Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №7, №8

По дисциплине «Архитектура вычислительных систем»
По теме «Программирование арифметического сопроцессора»

Выполнил:
студент гр. 953501
Кременевский В.С.

Проверил:
Старший преподаватель
Шиманский В.В.

Минск 2021

Содержание

1. Цель работы

Ознакомиться с предназначением арифметического сопроцессора. Рассмотреть строение и основные принципы работы сопроцессора. Изучить команды, доступные при использовании сопроцессора. Научиться работать с сопроцессором в ходе выполнения лабораторного задания.

2. Постановка задачи

Задание к лабораторной работе 7

Написать программу, находящую решение квадратного уравнения

$$ax^2 + bx + c = 0$$

с помощью сопроцессора.

Задание к аабораторная работе 8

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.

Вариант: 14

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

3. 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 промежуточных 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 кодируется 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 тетради запрещающих комбинаций 1010 – 1111 результат операции не определен. Т.е. сопроцессор не контролирует правильность результата.

Вещественные числа. Различают короткие вещественные (КВ) (мантисса – 24 бита, порядок – 8 бит), длинные вещественные (ДВ) (мантисса – 53 бита, порядок – 11 бит) и временные вещественные (ВВ) (мантисса – 64 бита, порядок – 15 бит). Для них применяется формат с плавающей точкой. Значащие цифры находятся в поле мантиссы, порядок показывает фактическое положение двоичной точки в разрядах мантиссы, бит знака S определяет знак числа. Порядок дается в смещенной форме:

$$E = \text{истинный порядок} + \text{смещение}$$

Смещение для соответствующих форматов равно 127, 1023, 16383 это упрощает операцию сравнения. Операция с целыми числами быстрее операции над плавающей точкой. Это важно в алгоритмах.

Значение числа равно

$$(1)^{S} \times 2^{E\text{-смещение}} \times F_0F_1F_2\ldots F_n,$$

где n для разных форматов равно 23,52 или 63.

Порядок имеет фиксированную длину, определяя один диапазон представимых чисел. Мантисса – правильная дробь. В коротком и длинном вещественном формате бит F_0 при передаче чисел и хранении их в памяти не фигурирует. Это скрытый бит, который в нормализованных числах содержит 1. Скрытый бит не дает представить в этих форматах нуль, и он должен кодироваться как спец значение.

Числа во временном вещественном формате имеют явный бит F_0. Формат повышает скорость выполнения операций благодаря простоте представления чисел, не являющихся ненормализованными. При загрузке из памяти в регистр FPU оно преобразуется во временный вещественный формат. А при записи в память – обратный формат. Временной вещественный формат – единственный, в котором абсолютно точно кодируется любые загружаемые из памяти числа.

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:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	IC	RC		PC		IEM	X	PM	UM	OM	ZM	DM	IM

Регистр состояния:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	C3	ST			C2	C1	C0	IR	X	PE	UE	OE	ZE	DE	IE

Рис 1. Форматы слова управления и слова состояния

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 управления 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, сопроцессор 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.

Таблица 1. Интерпретация кода условия в операциях сравнения и проверки (FCOM,FCOMP,FTST)

C3	C2	C1	C0	Описание
0	X	X	0	(ST)>источника (src)
0	X	X	1	(ST)<источника (src)
1	X	X	0	(ST)=источнику (src)
1	X	X	1	Не сравнимы

Система команд

Система команд сопроцессора содержит 6 групп команд: команды передач данных, арифметические команды, команды сравнения, трансцендентных операций, команды загрузки констант и управления сопроцессором. Операнды некоторых команд определяются неявно. Другие команды допускают или требуют явного задания операнда.

При рассмотрении системы команд сопроцессора будем пользоваться следующими обозначениями: src обозначает источник, т.е. операнд, значение которого не модифицируется, а dst – получатель, т.е. операнд, значение которого замещается результатом операции.

Особенности задания команд. Команды бинарных операций допускают несколько форм задания**.** При пустом поле операнда операция выполняется с двумя верхними элементами стека ST(источник) и ST (1) (получатель). После производства операции осуществляется инкрементирование указателя стека и результат помещается в новую вершину стека, заменяя исходное содержимое ST (1). Когда в бинарной команде определен один операнд, операция выполняется с привлечением указанного в команде регистра или ячейки памяти и содержимого вершины стека. Результат загружается в старую вершину стека и указатель стека не модифицируется. Если в бинарной команде указаны 2 операнда, ими является содержимое 2-х регистров стека, причём одним из них будет ST, а 2-м ST(i).

Альтернативные формы операндов условно показываются с помощью наклонной черты, причем черта без последующей спецификации означает отсутствие явно задаваемых операндов. Например, команда FADD имеет следующий общий вид:

FADD //src/dst,src

Такая запись подразумевает три возможных формы команд: без операндов, с одним источником, с получателем и источником.

В мнемониках команд сопроцессора приняты следующие соглашения: первая буква всегда F; вторая буква I обозначает операцию с целыми числами, буква B – операцию с десятичным целым операндом, а пустая – операцию с вещественными числами; предпоследняя или последняя буква R указывает обратную операцию (например, в обычной форме команды деления получатель делится на источник, а в обратной форме источник делится на получатель; в обоих формах результат помещается в получатель); последняя буква P идентифицирует команду, заключительным действием которой является извлечение из стека.

Команды передачи данных. Команды этой группы производят передачу данных между регистрами стека, а также между вершинами стека и памятью.

Одной командой число из памяти преобразуется во ременный вещественный формат и загружается в стек. Таким же образом, но в обратном порядке осуществляется передача числа в память.

*Команды загрузки***.** 3 команды загрузки имеют следующий вид:

Вещественное: FLD src

двоичное целое: FILD src

десятичное целое: FBLD src

Эти команды осуществляют декремент указания стека и передачу в новую вершину стека содержимого источника. В команде FLD источником может быть один из регистров стека или вещественное число. В командах FILD и FBLD – только операнд в памяти.

