Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Архитектура вычислительных систем

ОТЧЁТ

к практической работе

на тему

ПРОГРАММИРОВАНИЕ АРИФМЕТИЧЕСКОГО СОПРОЦЕССОРА

Выполнил: студент группы 153501
Тимофеев Кирилл Андреевич

Проверила:
Калиновская Анастасия Александровна

МИНСК 2023

СОДЕРЖАНИЕ

КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Сопроцессорные конфигурации

Использование сопроцессора позволяет значительно ускорить работу программ, выполняющих расчеты с высокой точностью, тригонометрические вычисления и обработку информации, которая должна быть представлена в виде действительных чисел. Сопроцессор подключается к системной шине параллельно с центральным процессором (CPU) и может работать только совместно с ним. Все команды попадают в оба процессора, а выполняет каждый свои. Сопроцессор не имеет своей программы и не может осуществлять выборку команд и данных. Это делает центральный процессор. Сопроцессор перехватывает с шины данные и после этого реализует конкретные действия по выполнению команды. Два процессора работают параллельно, что повышает эффективность системы. Но возникают ситуации, когда их работа требует синхронизации (из-за разницы во времени выполнения команд).

Синхронизация по командам. Когда центральный процессор выбирает для выполнения команду FPU, последний может быть занят выполнением предыдущей команды. Поэтому перед каждой командой сопроцессора в программе должна стоять специальная команда (wait), которая только проверяет текущее состояние FPU и, если он занят, переводит центральный процессор в состояние ожидания. Соответствующую команду в программу может вводить либо ассемблер, либо компилятор языка без указаний программиста.

Синхронизация по данным. Если выполняемая в FPU команда записывает операнд в память перед последующей командой CPU, которая обращается к этой ячейке памяти, требуется команда проверки состояния FPU. Если данные еще не были записаны, CPU должен переходить в состояние ожидания. Автоматически учесть такие ситуации довольно сложно, поэтому вводить команды, которые проверяют состояние сопроцессора и при необходимости заставляют центральный процессор ожидать, должен программист.

Программная модель сопроцессора

В программную модель любого процессора включаются только те регистры, которые доступны программисту на уровне машинных команд. Основу программной модели FPU образует регистровый стек из восьми 80-битных регистров R0-R7. В них хранятся числа в вещественном формате. В любой момент времени 3-битное поле ST в слове состояния определяет регистр, являющийся текущей вершиной стека и обозначаемый ST (0). При занесении в стек (push) осуществляется декремент поля ST и загружаются данные в новую вершину стека. При извлечении из стека (pop) в получатель, которым чаще всего является память, передается содержимое вершины стека, а затем инкрементируется поле ST.

В организации регистрового стека FPU есть отличия от классического стека.

1. Стек имеет кольцевую структуру. Контроль за использованием стека должен осуществлять программист. Максимальное число занесений без про-

межуточных извлечений равно 8.

2. В командах FPU допускается явное или неявное обращение к регистрам с модификацией или без поля ST. Например, команда fsqrt замещает число из вершины стека значением корня из него. В бинарных операциях допускается явное указание регистров. Адресация осуществляется относительно текущей вершины стека и обозначение ST (i) 0<i<7, считая от вершины.

3. Не все стековые команды автоматически модифицируют указатель вершины стека.

С каждым регистром стека ассоциируется 2-битный тег (признак), совокупность которых образует слово тегов. Тег регистра R0 находиться в младших битах, R7 – в старших. Тег фиксирует наличие в регистре действительного числа (код 00), истинного нуля (код 01), ненормализованного или бесконечности (код 10) и отсутствие данных (код 11). Наличие тегов позволяет FPU быстрее обнаруживать особые случаи (попытка загрузить в непустой регистр, попытка извлечь из пустого) и обрабатывать данные.

Остальными регистрами FPU являются регистр управления, регистр состояния, два регистра состояния команды и два регистра указателя данных. Длина их всех 16 бит.

Форматы численных данных

Арифметический FPU К1810ВМ87 оперирует с семью форматами численных данных, образующих три класса: двоичные целые, упакованные десятичные целые и вещественные числа. Во всех форматах старший (левый) бит отведен для знака.

Форматы различаются длиной, следовательно, диапазоном допустимых чисел, способом представления (упакованный и неупакованный формат), способом кодировки (прямой и дополнительный код).

Двоичные целые числа. Три формата целых двоичных (целое слово (16 бит), короткое целое (32 бита), длинное целое (64 бита)) отличаются длиной, следовательно, диапазоном чисел. Только в этих форматах применяется стандартный дополнительный код. 0 имеет единственное кодирование. Наибольшее положительное число кодируеться как 011…1, а наибольшее по модулю отрицательное как 100..0.

Упакованные десятичные целые. Числа представлены в прямом коде и упакованном формате, т.е. в байте содержится две десятичные цифры в коде 8421. Старший бит левого байта – знак, остальные игнорируются, но при записи в них помещаются нули. Но надо учитывать, что при наличии в тетраде запрещающих комбинаций 1010 – 1111 результат операции не определен. Т.е. сопроцессор не контролирует правильность результата.

