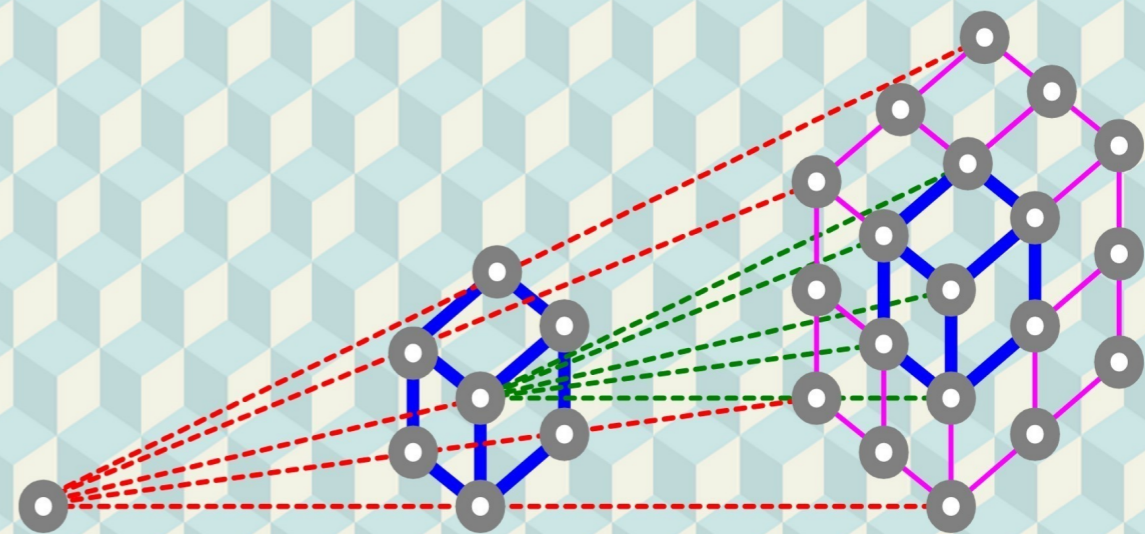



Компьютерная арифметика чисел, векторов, фигур и функций



С.И.Хмельник

Компьютерная арифметика чисел, векторов, фигур и функций. Алгоритмы и аппаратура 

ISBN 978-0-557-35768-0

ID: 8462841
www.lulu.com



9 780557 357680

90000

Компьютерная арифметика чисел, векторов, фигур и функций

Алгоритмы и аппаратура

Соломон Ицкович Хмельник

Россия Израиль
2008

Computer Arithmetic of Numbers, Vectors, Figures and Finctions

Algorithms and Hardware Design
(in Russian)

Solomon I. Khmelnik

Copyright © 2004 by Solomon I. Khmelnik

All right reserved. No portion of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, without written permission of the author.

Published by “MiC” - Mathematics in Computer Comp.

BOX 15302, Beney-Ayish, Israel, 60860

Fax: ++972-8-8691348

Printed in United States of America, Lulu Inc. **ID 7325828**

Russia Israel
2009

Памяти родителей
Ицко и Ревекки
посвящаю



Аннотация

В книге описываются малоизвестные методы кодирования математических объектов — действительных и комплексных чисел, многомерных векторов, плоских и пространственных геометрических фигур, функций одного и многих аргументов. Рассматриваются свойства получаемых кодов, алгоритмы различных операций с ними, аппаратная реализация этих алгоритмов с целью построения специализированных процессоров. Изложение иллюстрируется многочисленными примерами.

Оглавление

Подробное оглавление \ 0-6

Предисловие \ 0-21

Литература \ 0-24

Часть 1. Коды действительных чисел по
отрицательному основанию \ 1-1

Часть 2. Коды комплексных чисел и векторов
\ 2-1

Часть 3. Коды геометрических фигур \ 3-1

Часть 4. Коды функций \ 4-1

Приложение. Исторически первая статья о
кодировании комплексных чисел.

Подробное оглавление

Предисловие \ 0-21

Часть 1. Коды действительных чисел по отрицательному основанию \ 1-1

Предисловие \ 1-4

Глава 1. Представление чисел \ 1-6

1. Вступление \ 6
2. Р-коды \ 7
3. М-коды \ 8
4. С-коды \ 9

Глава 2. Алгебраическое сложение М-кодов \ 1-11

1. Введение \ 11
2. Одноразрядные схемы алгебраического сложения М-кодов \ 13
 1. Инвертор - Inv \ 13
 2. Инвертор – удвоитель - Inv2 \ 13
 3. Инверсный сумматор - InvAdd \ 14
 4. Сумматор - Add \ 15
 5. Вычитатель - Sub \ 15
 6. Алгебраический сумматор - AddAlg \ 16
 7. Вычитатель с удвоением - Sub2 \ 18
 8. Инверсный вычитатель с добавлениями - InvSubP \ 18
 9. Вычитатель с добавлениями - SubP \ 21
 10. Вычитатель с уменьшениями - SubM \ 22
 11. Знакоопределитель - Seven, Sodd \ 23
3. Многоразрядные схемы алгебраического сложения М-кодов \ 26
 1. Линейные схемы \ 26
 2. Вычитатель разреженных кодов с добавлением 2 - nSubP \ 28

3. Вычитатель разреженных кодов с вычитанием 2 - nSubM \ 28
4. Инверсный вычитатель разреженных кодов с добавлением 2 - nInvSubP \ 29
5. Многоярусные схемы \ 30
6. Трехярусный алгебраический сумматор - nAddAlg3 \ 31
7. Трехярусный сумматор - nAdd3 \ 31
8. Трехярусный вычитатель - nSub3 \ 31
9. Знакоопределитель - nSign \ 32

Глава 3. Алгебраическое сложение С-кодов \ 1-33

1. Многоразрядные схемы алгебраического сложения С-кодов \ 33

1. Алгебраический сумматор мантисс С-кода - SumM2 \ 33
2. Полный алгебраический сумматор мантисс С-кода - AddFull \ 35

2. Операции алгебраического сложения С-кодов \ 38

1. Унарные операции с целыми кодами – Uni \ 38
2. Унарные операции с нецелыми кодами – Unf \ 39
3. Бинарные операции с целыми кодами – Bini \ 40
4. Bini с запоминанием переполнения – BiniR \ 40
5. Bini с учетом предыдущего переполнения – BiniW \ 41
6. Bini с учетом предыдущего переполнения и запоминанием переполнения – BiniWR \ 41
7. Bini удвоенной точности \ 41
8. Бинарные операции с нецелыми кодами – Binf \ 42

Глава 4. Групповой перенос \ 1-43

1. Терия группового переноса \ 43

0. Введение \ 43
1. Определение критических комбинаций \ 45
2. Синтез логических формул для схемы выработки переноса \ 47
3. Схема замещения \ 49

2. Алгебраические сумматоры М-кодов с групповым переносом \ 52

1. Введение \ 52
2. Инверсный сумматор с нечетным групповым переносом \ 53
3. Инверсный сумматор с четным групповым переносом \ 55
4. Инвертор с нечетным групповым переносом \ 56

5. Инвертор с четным групповым переносом \ 57

Глава 5. Кодирование и декодирование \ 1-59

1. Устройства кодирования и декодирования действительных чисел \ 59

1. Прекодер Р-кода в М-код – PreCoder \ 59
2. Кодер положительного Р-кода в М-код – CoderPM \ 59
3. Кодер Р-кода в М-код с добавлением – ExpCoder \ 61
4. Декодер М-кода в Р-код – DecoderMP \ 62
5. Полный декодер М-кода в Р-код – mDecoderMP \ 64
6. Распределитель частей кода – Partitioning \ 64

2. Операции кодирования и декодирования комплексных чисел \ 66

0. Введение \ 66
1. Кодирование пары целых Р-кодов в целый С-код \ 66
2. Декодирование целого С-кода в пару целых Р-кодов \ 67
3. Кодирование пары Р-кодов в пару С-кодов \ 69
4. Декодирование С-кода в пару Р-кодов \ 70
5. Кодирование пары Р-кодов в С-код \ 71
7. Выделение целой и дробной части С-кода \ 71
8. Выделение целой части С-кода \ 73
9. Выделение дробной части С-кода \ 73
10. Преобразование экспоненты в целый С-код \ 73
11. Преобразование экспоненты в С-код \ 74
12. Преобразование целого С-кода в экспоненту \ 74
13. Преобразование С-кода в целый С-код \ 74
14. Преобразование целого С-кода в С-код \ 74

