­­­­**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
 РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет информационных технологий**

**Кафедра параллельных вычислений**

**ОТЧЕТ**

**О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ**

«ВВЕДЕНИЕ В АРХИТЕКТУРУ ARM»

студента 2 курса, группы 23201

Смирнова Гордея Андреевича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Матвеев А.С.

Новосибирск 2024

**Цель**

Изучить аспекты работы с языком ассемблера; Ознакомиться с архитектурой ARM; Провести детальный анализ ассемблерного кода программы первой лабораторной работы.

**Задание**

Изучить программную архитектуру ARM(Advanced RISC Machine). Ознакомиться с набором регистров процессора, основными отличиями ассемблерного кода архитектуры x86/x86-64 и ARM. Изучить способы адресации памяти, работу со стеком на ARM. Для программы на языке С сгенерировать ассемблерный листинг, после чего провести анализ,сопоставляя команды языка С с машинными командами. После детального анализа процессорных инструкций продемонстрировать использование ключевых особенностей архитектур ARM на конкретных участках кода.

**Листинг программы**

#include <stdio.h>

#define N\_CONST 1000000000

double PiLeibniz(long long const n) {

double result = 0;

for (long long i = 0; i < n; ++i) {

double const t = 1.0 / (2.0 \* (double)i + 1.0);

result = i % 2 == 0 ? result + t : result - t;

}

return result \* 4;

}

int main(void) {

double const result = PiLeibniz(N\_CONST);

printf("pi = %.8lf\n", result);

return 0;

}

**Ассемблерный листинг для архитектуры ARM без оптимизаций**

PiLeibniz:

sub sp, sp, #48 ; выделение места на стеке под переменные

str x0, [sp, 8] ; n [sp+8] = x0

str xzr, [sp, 40] ; result [sp+40] = 0

str xzr, [sp, 32] ; i [sp+32] = 0

b .L2 ; переход к L2

.L5:

ldr d31, [sp, 32] ; d31 = i [sp+32]

scvtf d31, d31 ; d31 = (double)i

fadd d30, d31, d31 ; d30 = 2\*d31

fmov d31, 1.0e+0 ; d31 = 1.0

fadd d31, d30, d31 ; d31 += d30

fmov d30, 1.0e+0 ; d30 = 1.0

fdiv d31, d30, d31 ; d31 = 1.0 [d30] / d31

str d31, [sp, 24] ; t [sp+24] = d31

ldr x0, [sp, 32] ; x0 = i [sp+32]

and x0, x0, 1 ; взятие последнего бита x0

cmp x0, 0 ; сравнение его с 0

bne .L3 ; если != то переход к L3 (i нечет)

ldr d30, [sp, 40] ; d30 = result [sp+40]

ldr d31, [sp, 24] ; d31 = t [sp+24]

fadd d31, d30, d31 ; d31 = d30 + d31

b .L4 ; переход к L4

.L3:

ldr d30, [sp, 40] ; d30 = result [sp+40]

ldr d31, [sp, 24] ; d31 = t [sp+24]

fsub d31, d30, d31 ; d31 = d30 - d31

.L4:

str d31, [sp, 40] ; result [sp+40] = d31

ldr x0, [sp, 32] ; x0 = i [sp+32]

add x0, x0, 1 ; x0 += 1

str x0, [sp, 32] ; i [sp+32] = x0

.L2:

ldr x1, [sp, 32] ; x1 = i [sp+32]

ldr x0, [sp, 8] ; x0 = n [sp+8]

cmp x1, x0 ; сравнение x1, x0

blt .L5 ; если < то переход к L5

ldr d30, [sp, 40] ; d30 = result [sp+40]

fmov d31, 4.0e+0 ; d31 = 4.0

fmul d31, d30, d31 ; d31 \*= d30

fmov d0, d31 ; d0 = d31

add sp, sp, 48 ; восстановление указателя стека

ret

.LC0:

.string "pi = %.8lf\n"

main:

stp x29, x30, [sp, -32]! ; сохранение frame pointer и link register, уменьшение указателя стека на 32

mov x29, sp ; x29 = текущий указатель стека

mov x0, 51712 ; x0 = 51712

movk x0, 0x3b9a, lsl 16 ; x0: установка 15258 со сдвигом 16бит, получилось n = 15258\*2^16 + 51712 = 10^9

bl PiLeibniz ; вызов функции

str d0, [sp, 24] ; result [sp+24] = d0

ldr d0, [sp, 24] ; d0 = result [sp+24]

adrp x0, .LC0 ; x0 = адрес константной строки

add x0, x0, :lo12:.LC0 ; x0 += младшие 12бит адреса

bl printf ; вызов printf

mov w0, 0 ; w0 = 0 для return

ldp x29, x30, [sp], 32 ; восстановление FR и LR из стека, увеличение указателя стека на 32

ret

**Ассемблерный листинг для архитектуры ARM с оптимизацией O1**

PiLeibniz:

cmp x0, 0 ; сравнение n [x0], 0

ble .L6 ; если <= то переход к L6

mov x1, 0 ; i [x1] = 0

movi d30, #0 ; result [d30] = 0.0

fmov d29, 1.0e+0 ; d29 = 1.0

b .L5 ; переход к L5

.L3:

fsub d30, d30, d31 ; result [d30] -= t [d31]