Команды запоминания.

Вещественное: FST dst

Двоичное целое: FIST dst

Они производят передачу содержимого вершины стека в память без модификации указателя ST и содержимого ST (0). В команде FST получатель регистр стека или вещественная переменная в памяти. В команде FIST получателем является переменная в памяти имеющая формат коротко целого и целого слова. Рассмотренные команды не допускают получателем формат длинного целого, временного вещественного и неупакованного десятичного.

Команды запоминания с извлечением из стека. Три команды, помимо передачи содержимого ST (0) осуществляют извлечение из стека. Регистры бывшей вершины стека отмечается как пустой и производится инкремент указателя стека:

Вещественное: FSTP dst

Двоичное целое: FISTP dst

Десятичное целое: FBSTP dst

Действия команды FSTP очень похожи на действия FST с добавлением извлечения из стека. Однако, FSTP может передать в память слово во временном вещественном формате, чего не может сделать FST. Команда FIST обеспечивает передачу в память числа в любом формате целого двоичного, включая длинное целое. Последний формат не допустим в формате FSTP. Команда FBST преобразует операнд из вершины стека в упакованное десятичное число, передает его в память и производит извлечение из стека.

Команда обмена содержимого регистров.

FXCH // dst ST(0) ?? (dst) обменивает содержимое получателя ST(i) с вершиной ST(0). При пустом поле операнда обменивается содержимое регистров ST (1) и ST (0).

Команды управления.

Команда FINIT / FNINIT - инициализировать сопроцессор.

Команда FLDCW src – загрузить слово управления. Источником является целое число в памяти.

Команды FSTCW dst и FSTSW dst – запомнить слово управления и состояния в ячейке памяти, определяемой получателем dst.

Команда FSTENV dst – запомнить среду. Под средой сопроцессора К1810ВМ87 понимается содержимое регистров управления, состояния, тегов, указателя команды и указателя операнда. Команда FSTENV передаёт его в область памяти с начальным адресом, указанным в команде. Формат хранения среды в памяти показан на рис. 2.

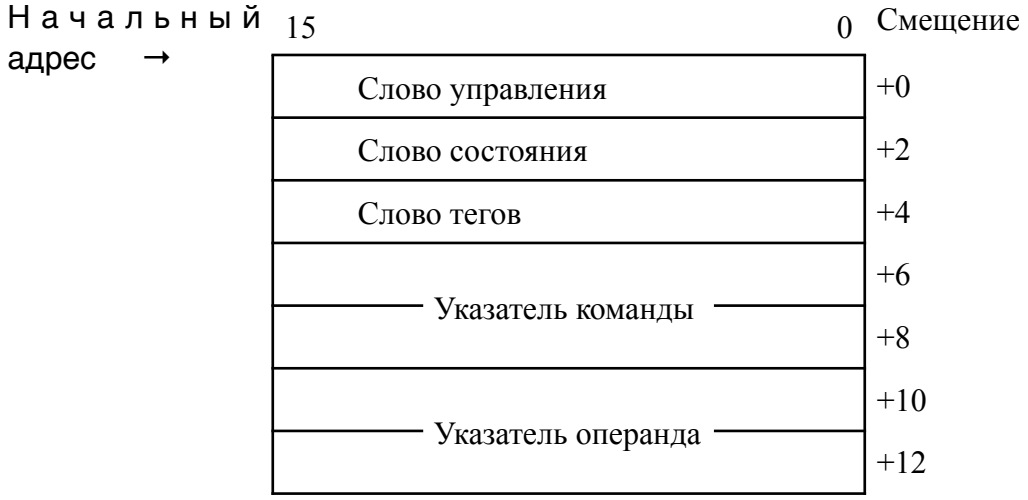

Рис. 2. Формат хранения среды сопроцессора в памяти.

До выборки из очереди следующей команды сопроцессора выполнение команды FSTENV должно закончиться.

Команда FLDENV src – загрузить среду. Парная предыдущей команде команда FLDENV src осуществляет загрузку среды сопроцессора из области памяти, определяемой src. После команды FLDENV не требуется команда FWAIT, так как сопроцессор автоматически контролирует завершение передачи в сех слов среды до перехода к своей следующей команде.

Команда FSAVE dst - сохранить полное состояние сопроцессора. Полное состояние сопроцессора представляет собой содержимое всех регистров программной модели – среды и восьми регистров стека. Размер полного состояния сопроцессора составляет 94 байта. Команда FSAVE передаёт его в область памяти с начальным адресом, указанным в команде. Формат размещения полного состояния сопроцессора в памяти (его иногда называют «образом в памяти») показан на рис.3.

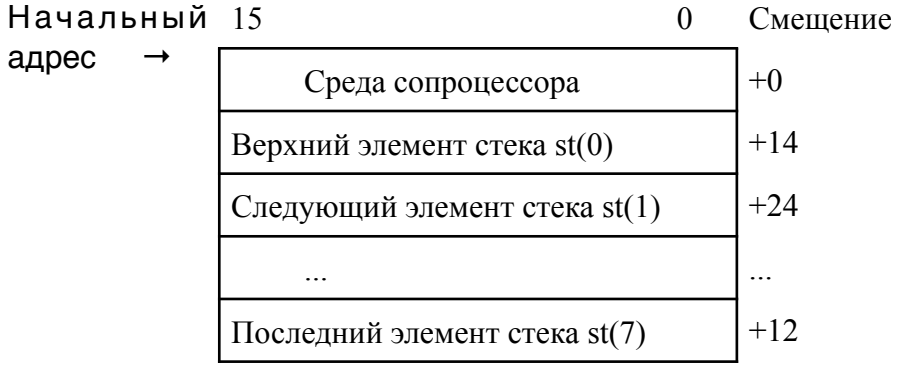

Рис. 3. Формат хранения полного состояния сопроцессора в памяти.