Глава 6. Специальные логические операции \ 1-75

1. Нормализация \ 76

2. Выравнивание экспонент \ 79

3. Сдвигатели \ 81

1. Сдвигатель мантиссы вправо – ShiftRigth \ 81
2. Сдвигатель мантиссы влево – ShiftLeft \ 83
3. Сдвигатель мантиссы вправо – ShFastR \ 83
4. Сдвигатель мантиссы влево – ShFastL \ 84

4. Компараторы \ 87

1. Блок сравнения С-кодов по модулю – CompModLfull \ 87
2. Блок сравнения М-кодов по длине – CompLen \ 87
3. Операции сравнения \ 89
 1. Сравнение по длине – CompareLen \ 89

2. Сравнение по модулю – CompareMod \ 89
3. Сравнение по длине кодов действительных и мнимых частей – CompareLenReIm \ 89
4. Сравнение по модулю действительных и мнимых частей – CompareModReIm \ 89
5. Сравнение по длине комплексных кодов с плавающей точкой – CompareModFloat \ 90
6. Сравнение экспонент – CompareExp \ 90

Глава 7. Умножение \ 1-91

1. Метод умножения М-кодов \ 91

2. Матричные умножители \ 93

1. Инверсный матричный умножитель М-кодов – MultMatrShort \ 93
2. Матричный квадратор М-кодов – QuadrMatr \ 96
3. Матричный умножитель М-кодов – MultMatr \ 96
4. Матричный умножитель С-кодов – FloatMultMatr \ 96
5. Матричный квадратор С-кодов – CalcModul \ 97

3. Сложение группы кодов \ 98

4. Формирование группы слагаемых при умножении \ 103

5. Составные умножители \ 1098

1. Умножитель М-кодов с деревом Уоллиса \ 108
2. Умножитель М-кодов с прекомпрессором \ 109
3. Умножитель М-кодов на квадраторах \ 110
4. Умножитель С-кодов на квадраторах \ 110

6. Операции умножения М-кодов и С-кодов \ 112

1. Умножение комплексных чисел \ 115
2. Квадрат комплексного числа \ 115
3. Квадрат модуля комплексного числа \ 115
4. Центроафинное преобразование \ 115
7. Скалярное умножение комплексных чисел \ 115
11. Умножение целых комплексных чисел \ 116
12. Квадрат целого комплексного числа \ 116
13. Квадрат модуля целого комплексного числа \ 116
14. Центроафинное преобразование целого комплексного числа \ 116
15. Афинное преобразование целого комплексного числа \ 117
16. Быстрое преобразование Фурье целых комплексных чисел \ 117
21. Умножение действительных чисел \ 117
22. Квадрат действительного числа \ 118

- 23. Детерминант квадратной матрицы действительных чисел \ 118
- 24. Умножение матрицы на вектор для действительных чисел \ 118
- 31. Умножение целых действительных чисел \ 119
- 32. Квадрат целого действительного числа \ 119
- 35. Умножение четырех пар целых действительных чисел \ 119

Глава 8. Деление \ 1-20

- 1. Метод деления \ 120
- 2. Декомпозиции и композиции \ 122
- 3. Алгоритм обращения и деления \ 124

Часть 2. Коды комплексных чисел и векторов

Предисловие \ 2-4

Глава 1. Позиционные коды комплексных чисел и векторов \ 2-8

- 1. О методе позиционного кодирования \ 8
- 2. Два способа синтеза кодов комплексных чисел \ 10
- 3. Метод кодирования точек многомерного пространства \ 12
- 4. Арифметические системы кодирования \ 18
- 5. Коды действительных чисел \ 21
- 6. Коды комплексных чисел \ 23
- 7. Коды многомерных векторов \ 32

Глава 2. Точность кодирования \ 2-36

- 1. Область представимых чисел \ 36
- 2. Область представимых модулей \ 44
- 3. Погрешности конечных кодов \ 51
- 4. Коды с плавающей точкой \ 54

Глава 3. Поразрядные арифметические операции \ 2-59

- 1. Поразрядные операции \ 59
- 2. Первый алгоритм поразрядных операций \ 61
- 3. Второй алгоритм поразрядных операций \ 68

4. Поразрядные операции при отрицательном основании \ 79

Глава 4. Алгоритмы кодирования и декодирования комплексных чисел \ 2-81

Глава 5. Умножение \ 2-85

1. Специальная алгебра в векторном пространстве \ 85

- 1.1. Алгебра в трехмерном векторном пространстве \ 85
- 1.2. Покомпонентное умножение \ 86
- 1.3. Векторное произведение \ 86
- 1.4. Скалярное произведение \ 86
- 1.5. Поворот вектора \ 87
- 1.6. Центроаффинное преобразование \ 88
- 1.7. Многомерное пространство \ 88

2. Умножение многомерных векторов \ 89

- 2.1. Метод умножения комплексных чисел и многомерных векторов \ 81
- 2.2. Умножение векторов по основанию (1.31) \ 82
- 2.3. Умножение векторов по основанию (1.30) \ 82
- 2.4. Последовательное и матричное умножение \ 83
- 2.5. Умножение целых кодов векторов по основанию (1.31) \ 86
- 2.6. Умножение целых кодов векторов по основанию (1.30) \ 86
- 2.7. Покомпонентное умножение многомерных векторов \ 87

3. Скалярное и векторное умножения \ 97

- 3.1. Скалярное произведение \ 97
- 3.2. Векторное произведение \ 98
- 3.3. Переносы при скалярном умножении \ 99
- 3.4. Переносы при векторном умножении \ 101

Глава 6. Метод «цифра за цифрой» \ 2-103

1. Введение \ 103

2. Декомпозиции \ 105

- 1. Вступление \ 105
- 2. Алгоритм декомпозиции \ 106
- 3. Варианты декомпозиций \ 109

3. Композиции \ 114

4. Двухшаговые операции \ 118

- 1. Введение \ 118
- 2. Алгоритм вычисления функций \ 118

- 3. Об аппаратной реализации \ 119
- 5. Схемы и операции с биномами \ 122**
 - 1 Схемы умножения на бином \ 122
 - 2. Операции с биномами \ 125
- 6. Коллектор \ 126**
 - 1. Устройство коллектора \ 126
 - 2. Блок поиска триады \ 128
 - 3. Логический блок \ 129
 - 4. Ускоренный логический блок \ 130
 - 5. Трехразрядная логическая схема \ 131
 - 6. Одноразрядная логическая схема \ 132
 - 7. Операции с коллектором \ 133
 - 1. Чтение N-разряда коллектора \ 133
 - 2. Обнуление N-разряда коллектора \ 133
 - 3. Добавление "1" в N-разряд коллектора \ 133
 - 4. Поиск в первого элемента коллекторе \ 133
 - 5. Поиск в следующего элемента коллекторе \ 133
 - 6. Установка максимума в счетчике коллектора \ 133

Глава 7. Деление \ 2-134

- 1. Метод деления \ 134**
- 2. Деление комплексных чисел по основанию ρ_2 \ 137**
- 3. Деление действительных чисел в системе с основанием «-2» \ 144**
- 4. Деление трехмерных векторов \ 147**

Глава 8. Вычисление функций комплексного переменного \ 2-155

- 1. Логарифмирование \ 155**
 - 1. Определение натурального логарифма комплексного числа (вариант 1) \ 155
 - 2. Вычисление логарифма модуля комплексного числа \ 156
 - 3. Определение натурального логарифма комплексного числа (вариант 2) \ 156
 - 4. Определение натурального логарифма положительного действительного числа \ 157
 - 5. Определение натурального логарифма отрицательного действительного числа \ 157
 - 6. Определение натурального логарифма действительного числа \ 157
- 2. Потенцирование \ 158**

