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 ПРАКТИЧЕСКОЙ РАБОТЫ

«Знакомство с программной архитектурой ARM и анализ ассемблерного листинга»

студента 2 курса, группы 20203

Синюкова Валерия Константиновича

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
доцент кафедры параллельных вычислений
Власенко Андрей Юрьевич

Новосибирск 2021

СОДЕРЖАНИЕ

ЦЕЛЬ

1. Знакомство с программной архитектурой ARM.
2. Анализ ассемблерного листинга программы для архитектуры ARM.

ЗАДАНИЕ

1. Изучить основы программной архитектуры ARM.
2. Для программы на языке С (вычисление числа π с помощью разложения в ряд Лейбница) сгенерировать ассемблерные листинги для архитектуры ARM, используя различные уровни комплексной оптимизации.
3. Проанализировать полученные листинги и сделать следующее:
   - Сопоставьте команды языка Си с машинными командами.
   - Определить размещение переменных языка Си в программах на ассемблере (в каких регистрах, в каких ячейках памяти).
   - Описать и объяснить оптимизационные преобразования, выполненные компилятором.
   - Продемонстрировать использование ключевых особенностей архитектуры ARM на конкретных участках ассемблерного кода.
4. Составить отчет по лабораторной работе. Отчет должен содержать следующее.
   - Титульный лист.
   - Цель лабораторной работы.
   - Полный компилируемый листинг реализованной программы и команды для ее компиляции.
   - Листинг на ассемблере с описаниями назначения команд с точки зрения реализации алгоритма выбранного варианта.
   - Вывод по результатам лабораторной работы.

ОПИСАНИЕ РАБОТЫ

1) Были сгенерированы ассемблерные листинги для программы на языке С «вычисление числа Пи с помощью разложения в ряд Грегори-Лейбница» для программной архитектуры ARM (компилятор gcc 11.1) с уровнями оптимизации O0 и O3.
2) Были сопоставлены команды программы на языке С с командами получившихся ассемблерных листингов. Далее приведен полный компилируемый листинг программы на С и ассемблерный листинг программы с уровнем компиляции O0 с комментариями, каким командам на языке С соответствуют различные секции ассемблерного кода.

Код на языке С:

```
#include <stdio.h>
#include <string.h>

double LeibnizFormula (long long N)
{
    double Pi = 0, nextMember;
    long long i;
    for (i = 0; i < N; i++)
    {
    if (i % 2)
        nextMember = -1;
    else
        nextMember = 1;
    nextMember /= 2*(double)(i) + 1;
    Pi += nextMember;
    }
    Pi *= 4;
    return Pi;
}

int fromCharToInt (char c)
{
    return (int)(c - '0');
}

long long getN (char * charN)
{
    int i, lengthN = strlen(charN);
    long long N = 0;
    for (i = 0; i < lengthN; i++)
    {
    N *= 10;
    N += (long long)(fromCharToInt(charN[i]));
    }
    return N;
}
```

```
void printAnswer(double Pi)
{
    printf("%f\n",Pi);
}

int main(int argc, char** argv)
{
    long long N = getN(argv[1]);
    double Pi = LeibnizFormula(N);
    printAnswer(Pi);
    return 0;
}
```