Команда FRSTOR src – восстановить полное состояние сопроцессора.

Для сохранения и восстановления состояния сопроцессора обычно прим еняется следующий ассемблерный фрагмент:

```
SUB    SP,94          ; Зарезервировать пространство в стеке
MOV    BP, SP         ; BP является базой для состояния
FSAVE  [BP]           ; Сохранить полное состояние
...
MOV    BP, SP         ; BP является базой для состояния
FRSTOR[BP]            ; Восстановить состояние
ADD    SP,94          ; Освободить пространство в стеке
```

Арифметические команды

Необходимо отметить, что вещественные числа в памяти, не могут быть в формате временного вещественного, а целые числа – в формате длинного целого. Здесь сказывается недостаточность наборов кодов операций.

Команды сложения. Операция сложения реализуется командами со следующими формами:

вещественные числа FADD //src/dst, src

вещественные числа с извлечением из стека FADDP dst,src

целые числа FIADD src

Отметим, что команда FADD ST, ST (0) удваивает содержимое вершины стека.

Команды вычитания. Обычное вычитание осуществляют команды:

вещественные числа FSUB //src/dst, scr

вещественные числа с извлечением из стека FSUBP dst,src

целые числа FISUB src

Для производства обратного вычитания предназначены команды FSUBR, FSUBRP, FISUBR, имеющие аналогичные формы.

*Команды умножения***.** Операция умножение реализуется следующими командами:

вещественные числа FMUL //src/dst, scr

вещественные числа с извлечением из стека FMULP dst,src

целые числа FIMUL src

Команды деления. Для выполнения обычной операции деления предусмотрены команды:

вещественные числа FDIV //src/dst, scr

вещественные числа с извлечением из стека FDIVP dst,src

целые числа FIDIV src

Соответствующие команды обратного деления FDIVR, FDIVRP, FIDIVR загружают в получатель частное от деления источника на получателя.

Приведём несколько примеров арифметических команд:

```
    FADD        ST, ST (5)            ; Сложить содержимое регистров
    FIADD       WORD PTR COUNT [SI]   ; Прибавить целое слово
    FSUBP       ST (2), ST            ; Вычесть содержимое регистров
    FDIVR       DWORD PTR [SI]        ;   Разделить   короткое
вещественное
    FIDIVR      DWORD PTR [BX+5]      ; Разделить короткое целое
```

Команда извлечения квадратного корня. Команда FSQRT извлечения квадратного корня заменяет число, находящееся в вершине стека, значением квадратного корня:

FSQRT ST (0) ? ST $(0)^{1/2}$.

4. Программная реализация

В качестве языка для написания программы с использованием арифметического сопроцессора был выбран язык C++.

Лабораторная работа 7.

С помощью консоли вводятся три вещественных числа – коэффициенты a, b, c уравнения $ax^2 + bx + c = 0$. Производится проверка на равенство нулю коэффициента a. По формуле $D = b^2 - 4ac$ вычисляется дискриминант. В зависимости от значения числа D определяется дальнейший ход программы:

1) D не является конечным числом – вызов исключения
2) $D < 0$ – уравнение не имеет вещественных корней
3) $D = 0$ – вычисление единственного корня по формуле $x = \frac{-b}{2a}$
4) $D > 0$ – вычисле ние двух корней по формуле $x_{1,2} = \frac{-b \pm \sqrt{D}}{2a}$

На рисунках представлен результат работы программы. Демонстрируется нахождение вещественных корней квадратного уравнения при различных значениях коэффициентов. Показана обработка исключительных ситуаций.

Обработка нулевого значения a:

Пример 1.

```
Enter a, b, c!
a = 0
b = 0
c = 0
Quadratic equation:
0 * x^2 + 0 * x + 0 = 0
a can't be zero!!
Equation isn't quadratic!
```

Пример 2.

```
Enter a, b, c!
a = 0
b = 100
c = 10
Quadratic equation:
0 * x^2 + 100 * x + 10 = 0
a can't be zero!!
Equation isn't quadratic!
```

Обработка отрицательно значения дискриминанта:

Пример 1.

```
Enter a, b, c!
a = 1
b = 1
c = 1
Quadratic equation:
1 * x^2 + 1 * x + 1 = 0

Discriminant is:
D = b^2 - 4ac
D = 1^2 - 4 * 1 * 1
D = -3
D < 0 !!!
No roots  in real numbers
```

Пример 2.

```
Enter a, b, c!
a = -4
b = 2.34
c = -922.90
Quadratic equation:
-4 * x^2 + 2.34 * x + -922.9 = 0

Discriminant is:
D = b^2 - 4ac
D = 2.34^2 - 4 * -4 * -922.9
D = -14760.9
D < 0 !!!
No roots  in real numbers
```

Пример 3.

```
Enter a, b, c!
a = 2
b = 5
c = 10
Quadratic equation:
2 * x^2 + 5 * x + 10 = 0

Discriminant is:
D = b^2 - 4ac
D = 5^2 - 4 * 2 * 10
D = -55
D < 0 !!!
No roots  in real numbers
```

Пример 4.

```
Enter a, b, c!
a = -123
b = -12
c = -100
Quadratic equation:
-123 * x^2 + -12 * x + -100 = 0

Discriminant is:
D = b^2 - 4ac
D = -12^2 - 4 * -123 * -100
D = -49056
D < 0 !!!
No roots  in real numbers
```

Успешное нахождение единственного корня уравнения

Пример 1.

```
Enter a, b, c!
a = 3
b = -18
c = 27
Quadratic equation:
3 * x^2 + -18 * x + 27 = 0

Discriminant is:
D = b^2 - 4ac
D = -18^2 - 4 * 3 * 27
D = 0
Finding roots:
Discriminant = 0!!
Then we have only 1 root!
x = -b / 2a
x = 18 / 2 * 3
x = 3
```

