Отчет по ИДЗ №3 по Архитектуре вычислительных систем.
Вариант 12.
Харитонов Кирилл БПИ 214

12. Разработать программу, вычисляющую с помощью степенного ряда
с точностью не хуже 0,05% значение функции tan(x) для заданного
параметра x.

Необходимая точность достигается при входных значениях от -1.2 до 1.2 и аналогичных им в силу периодичности. В значениях близких к pi/2 и -pi/2 точность является недостаточной в силу ограничений на максимальный размер хранимого числа (в данном случае нужно считать большой факториал, который не влезает в long double). Данное ограничение связано с используемым по условию задачи численным методом.

Претендую на оценку 9.

- Тесты, демонстрирующие проверку программ содержатся в Makefile по директивам test_default и test_assembly.
- Результаты их прогонов:
  Содержание файла input1: 1.213124, содержание файла input2: -34.32.
  test_default:
  ./default < input1
  Enter x: tan(x) = 2.675601
  Time taken to execute this program: 0.000279
  ./default < input2
  Enter x: tan(x) = 0.242089
  Time taken to execute this program: 0.000281
  ./default -f input1:output1
  Time taken to execute this program: 0.000313
  cat output1
  tan(x) = 2.675601
  ./default -f input2:output2
  Time taken to execute this program: 0.000276
  cat output2
  tan(x) = 0.242089
  ./default -r 1
  Following floating point number was generated: 0.176052
  tan(x) = 0.177894
  Time taken to execute this program: 0.000302

  test_assembly:
  ./assembly < input1
  Enter x: tan(x) = 2.675601
  Time taken to execute this program: 0.000274
  ./assembly < input2

Enter x: tan(x) = 0.242089
Time taken to execute this program: 0.000268
./assembly -f input1:output1
Time taken to execute this program: 0.000303
cat output1
tan(x) = 2.675601
./assembly -f input2:output2
Time taken to execute this program: 0.000274
cat output2
tan(x) = 0.242089
./assembly -r 1
Following floating point number was generated: 0.190631
tan(x) = 0.192974
Time taken to execute this program: 0.000269

- Исходные тексты программы на C содержатся в файлах main.c и functions.c
- Исходные тексты программы на ассемблере содержатся в файлах main.s и functions.s
- Текст на ассемблере программы, полученный после компиляции программы на C представлен в файлах main_original.s и functions_original.s

Критерии:
4 балла

- Исходные тексты программы на C **содержатся в файлах main.c и functions.c,** исходные тексты программы на asm **содержатся в файлах main_original.s и functions_original.s**
- Добавлены комментарии, представляющие эквивалентное представление переменных в программе на ассемблере. **Например,**

```
mov r12d, 0 # r12d = i = 0
add r12d, 1 # i++
```

- Из ассемблерной программы убраны лишние макросы за счет использования соответствующих аргументов командной строки (откомпилирована с аргументами -S -masm=intel -fno-asynchronous-unwind-tables -fno-jump-tables -fno-stack-protector -fno-exceptions -O0) и/или за счет ручного редактирования исходного текста ассемблерной программы. **(например удалена метка .name и информация о ОС, на которой была скомпилирована программа)**
- Модифицированная ассемблерная программа отдельно откомпилирована и скомпонована без использования опций отладки. **(директива assembly в Makefile и исполняемый файл assembly)**
- Представлено полное тестовое покрытие, дающее одинаковый результат на обоих программах. Приведены результаты тестовых прогонов для обоих программ, демонстрирующие эквивалентность функционирования. **(есть на первой странице отчета)**

5 баллов

- В реализованной программе использованы функции с передачей данных через параметры. **Например,**

```
lea rdi, -96[rbp] # rdi = &start
mov     esi, 0 # esi = NULL
call    gettimeofday@PLT # gettimeofday(&start, NULL)
```

- Использованы локальные переменные на стеке. **Например,**

```
mov DWORD PTR -116[rbp], 0 # [rbp-116] = optionIndex
mov    DWORD PTR -4[rbp], 1 # [rbp-4] = iterations
```

- В ассемблерную программу при вызове функции добавлены комментарии, описывающие передачу фактических параметров и перенос возвращаемого результата. **Например,**

```
mov rdi, QWORD PTR optarg[rip] # rdi = optarg
lea    rsi, .LC1[rip] # rsi = ":"
call   strtok@PLT # strtok(optarg, ":")
```

- В функциях для формальных параметров добавлены комментарии, описывающие связь между параметрами языка Си и регистрами (стеком). **Например,**

```
lea rdx, -144[rbp] # rdx = &argument
mov    rdi, QWORD PTR -64[rbp] # rdi = file_input
lea    rsi, .LC4[rip] # rsi = "%Lf"
mov    eax, 0 # 0 так как не используем xmm регистры
call   __isoc99_fscanf@PLT # fscanf(file_input, "%Lf", &argument)
```

6 баллов

- Рефакторинг программы на ассемблере за счет оптимизации использования регистров процессора. (файлы main.s и functions.s). **Один из примеров проведенной оптимизации:**
  Было (счетчик в цикле for хранится на стеке):

```
mov DWORD PTR -44[rbp], 0
  jmp    .L8
.L9:
  fld    TBYTE PTR -144[rbp]
  lea    rsp, -16[rsp]
  fstp   TBYTE PTR [rsp]
  call   findAnswer@PLT
  add    rsp, 16
  fstp   TBYTE PTR -32[rbp]
  add    DWORD PTR -44[rbp], 1
.L8:
  mov    eax, DWORD PTR -44[rbp]
  cmp    eax, DWORD PTR -4[rbp]
  jl .L9
```