1. Потенцирование комплексного числа \	158
2. Потенцирование действительного числа \	160
3. Операции с логарифмическими формами \	162
1. Логарифмическая форма представления комплексного числа \	162
2. Формирования логарифмической формы \	162
3. Возврат из логарифмической формы \	163
4. Алгебраическое сложение логарифмических форм \	165
5. Умножение логарифмической формы на целое число \	165
6. Переполнения \	165
4. Извлечение квадратного корня \	166
1. Извлечение квадратного корня из комплексного числа \	166
2. Извлечение квадратного корня из сопряженного числа \	166
3. Извлечение корня из положительного действительного числа \	166
5. Полярные координаты \	168
1. Вычисление модуля комплексного числа \	168
2. Вычисление аргумента комплексного числа (вариант 1) \	168
3. Вычисление аргумента комплексного числа (вариант 2) \	168
4. Вычисление полярных координат \	169
5. Возврат из полярных координат \	170
6. Вычисление синуса и косинуса действительного числа \	170
7. Определение полуквadrанта \	172
8. Сегментация перед извлечением корня \	172
9. Сегментация после извлечения корня \	173
10. Сегментация для логарифмирования \	174
6. Операции с полярными формами \	176
1. Полярная форма представления комплексного числа \	176
2. Умножение показательных форм \	176
3. Поворот показательной формы \	177
7. Сложные функции \	178
Глава 9. Решение уравнений \	181
1. Метод «цифра за цифрой» и трансцендентные уравнения /	181
2. Определение корней степенного полинома \	185
3. Дифференциальные уравнения \	189
4. Решение квадратных уравнений \	190

- 5. Решение трансцендентных уравнений методом
Мюллера \ 192

Глава 10. Моделирование устройства извлечения квадратного корня из КОМПЛЕКСНЫХ чисел \ 2-193

- 1. Введение \ 193
- 2. Извлечение квадратного корня \ 194
 - 1. Композиции \ 194
 - 2. Декомпозиции \ 196
 - 3. Область представления чисел \ 200
 - 4. Оценка быстродействия \ 205
 - 5. Оптимизация по быстродействию \ 206
- 3. Sqrt – нормализация \ 210
- 4. О сходимости \ 215
- 5. Некоторые сравнения \ 217

Глава 11. Моделирование устройства для потенцирования и логарифмирования КОМПЛЕКСНЫХ чисел \ 2-220

- 1. Введение \ 220
- 2. Композиции \ 220
- 3. Декомпозиции \ 225
- 4. Область представимых чисел \ 229
- 5. Оценка быстродействия \ 236
- 6. Оптимизация по быстродействию \ 237
- 7. Дополнительные функции устройства \ 240
 - 1. Вычисление орта комплексного числа \ 240
 - 2. Вычисление логарифма модуля и аргумента комплексного
числа \ 241
 - 3. Вычисление корня квадратного и логарифма комплексного
числа \ 241

Глава 12. Моделирование устройства для вычисления одного корня степенного полинома \ 2-242

- 1. Композиции \ 242
- 2. Декомпозиции для решения нормального квадратного
трехчлена \ 245

3. Root – нормализация при вычислении одного корня
степенного полинома \ 252
 1. Введение \ 252
 2. Root-нормализация по углу \ 253
 3. Root-нормализация по модулю \ 253
 4. Полная root-нормализация \ 254

Часть 3. Коды геометрических фигур

Предисловие \ 3-4

Глава 1. Введение \ 3-6

Глава 2. Прототипы \ 3-10

1. Представление данных \ 10
2. Простейшее арифметическое устройство \ 11
3. Арифметическое устройство с прямоугольными кодами \ 13

Глава 3. Арифметика комплексных чисел и векторов \ 3-16

1. Вступление \ 16
2. Умножение многомерных векторов \ 16
 1. Метод умножения многомерных и векторов \ 16
 2. Умножение на базовую функцию для векторов по основанию (3.3.10) \ 17
 3. Умножение на базовую функцию для векторов по основанию (3.3.7) \ 17
 4. Умножение целых кодов векторов по основанию (3.3.10) \ 18
 5. Умножение целых кодов векторов по основанию (3.3.7) \ 19
 6. Покомпонентное умножение многомерных векторов \ 20
 7. Скалярное и векторное умножения \ 21
3. Алгоритмы и устройства для кодирования и декодирования многомерных векторов \ 21
 1. Кодирование комплексного числа в системе 1 \ 21
 2. Декодирование комплексного числа в системе 1 \ 22
 3. Кодирование комплексного числа в системе 2 \ 22
 4. Декодирование комплексного числа в системе 2 \ 23

Глава 4. Векторный процессор \ 3-24

1. Представление данных и векторное арифметическое устройство \ 24

2. Сравнения \ 28

Глава 5. Теория кодирования фигур \ 3-31

1. Первичные геометрические коды \ 31

1. Структура данных \ 31
2. Арифметические операции с геометрическими кодами по действительному основанию \ 34
 1. Общие положения \ 34
 2. Запись базисного кода \ 35
 3. Транспонирование \ 36
 4. Сложение геометрического и базисного кодов по основанию (2) \ 36
 5. Алгебраическое сложение геометрического и базисного кодов по основанию (2) \ 39
 6. Алгебраическое сложение геометрического и базисного кодов по основанию (-2) \ 39
 7. Умножение геометрического и базисного кодов \ 42
 8. Деление геометрического кода на базисный код \ 47
 9. Округление геометрического кода \ 47
3. Геометрические коды по комплексному основанию \ 47
 1. Алгебраическое сложение геометрического и базисного кодов \ 48
 2. Умножение геометрического и базисного кодов \ 49
4. Кодирование и преобразование плоских фигур \ 53
 1. Метод кодирования \ 53
 2. Перенос \ 58
 3. Центроаффинное преобразование \ 58
 4. Аффинное преобразование \ 58
5. Кодирование и преобразование пространственных фигур \ 59

2. Атрибутные геометрические коды \ 61

1. Структура данных \ 61
2. AGC по действительному основанию \ 65
 1. Запись данного номера \ 65
 2. Запись и поиск данного значения \ 65
 3. Чтение значения пути с данным номером \ 66
 4. Сложение AGC с базисным кодом по основанию (2) \ 66
 5. Обратное сложение AGC с базисным кодом по основанию (-2) \ 68
 6. Инвертирование AGC по основанию (-2) \ 70
 7. Алгебраическое сложение AGC \ 71

8. Поиск следующего открытого пути, его номера и его значения \ 71
9. Умножение AGC на базисный код \ 71
3. Атрибутные геометрические коды по комплексному основанию \ 72
 1. Обратное сложение AGCC с базисным кодом \ 73
 2. Инвертирование \ 73
 3. Центроаффинное преобразование \ 73
4. Атрибутные геометрические коды пространственных фигур \ 78
5. Сокращенные атрибутные геометрические коды \ 81

Глава 6. Геометрический процессор \ 3-83

0. Представление данных \ 83
 1. Полная специализированная оперативная память \ 86
 2. Фрагментарная специализированная оперативная память \ 87
 3. Максимальное арифметическое устройство геометрических фигур \ 90
 4. Фрагментарное арифметическое устройство геометрических фигур \ 91
 5. Процессор с максимальным арифметическим устройством \ 93
 6. Процессор с фрагментарным арифметическим устройством \ 95
 7. Основные процедуры \ 98
 1. Аффинное преобразование \ 98
 2. Округление \ 99
 3. Грубое округление \ 100
 4. Коррекция атрибутов \ 101
 5. Вычисление атрибутов \ 101
 6. Кодирование фигуры \ 102
 7. Декодирование фигуры \ 102
 8. Операционные блоки \ 102
 1. Блок записи номера с данным кодом \ 103
 2. Блок записи значения с данным кодом \ 103
 3. Блок чтения значения пути с данным номером \ 104
 4. Обратный сумматор \ 105
 5. Блок поиска первого открытого пути \ 106
 6. Блок чтения номера и значения пути с данной терминальной вершиной \ 107