Пример 2.

```
Enter a, b, c!
a = 1
b = 4
c = 4
Quadratic equation:
1 * x^2 + 4 * x + 4 = 0

Discriminant is:
D = b^2 - 4ac
D = 4^2 - 4 * 1 * 4
D = 0
Finding roots:
Discriminant = 0!!
Then we have only 1 root!
x = -b / 2a
x = -4 / 2 * 1
x = -2
```

Пример 3.

```
Enter a, b, c!
a = 9
b = 6
c = 1
Quadratic equation:
9 * x^2 + 6 * x + 1 = 0

Discriminant is:
D = b^2 - 4ac
D = 6^2 - 4 * 9 * 1
D = 0
Finding roots:
Discriminant = 0!!
Then we have only 1 root!
x = -b / 2a
x = -6 / 2 * 9
x = -0.333333
```

Пример 4.

```
a = 2.25
b = -9
c = 9
Quadratic equation:
2.25 * x^2 + -9 * x + 9 = 0

Discriminant is:
D = b^2 - 4ac
D = -9^2 - 4 * 2.25 * 9
D = 0
Finding roots:
Discriminant = 0!!
Then we have only 1 root!
x = -b / 2a
x = 9 / 2 * 2.25
x = 2
```

Пример 5.

```
Enter a, b, c!
a = 1000000
b = 3000
c = 2.25
Quadratic equation:
1e+06 * x^2 + 3000 * x + 2.25 = 0

Discriminant is:
D = b^2 - 4ac
D = 3000^2 - 4 * 1e+06 * 2.25
D = 0
Finding roots:
Discriminant = 0!!
Then we have only 1 root!
x = -b / 2a
x = -3000 / 2 * 1e+06
x = -0.0015
```

Успешное нахождение двух корней уравнения

Пример 1.

```
Enter a, b, c!
a = 1
b = -7
c = 10
Quadratic equation:
1 * x^2 + -7 * x + 10 = 0

Discriminant is:
D = b^2 - 4ac
D = -7^2 - 4 * 1 * 10
D = 9
Finding roots:
x1 = (-b + sqrt(D) ) / 2a
x1 = (7 + sqrt(9)) / 2 * 1
x1 = 5
x2 = (-b - sqrt(D) ) / 2a
x2 = (7 - sqrt(9)) / 2 * 1
x2 = 2
    Result:
x1 = 5
x2 = 2
```

Пример 2.

```
Enter a, b, c!
a = 1
b = 6
c = 8
Quadratic equation:
1 * x^2 + 6 * x + 8 = 0

Discriminant is:
D = b^2 - 4ac
D = 6^2 - 4 * 1 * 8
D = 4
Finding roots:
x1 = (-b + sqrt(D) ) / 2a
x1 = (-6 + sqrt(4)) / 2 * 1
x1 = -2
x2 = (-b - sqrt(D) ) / 2a
x2 = (-6 - sqrt(4)) / 2 * 1
x2 = -4
    Result:
x1 = -2
x2 = -4
```

Пример 3.

```
Enter a, b, c!
a = 9033
b = 9000
c = -42
Quadratic equation:
9033 * x^2 + 9000 * x + -42 = 0

Discriminant is:
D = b^2 - 4ac
D = 9000^2 - 4 * 9033 * -42
D = 8.25175e+07
Finding roots:
x1 = (-b + sqrt(D) ) / 2a
x1 = (-9000 + sqrt(8.25175e+07)) / 2 * 9033
x1 = 379010
x2 = (-b - sqrt(D) ) / 2a
x2 = (-9000 - sqrt(8.25175e+07)) / 2 * 9033
x2 = -8.1676e+07
    Result:
x1 = 379010
x2 = -8.1676e+07
```

Пример 4.

```
Enter a, b, c!
a = 90
b = 29.123
c = 1.1111
Quadratic equation:
90 * x^2 + 29.123 * x + 1.1111 = 0

Discriminant is:
D = b^2 - 4ac
D = 29.123^2 - 4 * 90 * 1.1111
D = 448.153
Finding roots:
x1 = (-b + sqrt(D) ) / 2a
x1 = (-29.123 + sqrt(448.153)) / 2 * 90
x1 = -357.902
x2 = (-b - sqrt(D) ) / 2a
x2 = (-29.123 - sqrt(448.153)) / 2 * 90
x2 = -2263.17
    Result:
x1 = -357.902
x2 = -2263.17
```

Пример 5.

```
Enter a, b, c!
a = 100
b = 100000
c = 100
Quadratic equation:
100 * x^2 + 100000 * x + 100 = 0

Discriminant is:
D = b^2 - 4ac
D = 100000^2 - 4 * 100 * 100
D = 9.99996e+09
Finding roots:
x1 = (-b + sqrt(D) ) / 2a
x1 = (-100000 + sqrt(9.99996e+09)) / 2 * 100
x1 = -10
x2 = (-b - sqrt(D) ) / 2a
x2 = (-100000 - sqrt(9.99996e+09)) / 2 * 100
x2 = -9.99999e+06
    Result:
x1 = -10
x2 = -9.99999e+06
```

Лабораторная работа 8.

Вариант 14

$$14.\ S(x)=\sum_{k=1}^{n}\frac{\cos(2kx)}{4k^2-1}, \qquad Y(x)=\frac{1}{2}-\frac{\pi}{4}\left|\sin(x)\right|.$$

В ходе анализа поведений следующий функций, выяснено, что их разность по модулю, то есть | Y(x) - S(x) | —> 0.5 (стремится к 0.5, если количеситво итераций стремится к бесконечности).

Начнем с задания небольшой точноти, далее постепенно будем ее увеличивать и следить за количеством итераций, которые понадобились для ее достижения.

Пример 1.