.L4:

add x1, x1, 1 ; i [x1] += 1

cmp x0, x1 ; сравнение n [x0], i [x1]

beq .L2 ; если == то переход к L2

.L5:

scvtf d31, x1 ; d31 = (double)i [x1]

fadd d31, d31, d31 ; d31 += d31

fadd d31, d31, d29 ; d31 += 1.0 [d29]

fdiv d31, d29, d31 ; t [d31] = 1.0 [d29] / d31

tbnz x1, 0, .L3 ; последний бит x1: если !=0 то переход к L3 (i нечет)

fadd d30, d30, d31 ; result [d30] += t [d31]

b .L4 ; переход к L4

.L6:

movi d30, #0 ; result [d30] = 0

.L2:

fmov d0, 4.0e+0 ; d0 = 4.0

fmul d0, d30, d0 ; d0 \*= result [d30]

ret

.LC0:

.string "pi = %.8lf\n"

main:

stp x29, x30, [sp, -16]! ; то же самое, но уменьшение указателя стека на 16

mov x29, sp

mov x0, 51712

movk x0, 0x3b9a, lsl 16

bl PiLeibniz

adrp x0, .LC0

add x0, x0, :lo12:.LC0

bl printf

mov w0, 0

ldp x29, x30, [sp], 16

ret

**Ассемблерный листинг для архитектуры ARM с оптимизацией O2**

PiLeibniz:

cmp x0, 0 ; сравнение n [x0], 0

ble .L6 ; если <= то переход к L6

movi d29, #0 ; result [d29] = 0

mov x1, 0 ; i [x1] = 0

fmov d30, 2.0e+0 ; d30 = 2.0

fmov d31, 1.0e+0 ; d31 = 1.0

b .L5 ; переход к L5

.L10:

add x1, x1, 1 ; i [x1] += 1

fadd d29, d29, d1 ; result

cmp x0, x1 ; сравнение n [x0], i [x1]

beq .L9 ; если == то переход к L9

.L5:

scvtf d1, x1 ; d1 = (double)i [x1]

fmadd d1, d1, d30, d31 ; d1 = d1 \* 2.0 [d30] + 1.0 [d31]

fdiv d1, d31, d1 ; t [d1] = 1.0 [d31] / d1

tbz x1, 0, .L10 ; последний бит x1: если ==0 то переход к L10 (i чет)

add x1, x1, 1 ; i [x1] += 1

fsub d29, d29, d1 ; result [d29] -= t [d1]

cmp x0, x1 ; сравнение n [x0], i [x1]

bne .L5 ; если != то переход к L5

.L9:

fmov d28, 4.0e+0 ; d28 = 4.0

fmul d0, d29, d28 ; d0 = result [d29] \* 4.0 [d28]

ret

.L6:

movi d0, #0 ; d0 = 0

ret

.LC0:

.string "pi = %.8lf\n"

main:

stp x29, x30, [sp, -16]!

mov x0, 51712

movk x0, 0x3b9a, lsl 16

mov x29, sp

bl PiLeibniz

adrp x0, .LC0

add x0, x0, :lo12:.LC0

bl printf

mov w0, 0

ldp x29, x30, [sp], 16

ret

**Ассемблерный листинг для архитектуры ARM с оптимизацией O3, Ofast**

PiLeibniz:

cmp x0, 0

ble .L6

movi d29, #0

mov x1, 0

fmov d30, 2.0e+0

fmov d31, 1.0e+0

b .L5

.L10:

add x1, x1, 1

fadd d29, d29, d1

cmp x0, x1

beq .L9

.L5:

scvtf d1, x1

fmadd d1, d1, d30, d31

fdiv d1, d31, d1

tbz x1, 0, .L10

add x1, x1, 1

fsub d29, d29, d1

cmp x0, x1

bne .L5

.L9:

fmov d28, 4.0e+0

fmul d0, d29, d28

ret

.L6:

movi d0, #0

ret

.LC0:

.string "pi = %.8lf\n"

main:

movi d30, #0 ; команды из О2 но с инлайном в main

mov x1, 51712

stp x29, x30, [sp, -16]!

mov x0, 0

fmov d28, 2.0e+0

fmov d29, 1.0e+0

movk x1, 0x3b9a, lsl 16

mov x29, sp

.L15:

scvtf d31, x0

fmadd d31, d31, d28, d29

fdiv d31, d29, d31

tbnz x0, 0, .L12

.L17:

add x0, x0, 1

fadd d30, d30, d31

scvtf d31, x0

fmadd d31, d31, d28, d29

fdiv d31, d29, d31

tbz x0, 0, .L17

.L12:

add x0, x0, 1

fsub d30, d30, d31

cmp x0, x1

bne .L15

fmov d0, 4.0e+0

adrp x0, .LC0

add x0, x0, :lo12:.LC0

fmul d0, d30, d0

bl printf

mov w0, 0

ldp x29, x30, [sp], 16

ret

**Выводы**

По результатам проведенного анализа было установлено, что в ассемблерном коде ARM обращение к памяти, работа с математическими функциями, работа с регистрами существенно отличаются от ассемблерного кода на x86/x86-64. Изучены основные отличия этих архитектур и изучены основы ARM.