09-2.asm
nasm: fatal: unable to open input file `lab09-2.asm' No such file or directory
[damalkina@ArchVBox lab09]$ nasm -f elf -g -l lab9-2.lst lab9-2.asm
[damalkina@ArchVBox lab09]$ ld -m elf_i386 -o lab9-2 lab9-2.o
[damalkina@ArchVBox lab09]$ gdb lab9-2
GNU gdb (GDB) 15.2
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-pc-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 lab9-2...
```

Рис. 2.4: Отладка программы lab9-2 с помощью GDB

После этого загрузим исполняемый файл в отладчик GDB и проверим работу программы командой run:

```
(gdb) run
Starting program: /home/damalkina/work/arch-pc/lab09/lab9-2

This GDB supports auto-downloading debuginfo from the following URLs:
  <https://debuginfod.archlinux.org>
Enable debuginfod for this session? (y or [n]) y
Debuginfod has been enabled.
To make this setting permanent, add 'set debuginfod enabled on' to .gdbinit.
Hello, world!
[Inferior 1 (process 22489) exited normally]
```

Рис. 2.5: Запуск программы lab9-2 в оболчке GDB

Далее установим точку останова на метке _start командой break _start и запустим программу еще раз:

Рис. 2.6: Установка точки останова на метке _start

Затем посмотрим дизассемблированный код командой disassemble _start:

Рис. 2.7: Дизассемблированный код программы lab9-2

Переключимся на синтаксис Intel командой set disassembly-flavor intel и снова выведем дизассемблированный код:

Рис. 2.8: Дизассемблированный код - синтаксис Intel

Заметим, что отображения кода в режимах ATT и Intel различаются:

Режим ATT - операнды пишутся в порядке источник, затем приемник, при этом перед именем регистра ставится символ %, а непосредственные значения обозначаются знаком $ перед числом.

Режим Intel - операнды пишутся в порядке приемник, затем источник. Префикса % перед регистрами нет и нак $ не используется для обозначения непосредственных значений.

Включим режим псевдографики командами layout asm и layout regs:

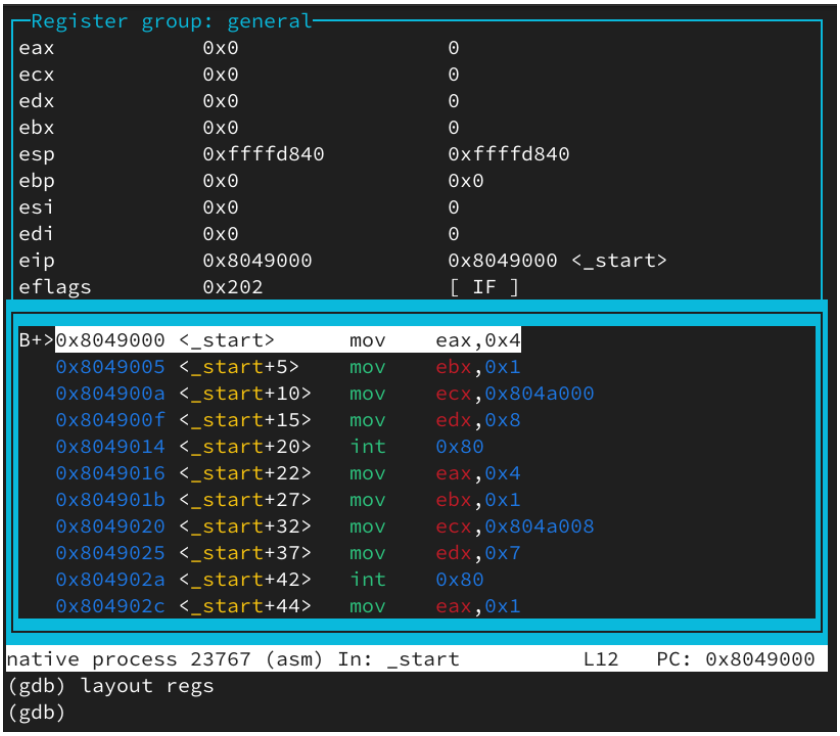

Рис. 2.9: Режим псевдографики

2.1.1 Добавление точек останова

До этого мы установили точку останова на метке _start, проверим это командой info breakpoints (i b). Затем по адресу инструкции установим еще одну точку останова командой break *. После посмотрим информацию о всех установленных точках останова командой i b:

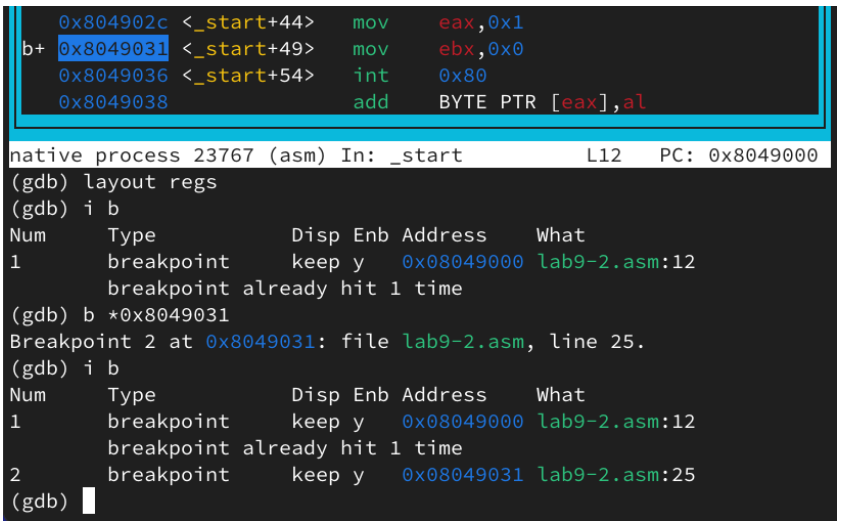

Рис. 2.10: Точки останова

2.1.2 Работа с данными программы в GDB

Воспользуемся командой stepi (si) пять раз, и понаблюдаем, как меняются значения регистров после каждой инструкции:

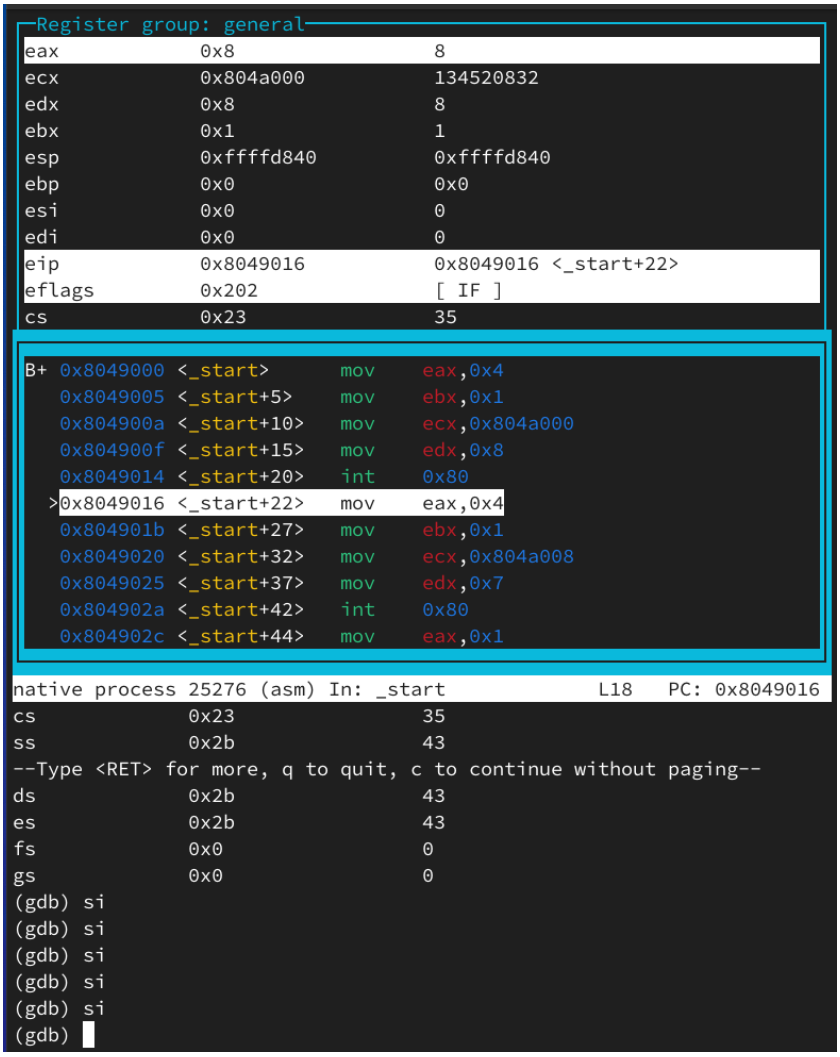

Рис. 2.11: Команда stepi

Воспользуемся командой x/1sb &msg1 для просмотра значения msg1. Затем мы найдем значение msg2 по адресу (определим адресс в дизассемблированной инструкции).

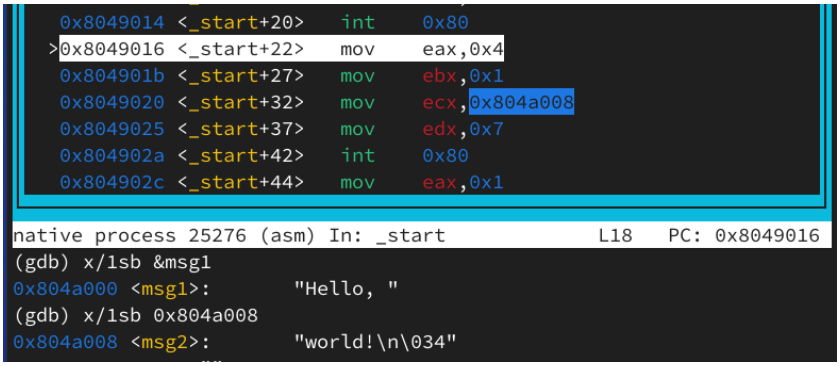

Рис. 2.12: Просмотр значений msg1 и msg2

Изменим первый символ msg1 командой set {char}msg1='h', после аналогично заменим первый символ в переменной msg2:

Изменение переменных - команда set

Рис. 2.13: Изменение переменных - команда set