Точность: 0.1

```
S = sum ( cos(2kx) / (4*k*k - 1) )
Y = 1/2 - PI/4 * | sin(x) |

|Y(x) - S(x)|   стремиться к 0.5. (при количестве итераций -> бесконечности)
Исходные данные:
a = 0 | b = 2 | h = 1 | e = 0.1

    x = 0
    Y = 0
1. S = 0.333333 | diff = 0.333333
2. S = 0.4 | diff = 0.4
3. S = 0.428571 | diff = 0.428571
Для достижения необходимой точночти понадобилось
итераций = 3

    x = 1
    Y = -0.66089
1. S = -0.138716 | diff = 0.522174
Для достижения необходимой точночти понадобилось
итераций = 1

    x = 2
    Y = -0.714161
1. S = -0.217881 | diff = 0.496279
Для достижения необходимой точночти понадобилось
итераций = 1
```

Пример 2.

Точность: 0.01

```
S = sum ( cos(2kx) / (4*k*k - 1) )
Y = 1/2 - PI/4 * | sin(x) |

|Y(x) - S(x)|   стремиться к 0.5. (при количестве итераций -> бесконечности)
Исходные данные:
a = 50.5 | b = 51 | h = 0.1 | e = 0.01

    x = 50.5
    Y = -0.182506
1. S = 0.297335 | diff = 0.479841
2. S = 0.336758 | diff = 0.519264
3. S = 0.341414 | diff = 0.52392
4. S = 0.336642 | diff = 0.519148
5. S = 0.329579 | diff = 0.512085
6. S = 0.322957 | diff = 0.505463
Для достижения необходимой точночти понадобилось
итераций = 6

    x = 50.6
    Y = -0.257857
1. S = 0.261473 | diff = 0.51933
2. S = 0.276849 | diff = 0.534706
3. S = 0.264775 | diff = 0.522632
4. S = 0.25059 | diff = 0.508447
Для достижения необходимой точночти понадобилось
итераций = 4


     x = 50.7
     Y = -0.330631
 1. S = 0.215188 | diff = 0.545819
 2. S = 0.204088 | diff = 0.53472
 3. S = 0.179502 | diff = 0.510133
 4. S = 0.164509 | diff = 0.49514
 Для достижения необходимой точночти понадобилось
 итераций = 4

     x = 50.8
     Y = -0.400102
 1. S = 0.160323 | diff = 0.560426
 2. S = 0.124501 | diff = 0.524603
 3. S = 0.095991 | diff = 0.496093
 Для достижения необходимой точночти понадобилось
 итераций = 3

     x = 50.9
     Y = -0.465575
 1. S = 0.0990674 | diff = 0.564643
 2. S = 0.044178 | diff = 0.509753
 Для достижения необходимой точночти понадобилось
 итераций = 2
```

Пример 3.

Точность: 0.001

```
S = sum ( cos(2kx) / (4*k*k - 1) )
Y = 1/2 - PI/4 * | sin(x) |

|Y(x) - S(x)|   стремиться к 0.5. (при количестве итераций -> бесконечности)
Исходные данные:
a = 100 | b = 105 | h = 1 | e = 0.001

    x = 100
    Y = -0.397699
1. S = 0.162396 | diff = 0.560095
2. S = 0.127376 | diff = 0.525075
3. S = 0.0988326 | diff = 0.496531
4. S = 0.0917195 | diff = 0.489418
5. S = 0.0974001 | diff = 0.495099
6. S = 0.104366 | diff = 0.502064
7. S = 0.106459 | diff = 0.504158
8. S = 0.104113 | diff = 0.501811
9. S = 0.101044 | diff = 0.498742
10. S = 0.100123 | diff = 0.497821
11. S = 0.101434 | diff = 0.499132
Для достижения необходимой точночти понадобилось
итераций = 11
```

```
    x = 101
    Y = -0.35502
1. S = 0.197115 | diff = 0.552135
2. S = 0.177074 | diff = 0.532094
3. S = 0.15002 | diff = 0.50504
4. S = 0.137016 | diff = 0.492036
5. S = 0.136793 | diff = 0.491813
6. S = 0.14234 | diff = 0.49736
7. S = 0.147264 | diff = 0.502284
8. S = 0.148606 | diff = 0.503627
9. S = 0.146887 | diff = 0.501908
10. S = 0.144383 | diff = 0.499404
Для достижения необходимой точночти понадобилось
итераций = 10

    x = 102
    Y = -0.781335
1. S = -0.326454 | diff = 0.454882
2. S = -0.265234 | diff = 0.516101
3. S = -0.288643 | diff = 0.492692
4. S = -0.277746 | diff = 0.503589
5. S = -0.283053 | diff = 0.498282
6. S = -0.280657 | diff = 0.500678
Для достижения необходимой точночти понадобилось
итераций = 6
```

```
    x = 103
    Y = -0.489294
1. S = 0.0745901 | diff = 0.563884
2. S = 0.0145999 | diff = 0.503894
3. S = -0.00329989 | diff = 0.485994
4. S = 0.00653302 | diff = 0.495827
5. S = 0.0156616 | diff = 0.504956
6. S = 0.014158 | diff = 0.503452
7. S = 0.00903 | diff = 0.498324
8. S = 0.00811821 | diff = 0.497412
9. S = 0.0108919 | diff = 0.500186
Для достижения необходимой точночти понадобилось
итераций = 9

    x = 104
    Y = -0.252602
1. S = 0.264373 | diff = 0.516974
2. S = 0.281578 | diff = 0.534179
3. S = 0.270613 | diff = 0.523215
4. S = 0.256854 | diff = 0.509456
5. S = 0.246842 | diff = 0.499444
Для достижения необходимой точночти понадобилось
итераций = 5
```

```
    x = 105
    Y = -0.762257
1. S = -0.294626 | diff = 0.467631
2. S = -0.257127 | diff = 0.505129
3. S = -0.260283 | diff = 0.501974
4. S = -0.266112 | diff = 0.496145
5. S = -0.258439 | diff = 0.503818
6. S = -0.265261 | diff = 0.496995
7. S = -0.260313 | diff = 0.501944
8. S = -0.263177 | diff = 0.49908
Для достижения необходимой точночти понадобилось
итераций = 8
```

Пример 4.

