Лабораторная работа №9

Дисциплина: Архитектура компьютера

Малюга Валерия Васильевна

Содержание

Список иллюстраций

1 Цель работы

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

2 Задание

1. Реализация подпрограмм в NASM
2. Отладка программ с помощью GDB 2.1 Добавление точек останова. 2.2 Работа с данными программы в GDB 2.3 Обработка аргументов командной строки в GDB
3. Задания для самостоятельной работы

3 Теоретическое введение

Отладка — это процесс поиска и исправления ошибок в программе. Отладчики позволяют управлять ходом выполнения программы, контролировать и изменять данные. Это помогает быстрее найти место ошибки в программе и ускорить её исправление. Наиболее популярные способы работы с отладчиком — это использование точек останова и выполнение программы по шагам.

GDB (GNU Debugger — отладчик проекта GNU) работает на многих UNIXподобных 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.

Отладчик GDB (как и любой другой отладчик) позволяет увидеть, что происходит «внутри» программы в момент её выполнения или что делает программа в момент сбоя.

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

Команда kill (сокращённо k) прекращает отладку программы, после чего следует вопрос о прекращении процесса отладки. 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).
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.
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 (кратко i).
Для того чтобы сделать неактивной какую-нибудь ненужную точку останова, можно воспользоваться командой disable.
Обратно точка останова активируется командой enable.
Если же точка останова в дальнейшем больше не нужна, она может быть удалена с помощью команды delete.
Для продолжения остановленной программы используется команда continue (c). 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.
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. Если в программе встречается одинаковый участок кода, его можно оформить в виде подпрограммы, а во всех нужных местах поставить её вызов.

При этом подпрограмма будет содержаться в коде в одном экземпляре, что позволит уменьшить размер кода всей программы.

Для вызова подпрограммы из основной программы используется инструкция call, которая заносит адрес следующей инструкции в стек и загружает в регистр eip адрес соответствующей подпрограммы, осуществляя таким образом переход. Затем начинается выполнение подпрограммы, которая, в свою очередь, также может содержать подпрограммы. Подпрограмма завершается инструкцией ret, которая извлекает из стека адрес, занесённый туда соответствующей инструкцией call, и заносит его в eip. После этого выполнение основной программы возобновится с инструкции, следующей за инструкцией call.

4 Выполнение лабораторной работы

4.1 Реализация подпрограмм в NASM

Создала каталог для программ лабораторной работы № 8, перешла в него и создала файл lab8-1.asm (рис. **??**).

Создание каталога и файла lab8-1.asm

4.2 Отладка программ с помощью GDB

4.2.1 Добавление точек останова

4.2.2 Работа с данными программы в GDB

4.2.3 Обработка аргументов командной строки в GDB

4.3 Задание для самостоятельной работы

Написала программу, которая находит сумму значений функции f(x) = 15x + 2 (Вариант 11) для x = x1, x2, ..., xn. Создала исполняемый файл и проверила его работу на нескольких наборах x = x1, x2, ..., xn (рис. **??**).

Запуск исполняемого файла и проверка его работы

Прилагаю код:

```
%include 'in_out.asm'
SECTION .data
```

```
    msg db "Результат: ",0
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 программы
                ; (второе значение в стеке)
    sub ecx,1   ; Уменьшаем `ecx` на 1 (количество
                ; аргументов без названия программы)
    mov esi, 0 ; Используем `esi` для хранения
                ; промежуточных сумм
next:
    cmp ecx,0h ; проверяем, есть ли еще аргументы
    jz _end     ; если аргументов нет выходим из цикла
                ; (переход на метку `_end`)
    pop eax     ; иначе извлекаем следующий аргумент из стека
    call atoi   ; преобразуем символ в число

    imul eax, 15     ; умножаем x на 15
    add eax, 2       ; добавляем 2
    add esi,eax      ; добавляем значение функции для
                     ; конкретного аргумента к промежуточной сумме
    loop next        ; переход к обработке следующего аргумента
_end:
    mov eax,msg     ; вывод сообщения "Результат: "
    call sprint
    mov eax,esi     ; записываем сумму в регистр `eax`
    call iprintLF   ; печать результата
```

```
call quit       ; завершение программы
```

5 Выводы

Во время выполнения данной лабораторной работы я приобрела навыки написания программ с использованием подпрограмм и ознакомилась с методами отладки при помощи GDB и его основными возможностями.