Выведем в различных форматах (в шестнадцатеричном формате, в двоичном формате и в символьном виде) значение регистра edx:

Рис. 2.14: Команда print

С помощью команды set изменим значение регистра ebx сначала на '2',затем на 2, и после каждого изменения выведем значение edx как строку:

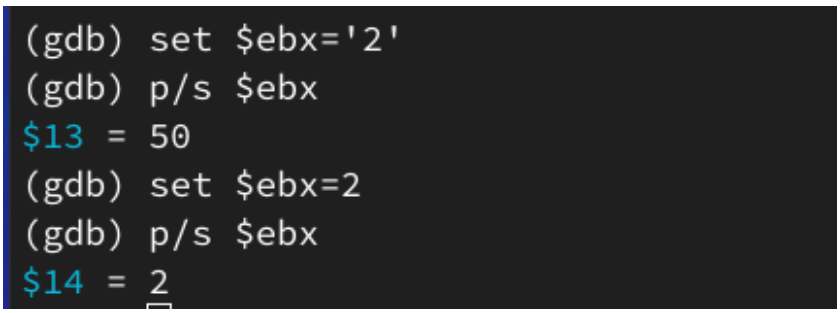

Рис. 2.15: Команда print

Заметим разницу в выводе:

В первом случае set $ebx='2' - мы присвоили регистру ebx строковое значение '2', поэтому команда p/s $ebx, которая выводит значение как строку, отображает ASCII-код символа "2", который равен 50.

Во втором случае set $ebx=2 - мы присвоили регистру ebx числовое значение 2, теперь команда p/s $ebx выводит число 2, потому что интерпретирует ebx как число, а не как строку.

Выходим из GDB.

2.1.3 Обработка аргументов командной строки в GDB

Скопируем файл lab8-2.asm в файл lab09-3.asm, создадим исполняемый файл:

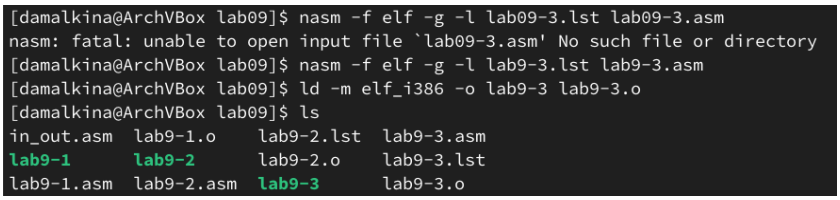

Рис. 2.16: Создаём файл lab9-3.as

После загрузим исполняемый файл в gdb с помощью ключа –args, передав ему аргументы:

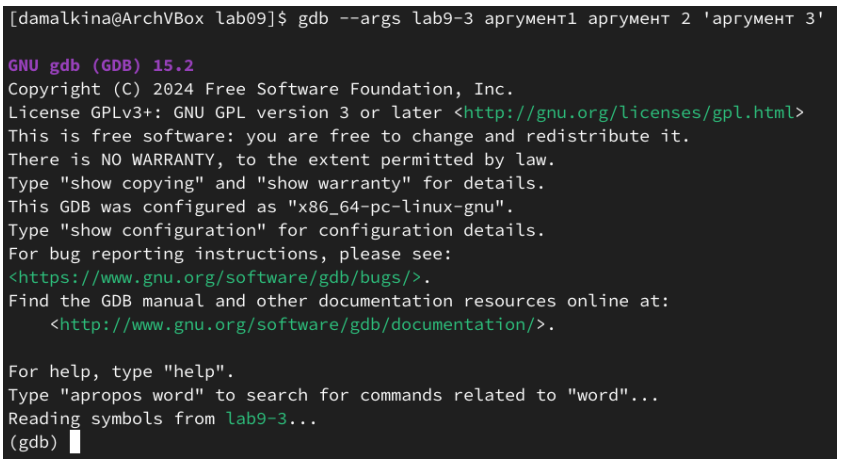

Рис. 2.17: Загружаем файл в GDB

Установим точку останова перед _start и запустим программу:

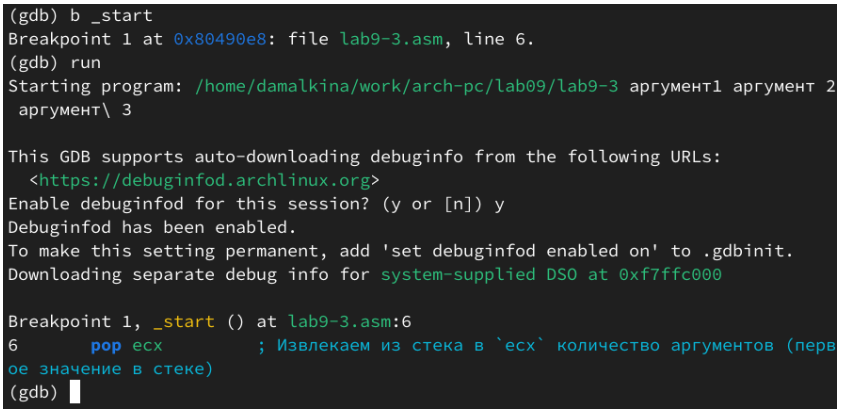

Рис. 2.18: Точка останова перед _start

Проверим количество аргументов с помощью команды x/x $esp. Результат - число 5, так как у нас есть имя программы и четыре аргумента. Далее, исследуем содержимое стека. По адресу [esp+4] находится адрес имени программы, по адресу [esp+8] — адрес первого аргумента, по адресу [esp+12] — второго, и так далее:

```
(gdb) x/x $esp
0xffffd800:     0x05
(gdb) x/s *(void**)($esp+4)
0xffffd9ba:     "/home/damalkina/work/arch-pc/lab09/lab9-3"
(gdb) x/s *(void**)($esp+8)
0xffffd9e4:     "аргумент1"
(gdb) x/s *(void**)($esp+12)
0xffffd9f6:     "аргумент"
(gdb) x/s *(void**)($esp+16)
0xffffda07:     "2"
(gdb) x/s *(void**)($esp+20)
0xffffda09:     "аргумент 3"
(gdb) x/s *(void**)($esp+24)
0x0:    <error: Cannot access memory at address 0x0>
(gdb)
```

Рис. 2.19: Содержимое стека

Так как в 32-битных системах, адреса памяти в виде 32-битными числами(32 бита = 4 байта), поэтому мы перемещаемся по стеку с интервалом в 4 байта вперед, чтобы добраться до начала следующего 32-битного адреса.

3 Выполнение задания для самостоятельной работы

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

В сегменте .bss объявим переменную res для хранения результатов вычислений:

```
 3 SECTION .data
 4 msg1 db "Функция: 4x+3",0
 5 msg2 db "Результат: ",0
 6
 7 SECTION .text
 8
 9 SECTION .bss
10 res: RESB 80
11
12 SECTION .text
13 GLOBAL _start
```

Рис. 3.1: Переменная res

В основной программе добавим вызов подрограммы _funcalcul (33 строка) и изменим операнды в сложении промежуточных результатов функции (35 строка):

```
;------------------------------------------
; Основная программа
;------------------------------------------
next:
cmp ecx,0h      ; проверяем, есть ли еще аргументы
jz _end         ; если аргументов нет выходим из цикла

pop eax         ; иначе извлекаем следующий аргумент из стека
call atoi       ; преобразуем символ в число

call _funcalcul ; вызов подпрограммы

add esi, [res]  ; складываем результат с предыдущими результатами
loop next       ; переход к обработке следующего аргумента

_end:
mov eax, msg1   ; вывод сообщения "Функция: 4x+3"
call sprintLF
mov eax, msg2   ; вывод сообщения "Результат: "
call sprint
mov eax, esi    ; записываем произведение в регистр `eax`
call iprintLF   ; печать результата
call quit       ; завершение программы
```

Рис. 3.2: Основная программа

Вынесем вычисления функции за основной код, добавим сохранение результата в переменную res (54 строка):

```
;------------------------------------------
; Подпрограмма вычисления выражения "4x+3"
;------------------------------------------
_funcalcul:
mov ebx, 4      ; 'ebx = 4'
mul ebx         ; домножаем '4' на аргумент 'eax=4*eax'
add eax, 3      ; прибавляем '3+4*eax'
mov [res],eax

ret             ; возвращению в основную программу
```

Рис. 3.3: Подпрограмма _funcalcul

Создаём исполнительный файл и запускаем его, проверим работу программы с разными аргументами, для простоты проверки введем те же аргументы, что и при проверке программы lab8-var5:

```
[damalkina@ArchVBox lab09]$ nasm -f elf lab9-var5.asm -o lab9-var5.o
[damalkina@ArchVBox lab09]$ ld -m elf_i386 -o lab9-var5 lab9-var5.o
[damalkina@ArchVBox lab09]$ ./lab9-var5 1 2 3 4
Функция: 4x+3
Результат: 52
[damalkina@ArchVBox lab09]$ ./lab9-var5 1 2 3
Функция: 4x+3
Результат: 33
[damalkina@ArchVBox lab09]$ ./lab9-var5 2 5 0
Функция: 4x+3
Результат: 37
[damalkina@ArchVBox lab09]$
```

Рис. 3.4: Программа lab9-var5

Для сравнения приведем результаты работы программы lab8-var5:

```
[damalkina@ArchVBox lab08]$ nasm -f elf lab8-var5.asm -o lab8-var5.o
[damalkina@ArchVBox lab08]$ ld -m elf_i386 -o lab8-var5 lab8-var5.o
[damalkina@ArchVBox lab08]$ ./lab8-var5 1 2
Функция: 4x+3
Результат: 18
[damalkina@ArchVBox lab08]$ ./lab8-var5 1 2 3 4
Функция: 4x+3
Результат: 52
[damalkina@ArchVBox lab08]$ ./lab8-var5 1 2 3
Функция: 4x+3
Результат: 33
[damalkina@ArchVBox lab08]$ ./lab8-var5 2 5 0
Функция: 4x+3
Результат: 37
[damalkina@ArchVBox lab08]$
```

Рис. 3.5: Программа lab8-var5

2. В листинге 9.3 приведена программа вычисления выражения (3+2)4+5. При запуске данная программа дает результат 10, что неверно, убедимся в этом, посчитав аналитически 5*4+5=20+5=25:

```
[damalkina@ArchVBox lab09]$ nasm -f elf lab9-4.asm -o lab9-4.o
[damalkina@ArchVBox lab09]$ ld -m elf_i386 -o lab9-4 lab9-4.o
[damalkina@ArchVBox lab09]$ ./lab9-4
Результат: 10
[damalkina@ArchVBox lab09]$
```

Рис. 3.6: Неверный результат

С помощью отладчика GDB, проанализируем изменения значений регистров и определим ошибку.

Соберем исполняемый файл, добавив отладочную информацию, после загрузим исполняемый файл в отладчик GDB и проверим работу программы:

```
[damalkina@ArchVBox lab09]$ nasm -f elf -g -l lab9-4.lst lab9-4.asm
[damalkina@ArchVBox lab09]$ ld -m elf_i386 -o lab9-4 lab9-4.o
[damalkina@ArchVBox lab09]$ gdb lab9-4
GNU gdb (GDB) 15.2
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-pc-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 lab9-4...
(gdb)
```

Рис. 3.7: Запуск программы lab9-2 в оболчке GDB

Установим точку останова на метке _start и запустим программу еще раз:

```
(gdb) b _start
Breakpoint 1 at 0x80490e8: file lab9-4.asm, line 14.
(gdb) r
Starting program: /home/damalkina/work/arch-pc/lab09/lab9-4