Точность: 0.0001

```
S = sum ( cos(2kx) / (4*k*k - 1) )
Y = 1/2 - PI/4 * | sin(x) |

|Y(x) - S(x)|   стремиться к 0.5. (при количестве итераций -> бесконечности)
Исходные данные:
a = 9999.7 | b = 10000 | h = 0.1 | e = 0.0001

    x = 9999.7
    Y = -0.0083122
1. S = 0.333259 | diff = 0.341571
2. S = 0.399866 | diff = 0.408178
3. S = 0.428379 | diff = 0.436692
4. S = 0.444196 | diff = 0.452508
5. S = 0.45424 | diff = 0.462552
6. S = 0.461177 | diff = 0.469489
7. S = 0.466249 | diff = 0.474561
8. S = 0.470114 | diff = 0.478426
9. S = 0.473154 | diff = 0.481466
10. S = 0.475605 | diff = 0.483917
11. S = 0.477619 | diff = 0.485931
```

```
36. S = 0.49114 | diff = 0.499453
37. S = 0.49127 | diff = 0.499582
38. S = 0.49139 | diff = 0.499702
39. S = 0.491501 | diff = 0.499814
40. S = 0.491605 | diff = 0.499917
Для достижения необходимой точночти понадобилось
итераций = 40
```

```
    x = 9999.8
    Y = -0.0866753
1. S = 0.325214 | diff = 0.411889
2. S = 0.385464 | diff = 0.47214
3. S = 0.407974 | diff = 0.494649
4. S = 0.41803 | diff = 0.504706
5. S = 0.42256 | diff = 0.509235
6. S = 0.424248 | diff = 0.510923
7. S = 0.424364 | diff = 0.511039
8. S = 0.42359 | diff = 0.510265
9. S = 0.422328 | diff = 0.509004
10. S = 0.42083 | diff = 0.507505
```

```
24. S = 0.411739 | diff = 0.498414
25. S = 0.412031 | diff = 0.498706
26. S = 0.412349 | diff = 0.499025
27. S = 0.412676 | diff = 0.499351
28. S = 0.412994 | diff = 0.499669
29. S = 0.413288 | diff = 0.499964
Для достижения необходимой точночти понадобилось
итераций = 29
```

```
    x = 9999.9
    Y = -0.164172
1. S = 0.304204 | diff = 0.468376
2. S = 0.348586 | diff = 0.512758
3. S = 0.357228 | diff = 0.5214
4. S = 0.355424 | diff = 0.519597
5. S = 0.350274 | diff = 0.514446
6. S = 0.344561 | diff = 0.508733
7. S = 0.339528 | diff = 0.5037
8. S = 0.335708 | diff = 0.49988
9. S = 0.333241 | diff = 0.497413
10. S = 0.332038 | diff = 0.49621
11. S = 0.331873 | diff = 0.496046
12. S = 0.332456 | diff = 0.496628
13. S = 0.33348 | diff = 0.497652
14. S = 0.334662 | diff = 0.498835
15. S = 0.335774 | diff = 0.499946
Для достижения необходимой точночти понадобилось
итераций = 15
```

Пример 5.

Точность: 0.000001

```
|Y(x) - S(x)|   стремиться к 0.5. (при количестве итераций -> бесконечности)
Исходные данные:
a = 10007 | b = 10010 | h = 1 | e = 1e-06

    x = 10007
    Y = -0.672267
1. S = -0.155108 | diff = 0.517159
2. S = -0.192904 | diff = 0.479362
3. S = -0.164534 | diff = 0.507732
4. S = -0.170203 | diff = 0.502064
5. S = -0.176876 | diff = 0.495391
6. S = -0.170079 | diff = 0.502188
7. S = -0.17133 | diff = 0.500937
8. S = -0.174251 | diff = 0.498016
9. S = -0.17135 | diff = 0.500917
10. S = -0.171449 | diff = 0.500508
```

```
75. S = -0.172291 | diff = 0.499976
76. S = -0.172264 | diff = 0.500003
77. S = -0.172247 | diff = 0.50002
78. S = -0.172288 | diff = 0.499979
79. S = -0.172267 | diff = 0.5
Для достижения необходимой точночти понадобилось
итераций = 79
```

```
    x = 10008
    Y = -0.704939
1. S = -0.203738 | diff = 0.501201
2. S = -0.220593 | diff = 0.484345
3. S = -0.1943 | diff = 0.510639
4. S = -0.208143 | diff = 0.496796
5. S = -0.20667 | diff = 0.498269
6. S = -0.201818 | diff = 0.503121
7. S = -0.206915 | diff = 0.498023
8. S = -0.204871 | diff = 0.500068
9. S = -0.203767 | diff = 0.501172
10. S = -0.206166 | diff = 0.498773
11. S = -0.204482 | diff = 0.500457
12. S = -0.204546 | diff = 0.500393

64. S = -0.204962 | diff = 0.499976
65. S = -0.204907 | diff = 0.500032
66. S = -0.204955 | diff = 0.499984
67. S = -0.20495 | diff = 0.499988
68. S = -0.20491 | diff = 0.500029
69. S = -0.204962 | diff = 0.499977
70. S = -0.204938 | diff = 0.5
Для достижения необходимой точночти понадобилось
итераций = 70


    x = 10009
    Y = -0.0894932
1. S = 0.324677 | diff = 0.414171
2. S = 0.384509 | diff = 0.474003
3. S = 0.406633 | diff = 0.496126
4. S = 0.41633 | diff = 0.505823
5. S = 0.42053 | diff = 0.510024
6. S = 0.421923 | diff = 0.511416
7. S = 0.42178 | diff = 0.511273
8. S = 0.420785 | diff = 0.510279
9. S = 0.419343 | diff = 0.508836

147. S = 0.410547 | diff = 0.50004
148. S = 0.410539 | diff = 0.500032
149. S = 0.410529 | diff = 0.500022
150. S = 0.410518 | diff = 0.500012
151. S = 0.410507 | diff = 0.500001
Для достижения необходимой точночти понадобилось
итераций = 151
```

```
    x = 10010
    Y = -0.608232
1. S = -0.0664893 | diff = 0.541743
2. S = -0.127851 | diff = 0.480381
3. S = -0.111661 | diff = 0.496571
4. S = -0.100639 | diff = 0.507593
5. S = -0.109161 | diff = 0.499071

70. S = -0.108255 | diff = 0.499977
71. S = -0.108205 | diff = 0.500026
72. S = -0.108221 | diff = 0.500011
73. S = -0.108261 | diff = 0.499971
74. S = -0.108231 | diff = 0.500001
Для достижения необходимой точночти понадобилось
итераций = 74
```