  Стало (счетчик хранится в регистре r12d):

```
mov r12d, 0 # r12d = i = 0
  jmp    .L8 # прыгаем к условию цикла
.L9: # тело цикла
  fld    TBYTE PTR -144[rbp] # кладем argument в стек x87
  lea    rsp, -16[rsp] # выделяем место для argument на обычном стеке
  fstp   TBYTE PTR [rsp] # кладем argument на стек
  call   findAnswer@PLT # findAnswer(argument);
  add    rsp, 16 # чистим стек
  fstp   TBYTE PTR -32[rbp] # answer = findAnswer(argument);
  add    r12d, 1 # i++
.L8: # условие цикла
  cmp    r12d, DWORD PTR -4[rbp] # сравнием i и iterations
  jl .L9 # если i < iterations то прыгаем в тело цикла
```

- Добавление комментариев в разработанную программу, поясняющих эквивалентное использование регистров вместо переменных исходной программы на С. **(можно увидеть в примере из предыдущего критерия)**

- Представление результатов тестовых прогонов для разработанной программы. Оценка корректности ее выполнения на основе сравнения тестовых прогонов результатами тестирования программы, разработанной на языке С. **(есть на первой странице отчета)**
- Сопоставление размеров программы на ассемблере, полученной после компиляции с языка С с модифицированной программой, использующей регистры. **Таким образом, размер программы полученной после компиляции программы на С 18 936 байт, размер модифицированной программы на ассемблере 18 784 байт.**

7 баллов

- Реализация программы на ассемблере в виде двух или более единиц компиляции (программу на языке С разделять допускается, но не обязательно). **Таким образом, программа на С разделена на файлы main.c и functions.c, аналогично и программа на ассемблере.**
- Использование файлов с исходными данными и файлов для вывода результатов. Имена файлов задаются с использованием аргументов командной строки. Командная строка проверяется на корректность числа аргументов и корректное открытие файлов. **Таким образом, для использования файлов с исходными данными есть ключ -f со следующими аргументами: input_file_name : output_file_name.**
- Подготовка нескольких файлов, обеспечивающих тестовое покрытие разработанной программы. **Это файлы input1, input2.**

8 баллов

- Использование в разрабатываемых программах генератора случайных наборов данных, расширяющих возможности тестирования. **Так, для использования генератора используется ключ -r с единственным аргументом число_повторов. Генератор создает число с плавающей точкой от 0 до 1 и выполняет вычисления над ним заданное число раз.**
- Изменение формата командной строки с учетом выбора ввода из файлов или с использованием генератора. **Так, добавлен соответствующий ключ командной строки.**
- **С помощью функций библиотеки time.h, вычисляется длительность выполнения программы, эти данные можно увидеть на первой странице отчете. Для увеличения времени работы времени программы до 1 секунды используется следующий тест: ./assembly -r 100000. При его выполнении время выполнения программы составило 1.208672 секунд.**
- **Временные данные прогонов программ на С и asm можно увидеть на первой странице отчета.**

9 баллов

- Используя опции оптимизации по скорости, сформированы из программы на С исходный код ассемблере. Провести сравнительный анализ оценки с ассемблерной программой без оптимизации по размеру ассемблерного кода, размеру исполняемого файла и производительности. Сопоставить эти программы с собственной программой, разработанной на ассемблере, в которой вместо переменных максимально возможно используются регистры. **Так, с использованиям опции компиляции -O3 были скомпилированы файлы main_O3.s и functions_O3.s. Суммарный размер этих ассемблерных файлов 1005 строк, размер исполняемого файла 18 992 байт,**

производительность тестируется на тесте ./O3 -f input1:output1. Скорость выполнения программы: 0.000461 секунд. Размер main_original.s и function_original.s (скомпилированы без оптимизаций) 662 строки суммарно, размер исполняемого файла 18 936 байт, производительность на аналогичном тесте 0.001260 секунд. Размер main.s и function.s (вручную отредактированы) 574 строки, размер исполняемого файла 18 784 байт, производительность 0.001186 секунд. Получается, что программа с оптимизациями по скорости имеет больший размер кода и исполняемого файла, но является более производительной относительно остальных.

- Аналогично, используя опции оптимизации по размеру, сформировать код на ассемблере. Провести сравнительный анализ с неоптимизированной программой по размеру ассемблерного кода, размеру исполняемого файла и производительности. Сопоставить эти программы с собственной программой, разработанной на ассемблере, в которой вместо переменных максимально возможно используются регистры. **Так, с помощью опции компиляции -Os были получены файлы с ассемблерным кодом main_Os.s и functions_Os.s и исполняемых файл Os. Размер файлов main_Os.s и functions_Os.s 644 строки, размер исполняемого файла 18 952 байт, производительность на аналогичном тесте 0.000614 секунд. Данные о других программах приведены в предыдущем критерии. Получается, что оптимизация по размеру с помощью опции компилятора, проигрывает ручному редактированию по размеру исполняемого файла и размеру ассемблерного кода, но выигрывает по производительности, при этом превосходит по всем параметрам, кроме размера исполняемого файла, программу скомпилированную без оптимизаций.**