7. Блок поиска следующей терминальной вершины \ 108

Глава 7. Сравнительный анализ \ 3-109

Обозначения \ 3-115

Список примеров \ 3-118

Список таблиц \ 3-119

Список рисунков \ 3-121

Часть 4. Коды функций \ 4-1

Предисловие \ 4-4

Глава 1. Позиционные коды функций \ 4-6

1. Треугольные коды \ 6

2. Алгебраическое сложение кодов вещественных чисел \ 10

3. Алгебраическое сложение треугольных кодов \ 12

4. Деление треугольных кодов на параметр \ 14

5. Умножение треугольных кодов \ 16

6. Кодирование и декодирование треугольных кодов \ 17

7. Дифференцирование треугольных кодов \ 19

8. Ступенчатые коды \ 21

Глава 2. Кодирование тригонометрических рядов \ 4-23

1. Треугольные коды функций по основанию $\sin^2(x)$ \ 23

2. Тригонометрические треугольные коды-ТТК \ 36

3. Операции с ТТК \ 40

1. Короткие операции \ 40

2. Умножение \ 40

3. Дифференцирование \ 41

4. Интегрирование \ 43

5. Инвертирование аргумента \ 44

6. Смещение оси ординат \ 45

4. Кодирование и декодирование ТТК \ 46

5. Погрешность кодирования ТТК \ 48

6. Укорочение ТТК \ 52

7. Гиперболические треугольные коды \ 55

Глава 3. Кодирование функций многих аргументов \ 4-56

1. Пирамидальные коды \ 56
2. Гиперпирамидальные коды \ 59

Глава 4. Четверичные тригонометрические треугольные коды \ 4-62

1. Арифметические операции с четверичными треугольными кодами \ 64
 1. Алгебраическое сложение \ 64
 2. Деление на параметр \ 66
 3. Умножение \ 70
2. Кодирование и декодирование четверичных треугольных кодов \ 71
 1. Кодирование и декодирование четверичных треугольных кодов \ 71
 2. Кодирование и декодирование четверичных тригонометрических треугольных кодов \ 76
3. Математические операции с четверичными тригонометрическими треугольными кодами \ 85
4. Укорочение четверичных тригонометрических треугольных кодов \ 86
5. Погрешность кодирования четверичных тригонометрических треугольных кодов \ 91

Глава 5. Арифметическое устройство для операций с функциями \ 4-95

1. Одноразрядные схемы \ 95
 1. Одноразрядный сумматор \ 95
 2. Одноразрядный вычитатель \ 96
 3. Одноразрядный инвертор \ 97
 4. Одноразрядный учетверитель \ 98
 5. Одноразрядный делитель на 4 \ 98
2. Многоразрядные схемы \ 99
 1. Столбцовый сумматор \ 99
 2. Строчный сумматор \ 100
 3. Столбцовый делитель \ 101
 4. Строчный делитель \ 102
 5. Параллельный сумматор \ 102
 6. Параллельный делитель \ 103

3. Вариант арифметического устройства \ 104

1. Структура арифметического устройства \ 104
2. Операции с треугольными кодами \ 106
 1. Алгебраическое сложение смешанных кодов \ 106
 2. Учетверение смешанного кода \ 106
 3. Деление треугольного кода на 4 \ 106
 5. Округление треугольного кода \ 106
 6. Округление треугольного кода \ 107
 7. Преобразование прямоугольного кода в треугольный код \ 107
 8. Преобразование треугольного кода в прямоугольный код \ 108
 9. Умножение треугольных кодов \ 108
3. Операции с тригонометрическими треугольными кодами \ 109
 1. Алгебраическое сложение ТТК \ 109
 2. Умножение ТТК \ 109
 3. Дифференцирование ТТК \ 109
 4. Интегрирование ТТК \ 110.
 5. Кодирование тригонометрического ряда \ 110
 6. Декодирование ТТК \ 111
 7. Укорочение ТТК \ 111

4. Сравнительный анализ \ 111

1. Взаимосвязь между разрядностью ТТК, рангом ряда и разрядностью коэффициентов ряда \ 112
2. Разрядность \ 113
3. Объем арифметического устройства \ 113
4. Длительность элементарных операций \ 114
5. Взаимосвязь между элементарными операциями и операциями с функциями \ 115
6. Выводы \ 116

Обозначения \ 4-118

Предисловие

Кратко остановимся на истории вопроса. Компьютерная арифметика сложных математических объектов берет свое начало в статье Шеннона о позиционном кодировании действительных чисел по отрицательному основанию [1]. Эта идея, по-видимому, впервые была реализована в Польше [2] и побудила (по-видимому) нескольких авторов к разработке методов кодирования комплексных чисел. Практически одновременно Кнут [3] предложил систему кодирования по основанию $j\sqrt{2}$. Хмельник [4] предложил несколько систем, в т.ч. по основаниям $j\sqrt{2}$ и $(-1+j)$. Основание $(-1+j)$ позднее рассмотрел Penney [5]. Хмельник в диссертации [6] рассмотрел комплекс вопросов конструирования арифметического устройства для операций с комплексными числами. Эти результаты развивались затем в работах [7, 8, 9, 11, 12, 13, 14, 33, 34, 35, 44].

В нескольких работах [16, 17, 18] рассмотрены методы построения умножителей комплексных чисел. При этом основное внимание уделяется способам реализации этих устройств на чипе. Для этого предлагаются избыточные системы кодирования, которые, по мнению авторов, позволяют построить более регулярные схемы. Однако при этом не рассматриваются другие операции с предлагаемыми кодами (например, деление).

Для кодов действительных чисел известен метод ‘цифра за цифрой’ [19, 20] для аппаратного вычисления элементарных функций. Он может быть обобщен на позиционные коды комплексных чисел, что впервые было сделано Хмельником в [6, 11]. При этом часто достаточно иметь аппаратную реализацию только потенцирования и логарифмирования, поскольку через эти функции в комплексной области можно выразить все элементарные функции. Кроме того, этот метод применим для построения алгоритмов аппаратного решения трансцендентных уравнений и систем таких уравнений. При использовании кодов комплексных (а не действительных) чисел класс таких уравнений расширяется, а алгоритмы их решения существенно упрощаются. В [6, 11] описан один из таких алгоритмов.

Дальнейшее развитие идеи позиционного кодирования шло по пути построения позиционных кодов векторов [21, 22], матриц [36,

37], функций [23, 24, 25, 33, 47], геометрических фигур [22, 26, 27, 28, 32, 38, 39, 46]. Следует отметить, что коды геометрических фигур могут рассматриваться как коды числовых массивов и для них могут быть построены эффективные поисковые алгоритмы [29, 30, 31]. Многие из этих результатов обобщены в книге [32].

Предпочтение, отдаваемое именно позиционным кодам, объясняется, главным образом, тем, что с ними очень просто выполняются арифметические операции. Так, вне зависимости от объекта кодирования, сложение позиционных кодов связано с распространением переносов от младших разрядов к старшим, а умножение состоит из сдвигов (то-есть перенумераций разрядов) и сложений. Упомянутый выше метод ‘цифра за цифрой’ вообще применим только в сочетании с позиционной системой кодирования.

Важно отметить, что в программировании для предлагаемых компьютеров используется существующий математический аппарат, не учитывающий, естественно, специфических возможностей этих компьютеров. Можно надеяться, что при распространении таких компьютеров будут найдены не только другие методы решения задач, но и другие неожиданные области применения, как это непрерывно происходит с существующими компьютерами. Например, существует теория функций пространственного комплексного переменного [40]. Алгебра четырехмерных векторов [21, 32], предложенная для их кодирования, совпадает с алгеброй пространственных комплексных чисел, используемой в [40]. В связи с этим появляется возможность разработки компьютерной арифметики пространственных комплексных чисел с аппаратным вычислением функций этого переменного, как дальнейшим обобщением метода ‘цифра за цифрой’ (подобно тому, как это было сделано для комплексных чисел [6, 11]). В этом есть практический смысл, поскольку теория функций пространственного комплексного переменного может быть использована в весьма сложных задачах теоретической физики [40].