Breakpoint 1, _start () at lab9-4.asm:14
14      mov ebx,3
(gdb)
```

Рис. 3.8: Установка точки останова на метке _start

Затем посмотрим дизассемблированный код:

Рис. 3.9: Дизассемблированный код программы lab9-4

Включим режим псевдографики и добавим точки установа по адресу инструкции после арифметических команд:

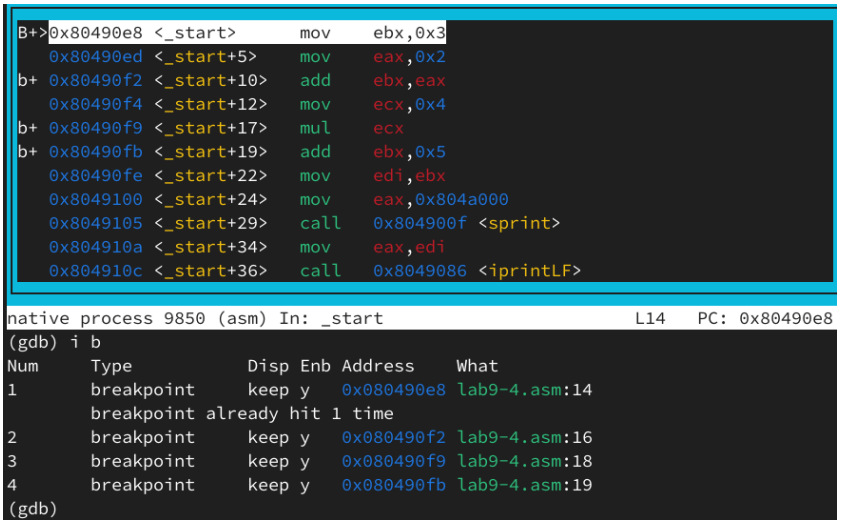

Рис. 3.10: Добавление точек останова

Воспользуемся командами stepi и print, и понаблюдаем, как меняются значения регистров после инструкций. После инструкции add ebx,eax, регистры принимают верные значения:

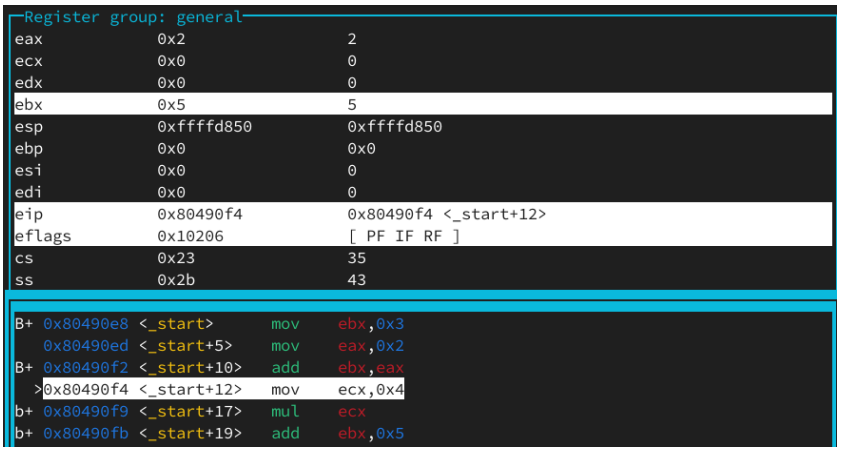

Рис. 3.11: Значения регистров после add ebx,eax

Заметим, что после команды mul ecx, значение ebx остаётся неизменным, хотя должно принимать значение 5*4=20, но меняется значение eax, хотя должно оставаться равным 2. Делаем вывод, что инструкция mul ecx умножает содержимое регистра eax на ecx, а не ebx на ecx, как задумано, результат записывается в eax:

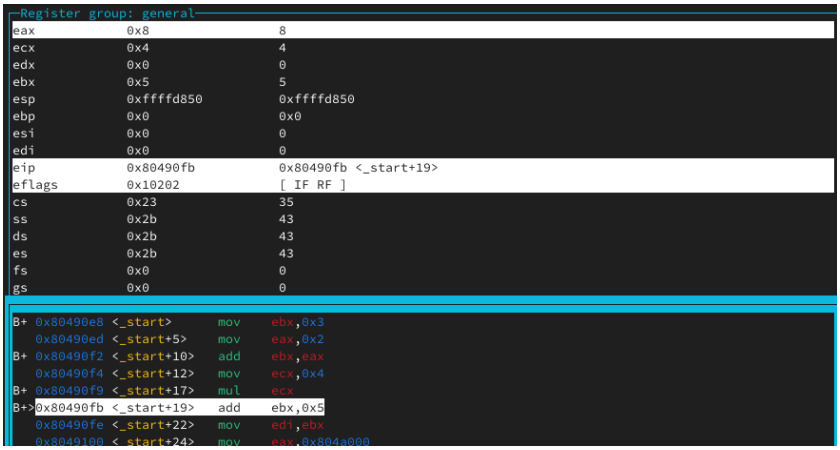

Рис. 3.12: Первая ошибка - неправильное использование инструкции mul

С помощью команды print сравним значения регистров eax, ebx, ecx до команды mul ecx и после:

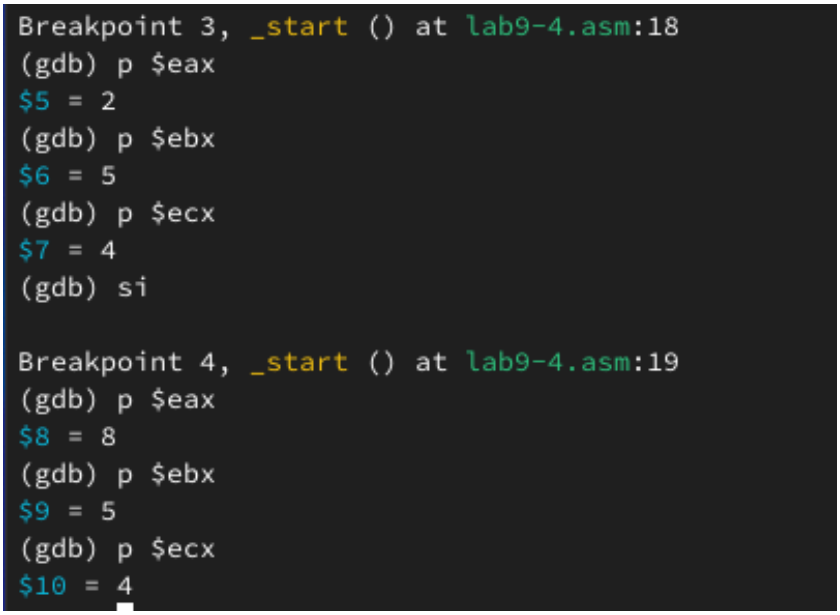

Рис. 3.13: Значения регистров

Чтобы исправить это, нужно сначала скопировать содержимое ebx в eax перед выполнением mul ecx. Однако код всё равно неверный из-за того, что после умножения 5 прибавляется к ebx, а не к результату умножения, который находится в eax. Это вторая ошибка - неправильное место для второго сложения:

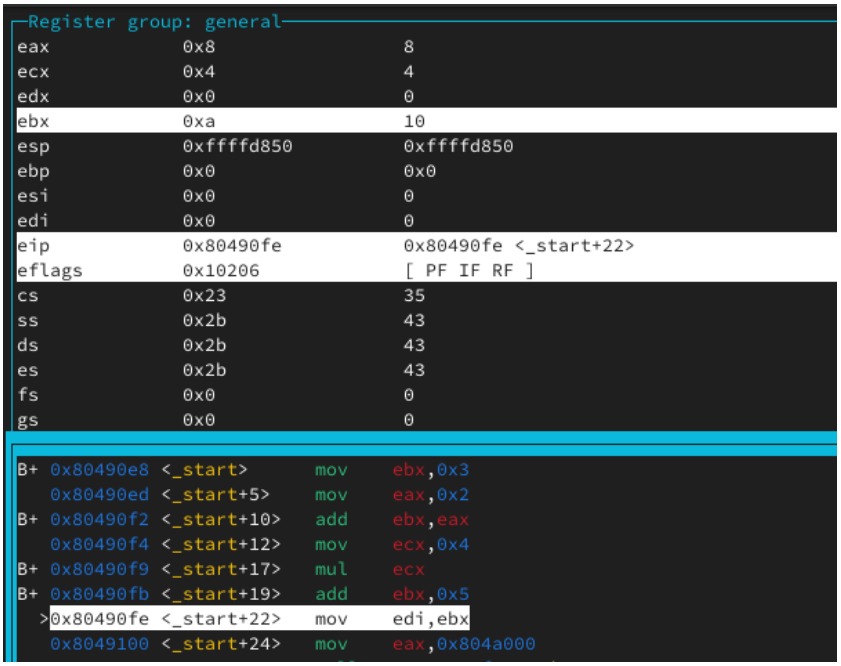

Рис. 3.14: Значения регистров после add edi,ebx

Исправим код, скопируем значение ebx в eax перед выполнением mul ecx и прибавим 5 напрямую к eax, после чего сохраним результат в edi:

```
; ------------------------------------
;   Вычисление выражения (3+2)*4+5
; ------------------------------------
mov ebx,3
mov eax,2
add ebx,eax
mov ecx,4
mov eax, ebx     ; содержимое из ebx в eax
mul ecx
add eax,5        ; 5+(eax * ecx)
mov edi,eax
```

Рис. 3.15: Исправленный код программы lab9-4

Создим исполнительный файл и запустим его, для проверки работы программы, убедимся, что после внесения изменений программа работает корректно:

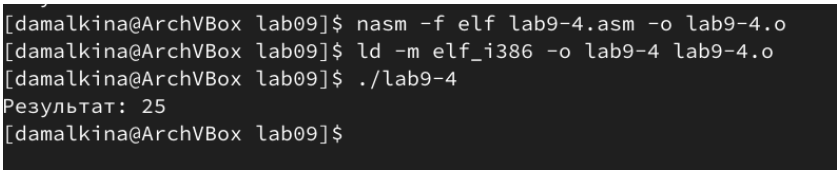

Рис. 3.16: Корекктная работа программы lab9-4

4 Выводы

В ходе лабораторной работы и выполнения самостоятельного задания мы приобрели практические навыки написания ассемблерных программ с использованием подпрограмм, попрактиковались в отладке программ с помощью GDB, научились использовать GDB для пошагового выполнения кода, анализа значений регистров и памяти, что позволило нам обнаружить и исправить ошибки в программе.