Ассемблерный код с уровнем оптимизации O0:

```
LeibnizFormula:
        push    {r4, r5, r7, r8, r9, lr}
        sub     sp, sp, #32
        add     r7, sp, #0
        strd    r0, [r7]
        mov     r2, #0
        mov     r3, #0
        strd    r2, [r7, #24]                   // *(r7 + 24) = Pi
        vmov.i32        d16, #0  @ di
        vstr.64 d16, [r7, #8]     @ int         // *(r7 + 8) = i
        b       .L2
.L5:
        ldrd    r2, [r7, #8]
        and     r4, r2, #1
        movs    r5, #0
        orrs    r3, r4, r5
        beq     .L3
        mov     r2, #0
        mov     r3, #0
        movt    r3, 49136
        strd    r2, [r7, #16]                   // *(r7 + 16) = -1;
        b       .L4
.L3:
        mov     r2, #0
        mov     r3, #0
        movt    r3, 16368
        strd    r2, [r7, #16]                   // *(r7 + 16) = 1;
.L4:
        ldrd    r0, [r7, #8]                    /*
        bl      __aeabi_l2d
        vmov    d16, r0, r1
        vadd.f64        d16, d16, d16
        vmov.f64        d17, #1.0e+0
        vadd.f64        d17, d16, d17
        vldr.64 d18, [r7, #16]                  nextMember += 1/(2*i + 1);
        vdiv.f64        d16, d18, d17
```

```
        vstr.64 d16, [r7, #16]                  *(r7 + 16) = nextMember
        vldr.64 d17, [r7, #24]
        vldr.64 d16, [r7, #16]                  */
        vadd.f64        d16, d17, d16           /* Pi += nextMember;
        vstr.64 d16, [r7, #24]                  */ *(r7 + 24) = Pi
        ldrd    r2, [r7, #8]                    /*
        adds    r8, r2, #1
        adc     r9, r3, #0                            i++;
        strd    r8, [r7, #8]                    */
.L2:
        ldrd    r0, [r7, #8]                    /*
        ldrd    r2, [r7]
        cmp     r0, r2                          Если i < N, то переход на след.
        sbcs    r3, r1, r3                      итерацию цикла for
        blt     .L5                             */
        vldr.64 d16, [r7, #24]
        vmov.f64        d17, #4.0e+0            /*
        vmul.f64        d16, d16, d17                 Pi *= 4;
        vstr.64 d16, [r7, #24]                  */
        ldrd    r2, [r7, #24]
        vmov    d16, r2, r3
        vmov.f64        d0, d16
        adds    r7, r7, #32
        mov     sp, r7
        pop     {r4, r5, r7, r8, r9, pc}
fromCharToInt:
        push    {r7}
        sub     sp, sp, #12
        add     r7, sp, #0
        mov     r3, r0
        strb    r3, [r7, #7]                    // *(r7 + 7) = c
        ldrb    r3, [r7, #7]    @ zero_extendqisi2
        subs    r3, r3, #48                     // c - 48
        mov     r0, r3
        adds    r7, r7, #12
        mov     sp, r7
        ldr     r7, [sp], #4
        bx      lr
getN:
        push    {r4, r5, r7, r8, r9, r10, fp, lr}
        sub     sp, sp, #56
        add     r7, sp, #0
        str     r0, [r7, #28]                   // *(r7 + 28) = charN
        ldr     r0, [r7, #28]
        bl      strlen
        mov     r3, r0
        str     r3, [r7, #36]                   // *(r7 + 36) = r3 = lengthN
        vmov.i32        d16, #0  @ di
        vstr.64 d16, [r7, #40]    @ int         // *(r7 + 40) = N
        movs    r3, #0
        str     r3, [r7, #52]                   // *(r7 + 52) = i
```

```
        b       .L10
.L11:
        ldrd    r0, [r7, #40]                   /*
        mov     r2, r0
        mov     r3, r1
        adds    r8, r2, r2
        adc     r9, r3, r3
        adds    r3, r8, r8
        str     r3, [r7, #16]
        adc     r3, r9, r9
        str     r3, [r7, #20]
        ldrd    r2, [r7, #16]                       N *= 10
        adds    r4, r2, r0
        adc     r5, r3, r1
        adds    r3, r4, r4
        str     r3, [r7, #8]
        adc     r3, r5, r5
        str     r3, [r7, #12]
        ldrd    r4, [r7, #8]
        strd    r4, [r7, #40]                   */
        ldr     r3, [r7, #52]
        ldr     r2, [r7, #28]
        add     r3, r3, r2
        ldrb    r3, [r3]        @ zero_extendqisi2
        mov     r0, r3
        bl      fromCharToInt                   /*
        mov     r3, r0
        asrs    r2, r3, #31
        mov     r10, r3
        mov     fp, r2
        ldrd    r2, [r7, #40]
        adds    r1, r2, r10                     N += fromCharToInt(charN[i]);
        str     r1, [r7]
        adc     r3, r3, fp
        str     r3, [r7, #4]
        ldrd    r2, [r7]
        strd    r2, [r7, #40]                   */
        ldr     r3, [r7, #52]                   /*
        adds    r3, r3, #1                          i++;
        str     r3, [r7, #52]                   */
.L10:
        ldr     r2, [r7, #52]
        ldr     r3, [r7, #36]
        cmp     r2, r3
        blt     .L11
        ldrd    r2, [r7, #40]
        mov     r0, r2
        mov     r1, r3
        adds    r7, r7, #56
        mov     sp, r7
        pop     {r4, r5, r7, r8, r9, r10, fp, pc}
```

```
.LC0:
        .ascii  "%f\012\000"
printAnswer:
        push    {r7, lr}
        sub     sp, sp, #8
        add     r7, sp, #0
        vstr.64 d0, [r7]
        ldrd    r2, [r7]
        movw    r0, #:lower16:.LC0
        movt    r0, #:upper16:.LC0
        bl      printf
        nop
        adds    r7, r7, #8
        mov     sp, r7
        pop     {r7, pc}
main:
        push    {r7, lr}
        sub     sp, sp, #24
        add     r7, sp, #0
        str     r0, [r7, #4]
        str     r1, [r7]
        ldr     r3, [r7]
        adds    r3, r3, #4
        ldr     r3, [r3]
        mov     r0, r3                          // r0 = argv[1]
        bl      getN
        strd    r0, [r7, #16]                   // *(r7 + 16) = N
        ldrd    r0, [r7, #16]
        bl      LeibnizFormula
        vstr.64 d0, [r7, #8]
        vldr.64 d0, [r7, #8]
        bl      printAnswer
        movs    r3, #0
        mov     r0, r3
        adds    r7, r7, #24
        mov     sp, r7
        pop     {r7, pc}
```