Пример 6.

Точность: 0.00000001

```
|Y(x) - S(x)|   стремиться к 0.5. (при количестве итераций -> бесконечности)
Исходные данные:
a = -1000.3 | b = -1000 | h = 0.1 | e = 1e-08

    x = -1000.3
    Y = -0.750953
1. S = -0.276139 | diff = 0.474814
2. S = -0.251302 | diff = 0.49965
3. S = -0.245269 | diff = 0.505683
4. S = -0.256736 | diff = 0.494217
5. S = -0.246779 | diff = 0.504174
6. S = -0.253148 | diff = 0.497804
7. S = -0.250465 | diff = 0.500488
8. S = -0.250293 | diff = 0.50066
9. S = -0.252138 | diff = 0.498815
10. S = -0.249773 | diff = 0.501179
11. S = -0.251776 | diff = 0.499177
12. S = -0.250629 | diff = 0.500323
13. S = -0.250814 | diff = 0.500138
14. S = -0.251392 | diff = 0.499561
15. S = -0.25042 | diff = 0.500533
16. S = -0.251393 | diff = 0.499559
```

```
439. S = -0.250952 | diff = 0.500001
440. S = -0.250953 | diff = 0.5
441. S = -0.250953 | diff = 0.5
442. S = -0.250952 | diff = 0.5
443. S = -0.250953 | diff = 0.499999
444. S = -0.250952 | diff = 0.500001
445. S = -0.250953 | diff = 0.5
446. S = -0.250953 | diff = 0.5
Для достижения необходимой точночти понадобилось
итераций = 446
```

```
     x = -1000.2
     Y = -0.724235
 1. S = -0.233542 | diff = 0.490693
 2. S = -0.234759 | diff = 0.489476
 3. S = -0.21401 | diff = 0.510225
 4. S = -0.229873 | diff = 0.494362
 5. S = -0.223063 | diff = 0.501172
 6. S = -0.222681 | diff = 0.501554
 7. S = -0.226531 | diff = 0.497704
 8. S = -0.22262 | diff = 0.501615
 9. S = -0.224622 | diff = 0.499613
 10. S = -0.22485 | diff = 0.499384
 11. S = -0.223247 | diff = 0.500988
 12. S = -0.224976 | diff = 0.499259
```

```
 461. S = -0.224234 | diff = 0.5
 462. S = -0.224236 | diff = 0.499999
 463. S = -0.224234 | diff = 0.5
 464. S = -0.224235 | diff = 0.5
 465. S = -0.224235 | diff = 0.5
 466. S = -0.224234 | diff = 0.500001
 467. S = -0.224235 | diff = 0.5
 468. S = -0.224235 | diff = 0.5
 469. S = -0.224234 | diff = 0.5
 470. S = -0.224235 | diff = 0.499999
 471. S = -0.224234 | diff = 0.5
 472. S = -0.224235 | diff = 0.5
 Для достижения необходимой точночти понадобилось
 итераций = 472
```

```
    x = -1000.1
    Y = -0.690281
1. S = -0.181635 | diff = 0.508646
2. S = -0.208712 | diff = 0.481569
3. S = -0.180497 | diff = 0.509784
4. S = -0.191133 | diff = 0.499148
5. S = -0.193732 | diff = 0.496549
6. S = -0.187085 | diff = 0.503196
7. S = -0.191078 | diff = 0.499203
8. S = -0.191478 | diff = 0.498803
9. S = -0.188723 | diff = 0.501558
10. S = -0.190898 | diff = 0.499383
11. S = -0.190782 | diff = 0.499499
12. S = -0.189379 | diff = 0.500902
13. S = -0.190764 | diff = 0.499517


91. S = -0.190264 | diff = 0.500017
92. S = -0.190291 | diff = 0.49999
93. S = -0.190286 | diff = 0.499995
94. S = -0.190265 | diff = 0.500015
95. S = -0.190292 | diff = 0.499988
96. S = -0.190283 | diff = 0.499998
97. S = -0.190267 | diff = 0.500014
98. S = -0.190293 | diff = 0.499988
99. S = -0.190281 | diff = 0.5
Для достижения необходимой точночти понадобилось
итераций = 99
```

5. Выводы

В ходе выполнения лабораторной работы были изучены понятия арифметического сопроцессора, сопроцессорных конфигураций, программной модели сопроцессора. Рассмотрены различные форматы представления численных данных: двоичные целые числа, упакованные десятичные числа, вещественные числа. Выяснены режимы работы и состояния арифметического сопроцессора, система команд сопроцессора, особенности задания команд, различные группы команд сопроцессора (команды управления, команды передачи данных, команды загрузки, команды запоминания, арифметические команды).

Написана программа, производящая решение квадратного уравнения с использованием арифметического сопроцессора. Также осуществлена проверка на возникновение исключительных ситуаций в процессе вычислений.

Также написана программа позволяющая находить значения функций и находить количество итераций необходимых для требуемой точности с использованием арифметического сопроцессора.

