Отчёт по лабораторной работе №10

Подпрограммы. Отладчик GDB

Мулин Иван Владимирович

Содержание

# 1 Цель работы

Цель работы - изучить написание программ, использующих подпрограммы, а также ознакомиться с основными возможностями отладчика GDB. Репозиторий автора расположен по адресу <https://github.com/ivmulin/study_2022-2023_arch-pc>.

# 2 Ход работы

## 2.1 Выполнение лабораторной работы

### 2.1.1 Работа с подпрограммами

Напишем программу lab10-1.asm, которая использует подпрограмму для вычисления значения функции в зависимости от введённого значения аргумента:

Значение выражения f(x)=2x+7

Значение выражения f(x)=2x+7

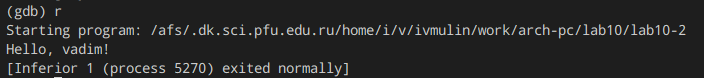
Перепишем эту программу так, чтобы она при помощи подпрограмм выводила значение выражения , где :

Значение выражения f(g(x))

Значение выражения

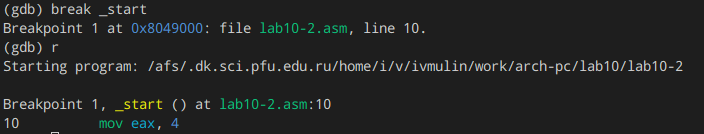
### 2.1.2 Отладчик GDB

Запустим программу lab10-2.asm в отладчике GDB:



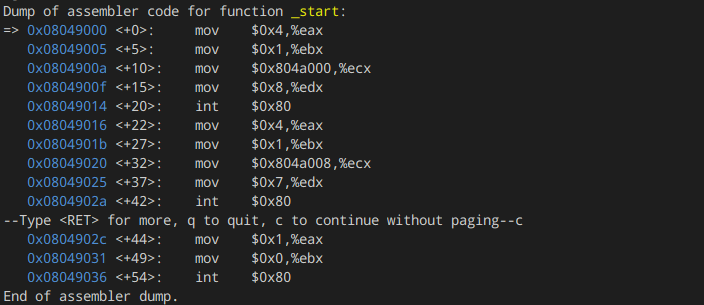
Запуск программы в отладчике

Далее установим точку останова на метке \_start:



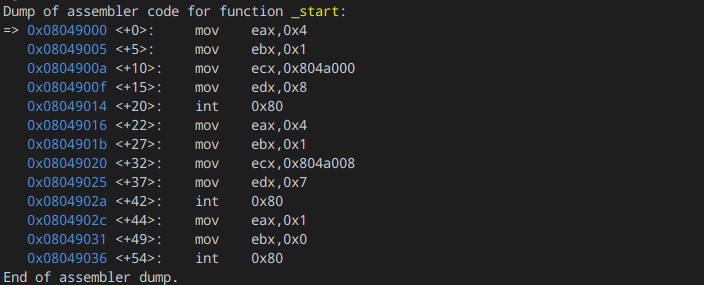
Установка первой точки останова

Дизассемблируем программу, начиная с метки \_start:



Дизассемблирование программы в режиме ATT

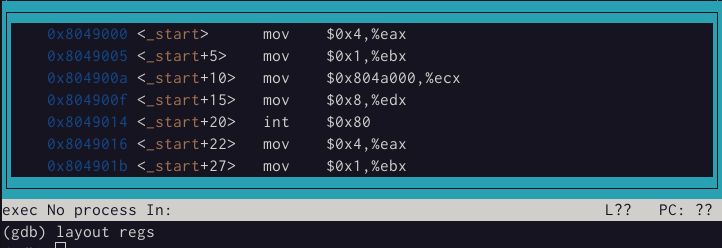
Изначально дизассемблированный код отображается в стиле ATT. Переключим его на Intel:



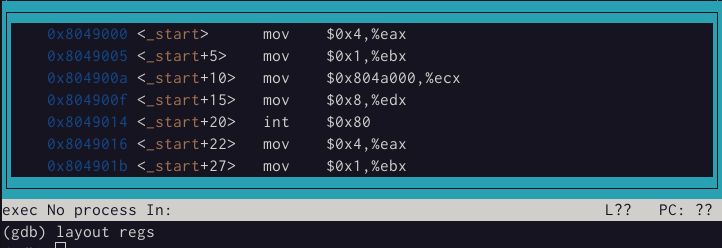
Дизассемблирование программы в режиме Intel

Как видно, отображение в стиле ATT и в стиле Intel отличаются: к примеру, дизассемблированный код ATT устанавливает символ $ перед ячёками памяти и числами и % перед названием регистра, чего не делает отображение Intel. Более того, порядок операндов в инструкциях с двумя операндами (таких, как, например, mov или add) различен в разных видах отображений.