3) Далее приведен ассемблерный листинг, сгенерированный при уровне оптимизации O3:

```
LeibnizFormula:
        push    {r3, r4, r5, r6, r7, lr}
        cmp     r0, #1
        sbcs    r3, r1, #0
        vpush.64        {d8, d9, d10}
        blt     .L7
        subs    r3, r0, #1
        mov     r6, r0
        mov     r7, r1
        orrs    r3, r3, r1
        vmov.f64        d10, #1.0e+0
```

```
        beq     .L3
        vmov.f64        d8, d10
        movs    r4, #1
        movs    r5, #0
        vmov.f64        d9, #-1.0e+0
.L6:
        mov     r0, r4
        mov     r1, r5
        lsls    r3, r4, #31
        bpl     .L11
        bl      __aeabi_l2d
        vmov    d16, r0, r1
        adds    r4, r4, #1
        vadd.f64        d16, d16, d16
        adc     r5, r5, #0
        cmp     r7, r5
        it      eq
        cmpeq   r6, r4
        vadd.f64        d16, d16, d8
        vdiv.f64        d17, d9, d16
        vadd.f64        d10, d10, d17
        bne     .L6
.L3:
        vmov.f64        d0, #4.0e+0
        vmul.f64        d0, d10, d0
        vldm    sp!, {d8-d10}
        pop     {r3, r4, r5, r6, r7, pc}
.L11:
        bl      __aeabi_l2d
        vmov    d16, r0, r1
        adds    r4, r4, #1
        vadd.f64        d16, d16, d16
        adc     r5, r5, #0
        cmp     r5, r7
        it      eq
        cmpeq   r4, r6
        vadd.f64        d16, d16, d8
        vdiv.f64        d17, d8, d16
        vadd.f64        d10, d10, d17
        bne     .L6
        b       .L3
.L7:
        vldm    sp!, {d8-d10}
        vmov.i64        d0, #0  @ float
        pop     {r3, r4, r5, r6, r7, pc}
fromCharToInt:
        subs    r0, r0, #48
        bx      lr
getN:
        push    {r3, r4, r5, lr}
        mov     r5, r0
```

```
        bl      strlen
        mov     r4, r0
        cbz     r0, .L16
        add     lr, r5, #-1
        movs    r0, #0
        add     r4, r4, lr
        mov     r1, r0
.L15:
        adds    r3, r0, r0
        ldrb    r2, [lr, #1]!   @ zero_extendqisi2
        adc     ip, r1, r1
        adds    r3, r3, r3
        adc     ip, ip, ip
        adds    r3, r3, r0
        adc     r1, r1, ip
        subs    r2, r2, #48
        adds    r3, r3, r3
        adcs    r1, r1, r1
        adds    r0, r2, r3
        adc     r1, r1, r2, asr #31
        cmp     lr, r4
        bne     .L15
        pop     {r3, r4, r5, pc}
.L16:
        mov     r1, r0
        pop     {r3, r4, r5, pc}
.LC0:
        .ascii  "%f\012\000"
printAnswer:
        movw    r0, #:lower16:.LC0
        movt    r0, #:upper16:.LC0
        vmov    r2, r3, d0
        b       printf
main:
        push    {r3, r4, r5, r6, r7, lr}
        ldr     r5, [r1, #4]                    // *(r1 + 4) = argv[1];
        vpush.64        {d8, d9, d10}
        mov     r0, r5
        bl      strlen
        cmp     r0, #0
        beq     .L28
        subs    r5, r5, #1
        movs    r4, #0
        add     r0, r0, r5
        mov     r6, r4
.L22:
        adds    r3, r4, r4
        ldrb    r1, [r5, #1]!   @ zero_extendqisi2
        adc     r2, r6, r6
        adds    r3, r3, r3
        adcs    r2, r2, r2
```

```
        adds    r3, r3, r4
        adc     r2, r6, r2
        subs    r1, r1, #48
        adds    r3, r3, r3
        adcs    r2, r2, r2
        adds    r4, r1, r3
        adc     r6, r2, r1, asr #31
        cmp     r0, r5
        bne     .L22
        cmp     r4, #1
        sbcs    r3, r6, #0
        blt     .L28
        subs    r3, r4, #1
        vmov.f64        d10, #1.0e+0
        orrs    r3, r3, r6
        beq     .L21
        vmov.f64        d8, d10
        movs    r5, #1
        movs    r7, #0
        vmov.f64        d9, #-1.0e+0
.L26:
        mov     r0, r5
        mov     r1, r7
        lsls    r3, r5, #31
        bpl     .L32
        bl      __aeabi_l2d
        vmov    d16, r0, r1
        adds    r5, r5, #1
        vadd.f64        d16, d16, d16
        adc     r7, r7, #0
        cmp     r6, r7
        it      eq
        cmpeq   r4, r5
        vadd.f64        d16, d16, d8
        vdiv.f64        d17, d9, d16
        vadd.f64        d10, d10, d17
        bne     .L26
.L21:
        vmov.f64        d16, #4.0e+0
        movw    r0, #:lower16:.LC0
        movt    r0, #:upper16:.LC0
        vmul.f64        d16, d10, d16
        vmov    r2, r3, d16
        bl      printf
        movs    r0, #0
        vldm    sp!, {d8-d10}
        pop     {r3, r4, r5, r6, r7, pc}
.L32:
        bl      __aeabi_l2d
        vmov    d16, r0, r1
        adds    r5, r5, #1
```

```
        vadd.f64        d16, d16, d16
        adc     r7, r7, #0
        cmp     r7, r6
        it      eq
        cmpeq   r5, r4
        vadd.f64        d16, d16, d8
        vdiv.f64        d17, d8, d16
        vadd.f64        d10, d10, d17
        bne     .L26
        b       .L21
.L28:
        vmov.i64        d10, #0 @ float
        b       .L21
```