В предлагаемой книге можно обнаружить много аналогий с традиционной компьютерной арифметикой. Можно указать ряд книг, где подробно рассматривается эта арифметика [41, 42, 43].

Данная книга включает в себя книги [32, 45-48, 51, 52] и является их продолжением. В [53, 58] описывается проект арифметического устройства, в котором воплощены многие результаты предложенной теории. Этот проект представляет собой подробное описание VHDL-модели арифметического устройства,

выполняющего около 400 различных операций с двоичными кодами действительных чисел по отрицательному основанию и с двоичными кодами комплексных чисел.

Книга сопровождается диском CD. Этот диск содержит открытые коды программ системы MATLAB, описанных в примерах книги. Диск можно приобрести отдельно – см. [54]. При упоминании программ в тексте книги указываются ссылки на файлы диска, содержащие эти программы. Они имеют следующий вид: CD/catalog/file. Для понимания теории диск необязателен, тем более, что некоторые программы приведены в тексте книги.

Книга ориентирована на пользователя, который намерен применять описываемую компьютерную арифметику в собственных разработках специализированных процессоров. С этой целью книга включает всю информацию, необходимую для

- понимания функционирования процессора для операций с функциями во всех деталях,
- использования приведенных в книге технических решений для собственной разработки.

Другими словами, книга включает

- Теорию кодирования
- Алгоритмы операций
- Примеры кодирования, декодирования и вычислений
- Описание нескольких вариантов процессоров
- Системы команд для них
- Схемы операционных блоков
- Сравнительный анализ

Литература

1. Shannon C. E., **A Symmetrical Notation of Number**, Amer. Math. Month., 57, 1950, 90.
2. Шевчик Ю., **Универсальная цифровая машина УМЦ-10**, сб. "Цифровая вычислительная техника и программирование", выпуск 2, "Советское радио", 1967.
3. Knuth D. E., **An Imaginary Number System**, Communication of the ACM-3, 1960, № 4.
4. Хмельник С. И., **Специализированная ЦВМ для операций с комплексными числами**. Вопросы радиоэлектроники, серия XII, выпуск 2, 1964 (*это – исторически первая в мире статья по кодированию комплексных чисел: указано, что поступила в редакцию в марте 1962 г.; статья приведена в приложении*).
5. Penney W., **A 'Binary' System for Complex Numbers**, Journal of ACM 12, No. 2, 1965, pp. 247-248.
6. Хмельник С. И., **Методы синтеза ЦВМ, оперирующих с комплексными числами, и их использование для решения некоторых задач ПВО**, Москва, ГКРЭ при СМ СССР, ГСНИИ-5, №7907, 1964.
7. Хмельник С. И., **Сумматор кодов комплексных чисел**. Вопросы радиоэлектроники, серия XII, выпуск 3, 1965.
8. Хмельник С. И., **Позиционное кодирование комплексных чисел**. Вопросы радиоэлектроники, серия XII, выпуск 9, 1966.
9. Хмельник С. И., **Системы счисления с комплексными основаниями**. В книге [12].
10. Поспелов Д. А., **Арифметические основы вычислительных машин дискретного действия**, изд. "Высшая школа", 1970.
11. Хмельник С. И., **Решение навигационных задач на ЦВМ, оперирующих с кодами комплексных чисел**. Вопросы специальной радиоэлектроники, серия "Телемеханика и системы управления", выпуск 6, 1971.

12. Хмельник С. И., **Арифметическое устройство для цифровой вычислительной машины.** Авт.св. 266362, 1970, БИ-11.
13. Хмельник С. И., **Арифметическое устройство для операций с комплексными числами** Авт. св. 377769, 1973, БИ-18.
14. Хмельник С. И., Свистунов Ю. А, **Узлы процессоров цифровых устройств**, в книге [15].
15. **Цифровые устройства на микросхемах.** Изд. "Энергия", 1975.
16. T. Aoki, H. Amada, and T. Higuchi: **Real/Complex Reconfigurable Arithmetic Using Redundant Complex Number Systems.** In Proc. 13th Symposium on Computer Arithmetic, 1997.
17. Y. Chang and K. Parhi. **High Performance Digit Serial Complex Number Multiplier-Accumulator**, Proc. Int. Conf. on Computer Design, 1998.
18. A.M. Nielsen and P. Kornerup, **Redundant Radix Representation of Ring**, IEEE Transactions on Computers, Vol. 48 (11), November 1999
19. В.Д. Байков, В.Б. Смолов, **Специализированные процессоры: итерационные алгоритмы и структуры**, "Радио и связь", М., 1985.
20. Miller J.M., **Elementary Functions. Algorithms and Implementation.** Birkhauser, 1997, Boston.
21. Хмельник С.И., **Кодирование векторов.** Кибернетика, АН УССР, 1969, №5.
22. Хмельник С.И., **Алгебра многомерных векторов и кодирование пространственных фигур.** Автоматика и вычислительная техника, АН АССР, 1971, №1.
23. Хмельник С.И., **Кодирование функций.** Кибернетика, АН УССР, 1966, №4.
24. Хмельник С.И., **Несколько типов позиционных кодов функций.** Кибернетика, АН УССР, 1970, №5.
25. Хмельник С.И., **Алгоритмы кодирования и декодирования функциональных рядов.** Сб. "Цифровая вычислительная техника и программирование", выпуск 8, 1974.
26. Хмельник С.И., **Кодирование плоских фигур.** Автоматика и вычислительная техника, АН АССР, 1970, №6.

27. Хмельник С.И., **Цифровое устройство для геометрических преобразований изображения.** Авт. св. 333573, 1972, БИ-11
28. Хмельник С.И., **Устройство для геометрических преобразований изображения.** Авт. св. 1030816, БИ-27, 1983.
29. Хмельник С.И., **Быстродействующие поисковые процедуры.** Третий международный симпозиум по теории информации, часть II, Таллин, 1973.
30. Хмельник С.И., **Многокритериальная задача о назначениях.** Изв.АН СССР, Техническая кибернетика, №4, 1977.
31. Хмельник С.И., **Поисковые процедуры с геометрическими кодами,** ж. "Кибернетика", АН УССР, 1990, №6.
32. Хмельник С. **Компьютерная арифметика векторов, фигур и функций,** изд. «Mathematics in Computers», Москва – Тель-Авив, 1995.
33. Khmelnik S. **A method and system for implementing a coprocessor.** Canadian Intellectual Property Office. Application No 2,293,953. Priority 05.01.00.
34. Khmelnik S. **A Method and System for Processing Complex Numbers.** International patent application under PCT. PCT/CA01/00007, WO 01 50332 A2. Priority 05.01.00.
35. Khmelnik S. **A Method and System for Processing Complex Numbers.** USA, United States Patent Application No. 10/189,195. Priority 05.06.02.
36. Khmelnik S. **Method and System for Processing Matrices of Complex Numbers.** Canadian Intellectual Property Office. Application No 2,339,919. Priority 07.03.01.
37. Khmelnik S. **Method and System for Processing Matrices of Complex Numbers and Complex Fast Fourier Transformations.** International patent application under PCT. PCT/CA02/00295, WO 02 071254 A2. Priority 07.03.01.
38. Khmelnik S. **Method and System for Processing Geometrical Figures.** Canadian Intellectual Property Office. Application No 2,349,924. Priority 07.07.01.
39. Khmelnik S. **Method and System for Processing Geometrical Figures.** International patent application under PCT. PCT/CA02/00835, WO 02 099625 A2. Priority 07.06.01.