Вещественные числа. Различают короткие вещественные (КВ)(мантиса – 24 бита, порядок – 8 бит), длинные вещественные (ДВ) (мантиса – 53 бита, порядок – 11 бит) и временные вещественные (ВВ) (мантиса – 64 бита, порядок – 15 бит). Для них применяется формат с плавающей точкой. Значащие цифры находятся в поле мантисы, порядок показывает факти-

ческое положение двоичной точки в разрядах мантисы, бит знака S определяет знак числа. Порядок дается в смещенной форме.

Режимы работы. Состояние

Сопроцессор имеет 2 доступных 16-битных регистра, содержимое которых определяет его режим работы и текущее состояние. Форматы регистров содержат слово управления CW и слово состояния SW. Регистр управления содержит 6 бит масок особых случаев. Регистр состояния – 6 бит флажков.

Регистр управления содержит 6 бит масок особых случаев, а регистр состояния 6 бит флажков особых случаев:

P – потеря точности

U – антипереполнение

O – переполнение

Z - деление на нуль

D - денормализованный операнд

I - недействительный операция

Слово управления. Оно определяет для FPU один из нескольких вариантов обработки численных данных. Программа центрального процессора может сформировать в памяти образ слова управления, а затем заставить сопроцессор загрузить его в регистр CW. Рассмотрим значение полей.

Шесть младших бит слова управления - индивидуальные маски особых случаев. Т.е. особых ситуаций, обнаруженных FPU при выполнении команд. Если бит=1, то не будет вызвано прерывание CPU. Иначе FPU устанавливает в 1 бит запроса прерывания в слове состояния и при общем разрешении прерываний генерирует сигнал int прерывания CPU.

Бит 7 слова управления содержит маску управлени прерыванием IEM, которая разрешает (IEM=0) или запрещает (IEM=1) прерывание центрального процессора.

Двухбитное поле управления точностью (PC) определяет точность вычислений в 24 бита (PC=00), 53 бита (PC=10) или 64 бита (PC=11). По умолчанию вводиться режим с максимальной точностью в 64 бита.

Двухбитное поле управления округлением RC определяет один из четырех возможных вариантов округления результатов операций сопроцессора.

Бит 12 управляет режимом бесконечности IC. Когда IC=0, сопрцессор обрабатывает два специальных значения “плюс бесконечность” и “минус бесконечность” как одно и то же значение “бесконечность”, не имеющее знака.

Слово состояния. В нем младшие 6 бит отведены для регистрации особых случаев. Бит 7 – запроса прерывания (IR), устанавливается в 1 при возникновении любого незамаскированного особого случая. Бит C3-C0 фиксирует код условия в операциях сравнения, проверки условия и анализа. Три бита ST указатели стека. Стековые операции сопровождаются модификацией поля ST. Наконец, флажок занятости B устанавливается в состояние 1 когда численное операционные устройство выполняет операцию. Большую роль играют биты кода условия, которые фиксируют особенности результата

(табл. 1.). Коды условия привлекаются для реализации условных переходов. Сопроцессор самостоятельно не может влиять на ход выполнения программ. Поэтому для условных переходов по результатам операций сопроцессора приходится сначала передавать код условия в память, а затем загружать один из регистров центрального процессора. После этого код условия передается в регистр флагов, производится условный переход.

ЗАДАНИЯ И ВЫВОД ПРОГРАММЫ

Задание к лабораторной работе 7

Написать программу, находящую решение квадратного уравнения

$ax^2 + bx + c = 0$

с помощью сопроцессора.

Задание к лабораторной работе 8

$$S(x) = \sum_{k=1}^{n} \frac{\cos(2kx)}{4k^2 - 1}, \qquad Y(x) = \frac{1}{2} - \frac{\pi}{4}\left|\sin(x)\right|.$$

Значение аргумента **x** изменяется от ***a*** до ***b*** с шагом ***h***. Для каждого ***x*** найти значения функции Y(x), суммы S(x) и число итераций n, при котором достигается требуемая точность ε = |Y(x)-S(x)|. Результат вывести в виде таблицы. Значения ***a***, ***b***, ***h*** и ***ε*** вводятся с клавиатуры.

Вывод программы

```
Введите a:1
В двоичном коде: 1.0
Введите b:6
В двоичном коде: 110.0
Введите c:5
В двоичном коде: 101.0
x1 = -1.0
x2 = -5.0
```

Рисунок 1 – Лабораторная работа №7

```
Введите начальное значение аргумента x: 1
Введите конечное значение аргумента x: 10
Введите шаг изменения аргумента x: 1
Введите требуемую точность ε: 0.001
x          Y(x)       S(x)       n
1.000      -0.161     -0.161     9
2.000      -0.214     -0.215     5
3.000      0.389      0.389      12
4.000      -0.094     -0.094     5
5.000      -0.253     -0.252     2
6.000      0.281      0.281      1
7.000      -0.016     -0.016     4
8.000      -0.277     -0.277     5
9.000      0.176      0.175      7
10.000     0.073      0.072      8
```