Выделим оптимизационные преобразования, выполненные компилятором:

- Было произведено встраивание всех функций в функцию main, таким образом, при выполнении программы не тратится время на вызов функций и создание локальных копий переменных.
- Программа обращается к оперативной памяти лишь единожды в начале функции main для того, чтобы получить доступ к argv[1] (argv[1] – параметр, вводимый пользователем, - количество членов ряда, участвующих в разложении), в остальном коде переменные хранятся в регистрах процессора.
- В случае, когда длина argv[1] равна 0, либо N = 0 (N – количество членов ряда, участвующих в разложении), программа сразу переходит к печати результата и завершению работы.

4) Выделим использование ключевых особенностей архитектуры ARM в нашем коде:

- Архитектура ARM позволяет записывать результат арифметической операции над двумя регистрами в третий регистр, это используется, например, в следующих фрагментах кода:

  ```
  adds    r1, r2, r10
  ```
  (метка .L11 первого листинга)

  ```
  vdiv.f64        d16, d18, d17
  ```
  (метка .L4 первого листинга)

  Данные фрагменты выделены в коде желтым цветом.

- Все операции преобразования данных производятся только с данными, находящимися в регистрах.
- Отсутствуют сложные команды по типу команды lea в архитектуре x86-64, которые выполняют несколько арифметических операций за раз.
- Для перехода по некоторым меткам используется команда bl (в листингах соответствующие фрагменты выделены бирюзовым цветом), которая сохраняет адрес возврата, таким образом, в конце вызываемой функции не нужно использовать команду ret.

- В некоторым местах к мнемонике команд добавляется суффикс «s», из-за чего данная команда может устанавливать флаги по результатам вычислений (примеры данного явления выделены в первом листинге зеленым цветом)
- Архитектура ARM позволяет выполнять не только условный переход на другой адрес программы, но и условное выполнение отдельных команд при помощи использования различных суффиксов. Это позволяет минимизировать количество переходов и, следовательно, эффективнее использовать конвейер процессора (соответствующие фрагменты выделены во втором листинге серым цветом).

ЗАКЛЮЧЕНИЕ

В результате данной практической работы была изучена программная архитектура ARM, были сгенерированы и проанализированы ассемблерные листинги программы на языке C с уровнями оптимизации O0 и O3. Ассемблерный код программы был сопоставлен с кодом на языке C. Были выделены оптимизационные преобразования, выполненные компилятором, а также использование ключевых особенностей архитектуры ARM.