Список литературы

1. Волорова Н. А. Лабораторный практикум по курсу «Архитектура вычислительных систем» для студентов специальности «Информатика» / 93-444-487-2- Мн.: БГУИР, 2003. – 32с.:ил.

Приложение 1. Текст программы

Решение квадратного уравнения:

```
#include <iostream>
#include <cmath>

#define show true

class QuadraticEquation {
    double a;
    double b;
    double c;
    double D;
    double x1;
    double x2;

public:
    QuadraticEquation() {
        this->a = 1;
        this->b = 1;
        this->c = 1;
    }

    QuadraticEquation (double a, double b, double c) {
        this->a = a;
        this->b = b;
        this->c = c;
        if (show) {
            std::cout << "Quadratic equation:\n";
            std::cout << a << " * x^2 + " << b << " * x + " << c
<< " = 0\n";
        }
    }

    double findDiscriminant() {
        D = pow(b, 2) - 4 * a * c;
        if (show) {
            std::cout << "\nDiscriminant is: \n";
            std::cout << "D = b^2 - 4ac\n";
            std::cout << "D = " << b << "^2 - 4 * " << a << " *
" << c << '\n';
            std::cout << "D = " << D << '\n';
        }
    };

    double getDiscriminant() {
        return D;
    }
```

```
    void solve () {
        if (a == 0) {
            std::cout << "a can't be zero!!\n";
            std::cout << "Equation isn't quadratic!\n";
            return;
        }
        findDiscriminant();
        if (D < 0) {
            std::cout << "D < 0 !!!\n";
            std::cout << "No roots  in real numbers\n";
            return;
        }
        if (D != 0) {
            x1 = (-b + sqrt(D)) / 2 * a;
            x2 = (-b - sqrt(D)) / 2 * a;
        }
        else {
            x1 = -b / (2 * a);
            x2 = -b / (2 * a);
        }

        if (show) {
            std::cout << "Finding roots:\n";
            if (x1 != x2 || D != 0) {
                std::cout << "x1 = (-b + sqrt(D) ) / 2a\n";
                std::cout << "x1 = " << "(" << -b << " + sqrt("
<< D << ")) / 2 * " << a << "\n";
                std::cout << "x1 = " << x1 << '\n';

                std::cout << "x2 = (-b - sqrt(D) ) / 2a\n";
                std::cout << "x2 = " << "(" << -b << " - sqrt("
<< D << ")) / 2 * " << a << "\n";
                std::cout << "x2 = " << x2 << '\n';
                std::cout << "\tResult:\n";
                std::cout << "x1 = " << x1 << '\n';
                std::cout << "x2 = " << x2 << '\n';
            }
            else {
                std::cout << "Discriminant = 0!!\n";
                std::cout << "Then we have only 1 root!\n";
                std::cout << "x = -b / 2a\n";
                std::cout << "x = " << -b << " / 2 * " << a <<
'\n';
                std::cout << "x = " << x1 << '\n';
            }
        }
    }
};


void solveQuadratic() {
```

```
    double a, b, c;
    std::cout << "Enter a, b, c! \n";
    std::cout << "a = ";
    std::cin >> a;
    std::cout << "b = ";
    std::cin >> b;
    std::cout << "c = ";
    std::cin >> c;

    QuadraticEquation equation(a, b, c);
    equation.solve();
}


int main() {
    solveQuadratic();

    return 0;
}
```

Нахождение функций с заданной точностью:

```
#include <iostream>
#include <cmath>

class FindSumFunc{
public:
    double a;
    double b;
    double h;
    double e;
    double S;
    double x;
    double Y;
    int iteration_cnt;

    FindSumFunc(double a, double b, double h, double e) {
        this->a = a;
        this->b = b;
        this->h = h;
        this->e = e;
        this->Y = 0;
        this->S = 0;
        this->iteration_cnt = 0;
    }
```

```
    double find_y(double xx) {
        double y = 1/2 - M_PI / 4 * abs(sin(xx));
        return y;
    }

    double find_s(int k) {
        double s = cos(2 * k * this->x) / (4 * k * k - 1);
        return s;
    }


    double solve() {
        std::cout << "\t\tФункции варианта 14: \n\n";
        std::cout << "S = sum ( cos(2kx) / (4*k*k - 1) )\n";
        std::cout << "Y = 1/2 - PI/4 * | sin(x) |\n\n";
        std::cout << "|Y(x) - S(x)|   стремиться к 0.5. (при
количестве итераций -> бесконечности)\n";
        std::cout << "Исходные данные: \n";
        std::cout << "a = " << a << " | b = " << b << " | h = "
<< h << " | e = " << e << "\n\n";
        x = a;
        for(int k = 1; x <= b; x += h) {
            std::cout << "\tx = " << x << '\n';

            Y = find_y(x);
            std::cout << "\tY = " << Y << '\n';

            iteration_cnt = 0;
            double sum = 0;
            double diff = 0;
            do {
                ++iteration_cnt;
                sum += find_s(iteration_cnt);
                diff = abs(Y - sum);
                std::cout << iteration_cnt << ". S = " << sum <<
" | diff = " << diff << '\n';
            }
            while(abs(diff - 0.5) > e);
            std::cout << "Для достижения необходимой точночти
понадобилось\nитераций = " << iteration_cnt << '\n';
            std::cout << "\n";
        }
    }

};



void preprocessing(){
    double a;
```

```
//    std::cout << "a = ";
//    std::cin >> a;
//
//    double b;
//    std::cout << "b = ";
//    std::cin >> b;
//
//    double h;
//    std::cout << "h = ";
//    std::cin >> h;
//
//    double e;
//    std::cout << "e = ";
//    std::cin >> e;

    FindSumFunc task(-1000.3, -1000, 0.1 , 0.00000001);
    task.solve();
}


int main() {
//    FindSumFunc a(2, 2, 2, 2);
//    a.solve();
    preprocessing();
}
```