-
40. В.И. Елисеев. **Введение в методы теории функций пространственного комплексного переменного**, Центр научно-технического творчества молодежи «Алгоритм». Москва, НИАТ, 1990. Шифр Д7-90/83308 в каталоге Государственной публичной научно-технической библиотеки, <http://www.maths.ru/>
 41. L. Wanhammar. **DSP Integrated Circuits**, Academic Press, 1999.
 42. B. Parhami. **Computer Arithmetic. Algorithms and Hardware Design**, Oxford University Press, 2000.
 43. M.J. Flynn, S.F. Oberman. **Advanced Computer Arithmetic Design**, A wiley interscience Publication John Wiley & Sons, Inc., 2001.
 44. Khmelnik S. **A Method and System for Processing Complex Numbers**. European Patent Office, EP 1248993. Priority 12.07.01.
 45. Хмельник С.И., Дубсон И.С., Хмельник С.М., Видуецкий А.Е. **Арифметическое устройство кодов по отрицательному основанию**, изд. «Mathematics in Computers», Израиль, 2004 и 2007, <http://www.lulu.com/content/93352 и 564868>.
 46. Хмельник С.И. **Компьютерная арифметика геометрических фигур. Алгоритмы и аппаратура**, изд. «Mathematics in Computers», Израиль, 2004, <http://www.lulu.com/content/71555>.
 47. Хмельник С.И. **Компьютерная арифметика функций. Алгоритмы и аппаратура**, изд. «Mathematics in Computers», Израиль, 2004, <http://www.lulu.com/content/84116>.
 48. Хмельник С.И. **Кодирование комплексных чисел и векторов**. Изд. «Mathematics in Computers», Израиль, 2004, <http://www.lulu.com/content/94846>.
 49. Байков В.Д., Смолов В.Б., **Специализированные процессоры: итерационные алгоритмы и структуры**, “Радио и связь”, М., 1985.
 50. Muller J.M., **Elementary Functions. Algorithms and Implementation**. Birkhauser, 1997, Boston.
 51. I. Koren, **Computer arithmetic algorithms**, A K Peters LTD, 2002.
 52. Хмельник С.И. **Компьютерная арифметика комплексных чисел и векторов. Теория, аппаратура, моделирование**.
-

- Изд. «Mathematics in Computers», Израиль, 2007, <http://www.lulu.com/content/560836>.
53. Хмельник С.И. **Арифметическое устройство комплексных чисел.** Изд. «Mathematics in Computers», Израиль, 2007, <http://www.lulu.com/content/641090>.
54. Хмельник С.И., **Программы моделирования комплексного арифметического устройства,** изд. «Mathematics in Computers», Израиль, 2006, <http://www.lulu.com/content/585292>
55. Малиновский Б.Н., **История вычислительной техники в лицах.** Киев, Фирма "КИТ", ПТОО "АСК", 1995 г.
56. Sébastien Roy, Daniel Lefebvre and Henri H. Arsenault. **Recognition invariant under unknown affine transformations of intensity,** Optics Communications, Volume 238, 2004.
57. Shih-Hsuan Yang, Chun-Yen Liao, and Chin-Yun Hsieh. **Watermarking MPEG-4 2D Mesh Animation in Multiresolution Analysis,** Computer Science, Volume 2532, 2002.

Часть 1. Коды действительных чисел по отрицательному основанию

Аннотация

В этой части описывается арифметическое устройство, которое оперирует двоичными кодами по основанию «-2», которые далее называются М-кодами. В отличие от известных публикаций о кодах по отрицательному основанию данная книга содержит подробное описание многочисленных схем и алгоритмов для операций с этими кодами. Приводятся ранее неизвестные быстродействующие методы и схемы сложения и умножения кодов по отрицательному основанию, в которых используются специфические свойства этих кодов.

Кроме того, описываются некоторые схемы и операции с кодами комплексных чисел, поскольку некоторая композиция кодов по отрицательному основанию может рассматриваться как единый код комплексного числа.

Описание дается в таком виде, чтобы заинтересованный читатель мог воспроизвести АУ полностью. Описываемое устройство выполняет кодирование традиционных кодов, представленных в формате IEEE-754, в М-коды, декодирование М-кодов в коды IEEE-754, алгебраическое сложение, умножение и деление М-кодов. Предусмотрены форматы М-кодов с фиксированной и плавающей точками, формат парных М-кодов также с фиксированной и плавающей точками. Для парных М-кодов в устройстве предусмотрены операции, эквивалентные алгебраическому сложению и умножению комплексных чисел, операция «бабочка» для FFT-преобразований и т.д.

Содержание

Предисловие \ 1-4

Глава 1. Представление чисел \ 1-6

Глава 2. Алгебраическое сложение М-кодов \ 1-11

Глава 3. Алгебраическое сложение С-кодов \ 1-33

Глава 4. Групповой перенос \ 1-43

Глава 5. Кодирование и декодирование \ 1-59

Глава 6. Специальные логические операции \ 1-75

Глава 7. Умножение \ 91

Глава 8. Деление \ 120

Предисловие

Широко известны и распространены двоичные позиционные коды действительных чисел по основанию 2. Однако такими кодами нельзя изобразить отрицательные действительные числа, для представления которых приходится применять искусственные приемы, в частности, использовать обратные и дополнительные коды, что вызывает ряд неудобств.

Существуют также двоичные позиционные коды действительных чисел по основанию «-2», предложенные Шенноном [1]. Идея такого кодирования, по-видимому, впервые была реализована в Польше [2].

Система кодирования действительных чисел по отрицательному основанию имеет ряд достоинств по сравнению с традиционной системой кодирования по положительному основанию. Эти достоинства являются следствием того, что положительные и отрицательные числа при отрицательном основании представляются единообразно, и заключаются в следующем:

- не требуется выполнять преобразования из прямого кода в обратный (или дополнительный) и обратно,
- отсутствует знаковый разряд и исключаются операции со знаковыми разрядами,
- упрощаются правила определения переполнения при алгебраическом сложении,
- упрощаются алгоритмы выполнения операций с кодами переменной длины.

Вместе с тем алгоритмы арифметических операций при отрицательном основании столь же просты, как и при положительном основании.

В настоящее время теория кодов по отрицательному основанию описывается во многих работах - см., например, [51]. В отличие от известных публикаций о кодах по отрицательному основанию данная книга содержит подробное описание многочисленных схем и алгоритмов для операций с этими кодами. Приводятся ранее неизвестные быстродействующие методы и схемы сложения и умножения кодов по отрицательному основанию, в которых используются специфические свойства этих кодов. Кроме того, описываются некоторые схемы и операции с кодами комплексных чисел, поскольку некоторая композиция кодов по отрицательному основанию может рассматриваться как единый код комплексного

числа. Такие коды и операции с ними рассмотрены в работах [34, 35, 44].

Итак, в этой части книги описывается арифметическое устройство, которое оперирует двоичными кодами по основанию «-2», которые далее называются М-кодами. Устройство выполняет кодирование традиционных кодов, представленных в формате IEEE-754, в М-коды, декодирование М-кодов в коды IEEE-754, алгебраическое сложение, умножение и деление М-кодов. Предусмотрены форматы М-кодов с фиксированной и плавающей точками, формат парных М-кодов также с фиксированной и плавающей точками. Для парных М-кодов в устройстве предусмотрены операции, эквивалентные алгебраическому сложению и умножению комплексных чисел, операция «бабочка» для FFT-преобразований и т.п.

Эта часть книги содержит 9 глав. В **главе 1** описывается представление действительных и комплексных чисел в виде двоичных кодов по основанию «-2» - т. н. М-кодов и С-кодов соответственно. В **главе 2** подробно рассматриваются многочисленные схемы алгебраического сложения М-кодов кодов. В **главе 3** описываются схемы алгебраического сложения С-кодов и операции, выполняемые на этих схемах. В **главе 4** предлагаются схемы группового переноса для ускорения алгебраического сложения. В **главе 5** рассматриваются схемы и операции кодирования и декодирования этих кодов. В **главе 6** рассматриваются нормализация, выравнивание экспонент, сдвигатели, компараторы М- и С-кодов. В **главе 7** описываются устройства для умножения и несколько операций умножения. Тут приводятся ранее неизвестные быстродействующие методы и схемы умножения кодов по отрицательному основанию, в которых используются специфические свойства этих кодов. В **главе 8** рассматривается метод и алгоритмы деления М-кодов.

Глава 1. Представление чисел

- 1. Вступление
- 2. Р-коды
- 3. М-коды
- 4. С-коды

1. Вступление

Далее будет рассмотрено несколько типов двоичных кодов определенной разрядности:

- Традиционный код по основанию «2» в формате IEEE-754 - так называемый Р-код,
- Код действительных чисел по основанию «-2» - так называемый М-код,
- Комплексный код, представляющий собой композицию М-кодов - так называемый С-код.

