

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

Отчёт по лабораторной работе №7

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	3
2. Постановка задачи	4
3. Теоретические сведения	5
Сопроцессорные конфигурации	5
Программная модель сопроцессора	5
Форматы численных данных	6
Режимы работы. Состояние	8
Система команд	10
Арифметические команды	13
4. Программная реализация	15
5. Выводы	18
Список литературы	19
Приложение 1. Текст программы	20

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.

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) $0 \leq i < 7$, 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 находится 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 K1810BM87 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 (KB) (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

$$(1) S \times 2^{E-\text{смещение}} \times F_0 F_1 F_2 \dots F_n,$$

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 (РС) определяет точность вычислений в 24 бита (РС=00), 53 бита (РС=10) или 64 бита (РС=11). По умолчанию вводится режим с максимальной точностью в 64 бита.

Двухбитное поле управления округлением РС 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 «В» устанавливается в состояние 1 когда численные 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 обеих формах результат помещается в получатель); последняя буква *R* 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 реченный 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 K1810BM87 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.

Н а ч а л ь н ы й	15	0	Смещение
адрес →			
	Слово управления		+0
	Слово состояния		+2
	Слово тегов		+4
			+6
	Указатель команды		+8
			+10
	Указатель операнда		+12

Рис. 2. Формат хранения среды сопроцессора в памяти.

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

Команда FL DENV src – загрузить среду. Парная предыдущей команде команда FL DENV src осуществляет загрузку среды сопроцессора из области памяти, определяемой src. После команды FL DENV 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.

Начальный адрес	15	0	Смещение
→			
	Среда сопроцессора		+0
	Верхний элемент стека st(0)		+14
	Следующий элемент стека st(1)		+24

	Последний элемент стека st(7)		+12

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

Команда FRSTOR src – восстановить полное состояние сопроцессора.

Для сохранения и восстановления состояния сопроцессора обычно применяется 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) $\boxed{?}$ ST (0)^{1/2}.

4. Программная реализация

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

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) $D > 0$ – вычисление 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
```

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.

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

```
#include <iostream>
#include <string>
#include <windows.h>

using namespace std;

string Solve(long double a, long double b, long double c)
{
    const long double LD_4 = 4.0L;
    if (a == 0) {
        return "Уравнение не является квадратным";
    }
    long double D;
    __asm
    {
        FLD b
        FLD b
        FMUL
        FLD LD_4
        FLD a
        FLD c
        FMUL
        FMUL
        FSUB
        FSTP D
    }
    if (!isfinite(D)) {
        return "Дискриминант не является конечным числом";
    }
    if (D < 0.0L) {
        return "Нет вещественных корней";
    }
    if (D == 0.0L) {
        long double x;
        __asm
        {
            FLD b
            FCHS
            FLD a
            FLD a
            FADD
            FDIV
            FSTP x
        }
        return "x = " + to_string(x);
    }
    if (D > 0.0L) {
        long double x1, x2;
        __asm
```

```

        {
            FLD b
            FCHS
            FLD D
            FSQRT

            FSUB

            FLD a
            FLD a
            FADD
            FDIV

            FSTP x1
        }
    } asm
    {
        FLD b
        FCHS
        FLD D
        FSQRT

        FADD

        FLD a
        FLD a
        FADD
        FDIV

        FSTP x2
    }

    return "x1 = " + to_string(x1) + ", " +
        "x2 = " + to_string(x2);
}
return "Что-то пошло не так";
}

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    while (true) {
        long double a, b, c;
        cout << endl << "Введите коэффициенты a, b, c:" << endl;
        cin >> a >> b >> c;
        cout << Solve(a, b, c) << endl;
    }
    return 0;
}

```