Рисунок 2 – Лабораторная работа №8

ПРОГРАММНЫЙ КОД

Лабораторная работа №7

```
import math
from math import copysign, fabs, floor, isfinite, modf

def float_to_bin_fixed(f):
    if not isfinite(f):
        return repr(f)
    sign = '-' * (copysign(1.0, f) < 0)
    frac, fint = modf(fabs(f))  # split on fractional, integer parts
    n, d = frac.as_integer_ratio()  # frac = numerator / denominator
    assert d & (d - 1) == 0  # power of two
    return f'{sign}{floor(fint):b}.{n:0{d.bit_length() - 1}b}'

def add_binary_decimals(decimal_num1, decimal_num2):
    decimal_str1 = str(decimal_num1)
    decimal_str2 = str(decimal_num2)
    int_part1, frac_part1 = decimal_str1.split('.')
    int_part2, frac_part2 = decimal_str2.split('.')
    max_len = max(len(frac_part1), len(frac_part2))
    frac_part1 = frac_part1.ljust(max_len, '0')
    frac_part2 = frac_part2.ljust(max_len, '0')
    frac_sum = int(frac_part1, 2) + int(frac_part2, 2)
    int_part_sum = frac_sum // 2
    frac_sum_binary = bin(frac_sum % 2**max_len).replace("0b", "").rjust(max_len,
'0')
    int_sum = int(int_part1) + int(int_part2) + int_part_sum
    int_sum_binary = float_to_bin_fixed(int_sum)
    return int_sum_binary + '.' + frac_sum_binary

a = float(input("Введите a:"))
a1 = float_to_bin_fixed(a)
print("В двоичном коде:", a1)
b = float(input("Введите b:"))
b1 = float_to_bin_fixed(b)
print("В двоичном коде:", b1)
c = float(input("Введите c:"))
c1 = float_to_bin_fixed(c)
print("В двоичном коде:", c1)

d = (b**2 - 4*a*c)**0.5
x1 = (-b + d) / (2 * a)
x2 = (-b - d) / (2 * a)

print("x1 =", x1)
print("x2 =", x2)
```

Лабораторная работа №8

```
import math
import re
from math import copysign, fabs, floor, isfinite, modf

def float_to_bin_fixed(f):
    if not isfinite(f):
        return repr(f)
    sign = '-' * (copysign(1.0, f) < 0)
    frac, fint = modf(fabs(f))  # разбить на дробные, целые части
    n, d = frac.as_integer_ratio()  # frac = числитель / знаменатель
    assert d & (d - 1) == 0
    return f'{sign}{floor(fint):b}.{n:0{d.bit_length() - 1}b}'

def add_binary_decimals(decimal_num1, decimal_num2):
    decimal_str1 = str(decimal_num1)
    decimal_str2 = str(decimal_num2)
    int_part1, frac_part1 = decimal_str1.split('.')
    int_part2, frac_part2 = decimal_str2.split('.')
    max_len = max(len(frac_part1), len(frac_part2))
    frac_part1 = frac_part1.ljust(max_len, '0')
    frac_part2 = frac_part2.ljust(max_len, '0')
    frac_sum = int(frac_part1, 2) + int(frac_part2, 2)
    int_part_sum = frac_sum // 2
    frac_sum_binary = bin(frac_sum % 2**max_len).replace("0b", "").rjust(max_len,
'0')
    int_sum = int(int_part1) + int(int_part2) + int_part_sum
    int_sum_binary = float_to_bin_fixed(int_sum)
    return int_sum_binary + '.' + frac_sum_binary

def frange(start, stop, step):
    i = 0
    while True:
        value = start + i * step
        if value >= stop:
            break
        yield value
        i += 1

a = float(input("Введите начальное значение аргумента x: "))
b = float(input("Введите конечное значение аргумента x: "))
h = float(input("Введите шаг изменения аргумента x: "))
eps = float(input("Введите требуемую точность ε: "))

# Создаем пустые списки для хранения результатов
x_list = []
y_list = []
s_list = []
n_list = []
```

```
# Вычисляем значения функции Y(x), суммы S(x) и число итераций n
for x in frange(a, b, h):
    y: float = 0.5 - math.pi * abs(math.sin(x)) /4
    s: float = 0
    n: float = 0
    while True:
        n += 1
        s += math.cos(2 * n * x) / (4 * n ** 2 - 1)
        if abs(y - s) < eps:
            break
    x_list.append(x)
    y_list.append(y)
    s_list.append(s)
    n_list.append(n)

# Выводим результаты в виде таблицы
print("{:<10} {:<10} {:<10} {:<10}".format("x", "Y(x)", "S(x)", "n"))
for i in range(len(x_list)):
    print("{:<10.3f} {:<10.3f} {:<10.3f} {:<10}".format(x_list[i], y_list[i],
s_list[i], n_list[i]))
```

ВЫВОДЫ

В результате данной работы были изучены конфигурация арифмитического сопроцессора и возможные варианты синхронизации с центральным процессором, программная модель сопроцессора и используемые форматы численных данных, система команд сопроцессора. Результатом работы является разработанные программы для решения поставленных задач.