Для иллюстрации запишем несколько М-кодов и их числовых значений:

код	значение	код	значение
01.01	1.25	10.10	-2.5
01.00	1	10.00	-2
01.11	0.75	11.00	-1
01.10	0.5	00.10	-0.5
00.01	0.25	00.11	-0.25

Каждый код, как правило, состоит из кода порядка – ExpCode и кода мантиссы – MantCode. Код мантиссы в комплексном С-коде состоит из двух частей – кода реальной части - ReCode и кода мнимой части – ImCode. Разряды в коде нумеруются двояко в зависимости от контекста. Существует *сквозная нумерация* справа налево, где правый разряд имеет номер 0, и *смысловая нумерация*, где номер разряда соответствует степени основания.

2. Р-коды

На рис. 1 представлен Р-код с плавающей точкой в формате IEEE-754. Он представляет число $(-1)^{S_m} \cdot M \cdot 2^{(-1)^{S_e} E}$, где

S_m - знак мантииссы,

S_e - знак экспоненты,

$0 \leq M < 1$ - модуль мантииссы,

$0 \leq E \leq (2^7 - 1)$ - модуль экспоненты.

Величина $F = 2^{(-1)^{S_e} E}$ всегда положительна и находится в пределах $(2^{-127}) \leq F \leq (2^{127})$. Модуль нормализованной мантииссы находится в пределах $0.5 \leq M < 1$. Относительная ошибка представления нормализованной мантииссы равна $\varepsilon = 2^{-24}$.

31	30	29	28	24	23	22	2	1	0
Sm	Se	Modul of exponent				Modul of mantissa					

Рис. 1. Р-код с плавающей точкой.

На рис. 2 представлен Р-код с точкой, фиксированной после младшего 0-го разряда. Этот код представляет целое число $(-1)^{S_m} \cdot M$, где положительное число $0 \leq M \leq (2^{24} - 1)$

31	30	29	28	24	23	22	2	1	0
Sm	Nil					Mantissa					

Рис. 2. Р-код с фиксированной точкой.

Вообще, пределы изменения положительного действительного целого числа, представленного n -разрядным Р-кодом, таковы:

$$(0) \div (2^{n+1} - 1).$$

3. М-коды

На рис. 3 представлен М-код с плавающей точкой. Он представляет число $M \cdot (-2)^E$, где мантисса M и экспонента E являются кодами по основанию «-2». При этом

$$-\frac{2}{3} < M < \frac{1}{3}, -682 \leq E \leq 341.$$

Рассмотрим пределы изменения величины $F = (-2)^E$:

- при $F > 0$ имеем: $(2^{-682}) \leq F \leq (2^{340})$
- при $F < 0$ имеем: $(-2^{-681}) \leq F \leq (-2^{341})$

Нормализованная мантисса находится в пределах $-\frac{2}{3} < M < -\frac{1}{6}$.

Относительная ошибка представления нормализованной мантиссы равна $\varepsilon = \left(\frac{2^{-27}}{3} \right)$.

36	35	27	26	25	2	1	0
Exponent				Mantissa					

Рис. 3. М-код с плавающей точкой.

На рис. 4 представлен М-код с точкой, фиксированной после младшего 0-го разряда. Этот код представляет целое число $\left(\frac{-2^{27} + 2}{3} \right) \leq M \leq \left(\frac{2^{28} - 1}{3} \right)$.

36	35	27	26	25	2	1	0
Nil				Mantissa					

Рис. 4. М-код с фиксированной точкой.

Вообще, пределы изменения действительного целого числа, представленного n -разрядным М-кодом, таковы:

$$\left(\frac{-2^n + 2}{3} \right) \div \left(\frac{2^{n+1} - 1}{3} \right) \text{ if } n - \text{odd} .$$
$$\left(\frac{-2^{n+1} + 2}{3} \right) \div \left(\frac{2^n - 1}{3} \right) \text{ if } n - \text{even}$$

4. C-коды

C-код целого комплексного числа $Z=Re+j*Im$ образуется объединением M-кодов целых действительных чисел Re и Im . На рис. 5 представлен C-код с точкой, фиксированной после младшего 0-го разряда. В нем четные разряды 52, 50, ..., 2, 0 образуют целый код числа Re , а нечетные разряды 53, 51, ..., 3, 1 образуют целый код числа Im .

63	62	54	53	52	2	1	0
Nil				Mantissa					

Рис. 5. C-код с фиксированной точкой.

На рис. 6 представлен C-код с плавающей точкой для комплексного числа $Z=Re+j*Im$. В нем четные разряды 52, 50, ..., 2, 0 образуют код мантиссы числа M_{Re} - мантиссы числа Re , а нечетные разряды 53, 51, ..., 3, 1 образуют код мантиссы числа M_{Im} - мантиссы числа Im . Экспонента E является общей для чисел M_{Re} и M_{Im} . Таким образом, $Z = (-2)^E (M_{Re} + j \cdot M_{Im})$

63	62	54	53	52	2	1	0
Exponent				Mantissa					

Рис. 6. C-код с плавающей точкой.

Код нормализованной мантиссы M имеет вид

$$0.xu^{***}.....,$$

где по крайней мере один из разрядов x, u имеет значение «1». Величина соответствующей нормализованной части мантиссы (x относится к Im , u относится к Re) находится в пределах

$$-\frac{2}{3} < \bullet < -\frac{1}{6}.$$

Величина другой (a , следовательно, любой) части

мантиссы находится в пределах $-\frac{2}{3} < \bullet < \frac{1}{3}.$

В табл. 1 приведены ошибки представления комплексного числа C-кодом с плавающей точкой.

Таблица 1.

Абсолютная ошибка представления действительной или мнимой части мантиссы	$\varepsilon = \left(\frac{2^{-26}}{3} \right).$
Относительная ошибка представления нормализованной действительной или мнимой части мантиссы	$\varepsilon/2.$
Абсолютная ошибка представления модуля комплексного числа	$\mu = \varepsilon\sqrt{2}.$
Относительная ошибка представления нормализованной мантиссы	$\frac{\mu}{2} = \frac{\varepsilon}{\sqrt{2}} \approx 2^{-28}.$

В табл. 2 приведены для сравнения ошибки представления действительных чисел в традиционном Р-коде.

Таблица 2.

Абсолютная ошибка представления мантиссы	$\varepsilon = 2^{-24}.$
Относительная ошибка представления нормализованной мантиссы	$\varepsilon/2.$
Абсолютная ошибка представления модуля комплексного числа	$\mu = \varepsilon\sqrt{2}.$

Сравнивая относительные ошибки представления комплексных чисел в виде пары Р-кодов и в виде С-кодов, замечаем, что для достижения той же точности Р-коды должны были бы иметь разрядность 35. (27 разрядов в мантиссе и 8 разрядов в экспоненте). Кроме того, в предложенном представлении комплексных чисел диапазон изменения экспоненты увеличивается в 4 раза. Таким образом, для сохранения точности и диапазона 64-разрядных комплексных С-кодов традиционные Р-коды должны были бы иметь разрядность 37. Следовательно, представление комплексных чисел в виде С-кодов эквивалентно *сокращению разрядности в $37 \cdot 2 / 64 = 1.16$ раза, т.е. на 14%.*

Глава 2. Алгебраическое сложение М-кодов

1. Введение
2. Одноразрядные схемы алгебраического сложения
3. Многоразрядные схемы алгебраического сложения

1. Введение

Алгебраическое сложение связано с распространением переносов из младших разрядов в старшие. В операциях с М-кодами переносы могут принимать до четырех значений и поэтому одноразрядные схемы связываются двумя каналами переносов – см. рис. 1, где

a, b - одноразрядные входы,

$v1, v2$ – разряды двухразрядного входного переноса,

c - одноразрядный результат,

$w1, w2$ - разряды двухразрядного выходного переноса.