Отобразим окно регистров при помощи команды layout regs и layout asm:



layout regs



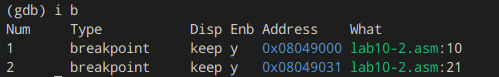
layout asm

Выведем информацию обо всех добавленных точках отсанова:

Просмотр точек останова

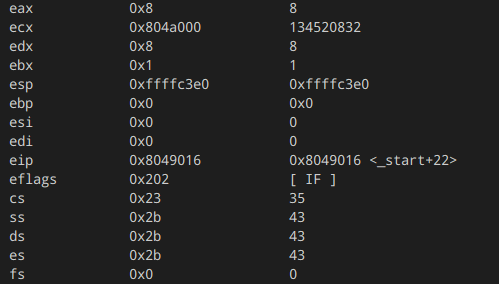
Просмотр точек останова

Установим точку останова по адресу предпоследней инструкции в программе:



Обзор новых точек останова

При помощи GDB можно просматривать информацию о регистрах:



Обзор значений регистров

Кроме того, можно получать значение памяти по нужному адресу:

Просмотр значения строки msg1

Просмотр значения строки msg1

Для печати обращение к памяти можно выполнять с использованием адресов:

Обращение к строке msg2 по её адресу

Обращение к строке msg2 по её адресу

Используя команду set, заменим первую букву в строке msg1. Попутно изменим вторую букву в msg2.

Замена буквы в msg1

Замена буквы в msg1

Замена буквы в msg2

Замена буквы в msg2

Выведем в шестнадцатиричном, двоичном и символьном форматах значение регистра edx:



Значение регистра edx в разных форматах

Изменим значение регистра edx сначала на '2', затем на 2. Значения '2' и 2 отличаются, и отладчик выводит их коды в таблице ASCII.



Изменение регистра edx

Напишем программу lab10-3.asm и запустим её в отладчике при помощи команды

gdb --args lab10-3 аргумент1 аргумент 2 'аргумент 3'

Обзор значений регистров

Обзор значений регистров

В регистре esp хранится число, равное количеству переданных аргументов. Остальные значения стека можно просмотреть при помощи четырёхкратного инкремента (к примеру, [$esp+4], [$esp+8]). Инкремент равен четырём из-за того, что аргументы хранятся как двойное слово, то есть занимают объём в 4 байта.

## 2.2 Выполнение заданий для самостоятельной работы

Скорректируем код программы из задания 1 самостоятельной работы к лабораторной работе № 9 так, чтобы значение функции вычислялось в отдельной подпрограмме:

Работающая программа lab10-4

Работающая программа lab10-4

Скопируем текст программы, вычисляющей значение выражения . В ней, очевидно, допущена ошибка. Текст исправленной программы представлен ниже в качестве программы lab10-5.asm

# 3 Листинги написанных программ

1. lab10-1.asm

%include 'in\_out.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:  
 mov eax, msg  
 call sprint  
   
 mov ecx, x  
 mov edx, 80  
 call sread  
  
 mov eax, x  
 call atoi  
  
 call \_calcul  
  
 mov eax, result  
 call sprint  
  
 mov eax, [res]  
 call iprintLF  
 call quit  
   
  
\_calcul:  
 ; f(x) = 2x + 7  
 call \_subcalcul  
 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  
 ret

1. lab10-2.asm

section .data  
 msg1: db "Hello, ", 0x0  
 msg1Len: equ $-msg1  
 msg2: db "vadim!", 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

1. lab10-3.asm

%include 'in\_out.asm'  
  
section .text  
global \_start  
  
\_start:  
 pop ecx  
 pop edx  
 sub ecx, 1  
   
next:  
 cmp ecx, 0  
 jz \_end  
  
 pop eax  
 call sprintLF  
 loop next  
  
\_end:  
 call quit

1. lab10-4.asm

%include 'in\_out.asm'  
  
; f(x) = 15x + 2  
  
section .data  
 msg db "Результат: ", 0  
 fun db "f(x) = 15 \* x + 2", 10  
  
section .bss  
 result resb 10  
  
section .text  
 global \_start  
  
\_start:  
 mov eax, fun  
 call sprint  
  
 pop ecx ; Извлекаем из стека в `ecx` количество аргументов  
 pop edx ; Извлекаем из стека в `edx` имя программы  
 sub ecx, 1 ; Уменьшаем `ecx` на 1 (количество аргументов без названия программы)  
 mov esi, 0 ; используем 'esi' для хранения промежуточных сумм  
  
 mov eax, 0  
 mov [result], eax  
  
extractArguments:  
 cmp ecx, 0 ; проверяем, есть ли еще аргументы  
 jz \_end ; если аргументов нет, выходим из цикла  
  
 pop eax  
 call atoi  
  
 call funkcia  
 add [result], eax  
  
 loop extractArguments  
  
\_end:  
 mov eax, msg  
 call sprint  
  
 mov eax, [result]  
 call iprintLF  
  
 call quit  
  
funkcia:  
 mov ebx, 15  
 mul ebx  
 add eax, 2

1. lab10-5.asm

%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

# 4 Заключение

Поставленная в начале данного отчёта цель была, очевидно, достигнута, ведь был освоён процесс использования подпрограмм в языке ассемблера NASM и отладчика GDB.