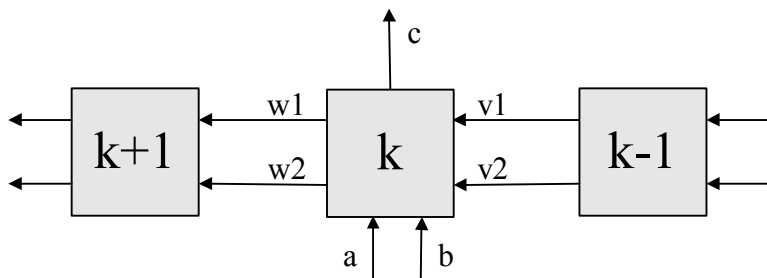


Рис. 1.

Рис. 2 иллюстрирует схему вычислений в k -разряде, где

V – число, изображаемое кодом ‘ $v1\ v2$ ’ по основанию (-2) ,
перенос из младшего разряда в k -ый разряд,

W – число, изображаемое кодом ‘ $w1\ w2$ ’ по основанию (-2) ,
перенос из k -го разряда в старший разряд,

$Q=a@b$ - результат операции $@$ над k -ми
разрядами – сложения, вычитания и др.,

$S=Q+V$ - разрядный результат.

Разрядный результат всегда можно представить в виде

$$S=c-2W.$$

(1)

При этом в определенной операции числа V и W принимают значения из одного и того же множества. На этом основано построение таблиц истинности для одноразрядных схем алгебраического сложения.

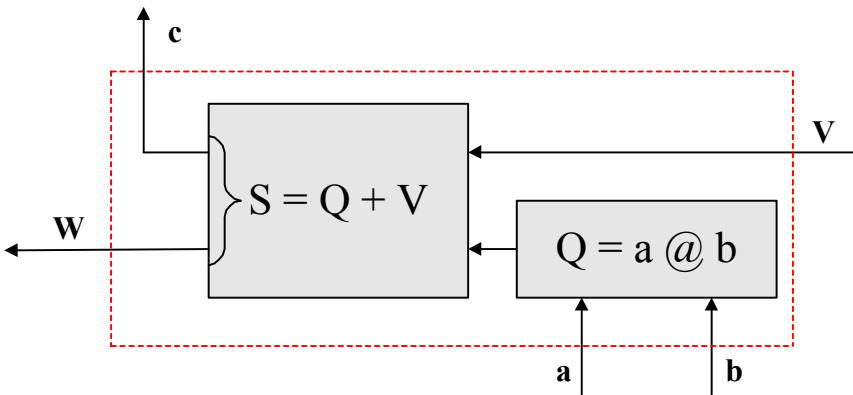


Рис. 2. Схема вычислений в k -разряде

Далее, кроме вышеуказанных приняты обозначения:

cin – одноразрядный входной перенос,

cout - одноразрядный выходной перенос,

cop - код операции (control of the operation).

2. Одноразрядные схемы алгебраического сложения

2.1. Инвертор. На рис. 1 представлена одноразрядная схема инвертирования Inv. Ее функционирование описывается таблицей истинности табл. 1. Эта таблица вычисляет величину $c - 2 * cout = -a$.

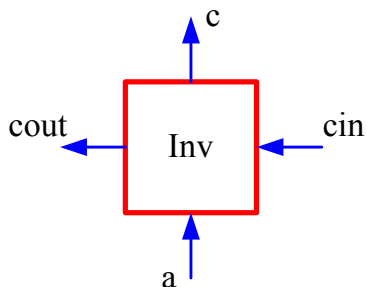


Рис. 1. Одноразрядная схема инвертирования.

Таблица 1 одноразрядной схемы инвертирования

a	cin	cout	c
0	0	0	0
1	0	1	1
0	1	0	1
1	1	0	0

2.2. Инвертор – удвоитель. На рис. 2 представлена одноразрядная схема инвертирования Inv2. Ее функционирование описывается таблицей истинности табл. 2. Эта таблица вычисляет величину $c - 2 * cout = cor * a$.

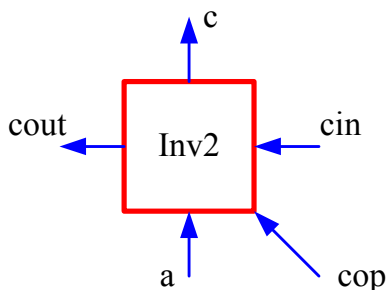


Рис. 2. Одноразрядная схема инвертора-удвоителя.

Таблица 2 одноразрядной схемы инвертирования-удвоения

сop			a	cin	cout	c
Передача «0»	0	00	*	*	0	0
Передача без изменения	1	01	0	*	0	0
	1	01	1	*	0	1
Инвертирование	-1	11	0	0	0	0
	-1	11	1	0	1	1
	-1	11	0	1	0	1
	-1	11	1	1	0	0
Сдвиг влево на 1 разряд	-2	10	0	0	0	0
	-2	10	1	0	1	0
	-2	10	0	1	0	1
	-2	10	1	1	1	1

2.3. Инверсный сумматор. На рис. 3 представлена одноразрядная схема инверсного сумматора InvAdd. Ее функционирование описывается таблицей истинности табл. 3. Эта таблица вычисляет сумму $c-2*cout = (-a - b + cin)$.

Таблица 3 одноразрядной схемы инверсного суммирования

a	b	cin	cout	c
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	0	1	0
0	0	1	0	1
0	1	1	0	0
1	0	1	0	0
1	1	1	1	1

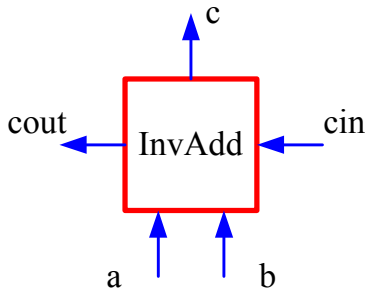


Рис. 3. Одноразрядная схема инверсного сумматора.

2.4. Сумматор. На рис. 4 представлена одноразрядная схема сумматора Add. Ее функционирование описывается таблицей истинности табл. 4. Эта таблица вычисляет сумму $c-2*w=(a+b+v)$.

Таблица 4 одноразрядной схемы суммирования

a	b	v1	v2	w1	w2	c
0	0	0	0	0	0	0
0	1	0	0	0	0	1
1	0	0	0	0	0	1
1	1	0	0	1	1	0
0	0	1	1	0	1	1
0	1	1	1	0	0	0
1	0	1	1	0	0	0
1	1	1	1	0	0	1
0	0	0	1	0	0	1
0	1	0	1	1	1	0
1	0	0	1	1	1	0
1	1	0	1	1	1	1

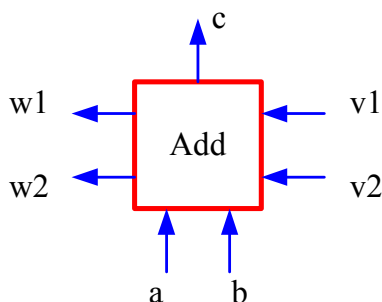


Рис. 4. Одноразрядная схема сумматора.

2.5. Вычитатель. На рис. 5 представлена одноразрядная схема сумматора Sub. Ее функционирование описывается таблицей истинности табл. 5. Эта таблица вычисляет сумму $c-2*w=(a-b+v)$.

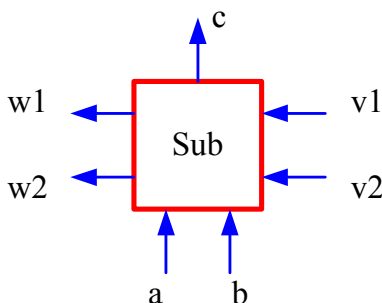


Рис. 5. Одноразрядная схема вычитателя.

Таблица 5 одноразрядной схемы вычитания

a	b	v1	v2	w1	w2	c
0	0	0	0	0	0	0
0	1	0	0	0	1	1
1	0	0	0	0	0	1
1	1	0	0	0	0	0
0	0	1	1	0	1	1
0	1	1	1	0	1	0
1	0	1	1	0	0	0
1	1	1	1	0	1	1
0	0	0	1	0	0	1
0	1	0	1	0	0	0
1	0	0	1	1	1	0
1	1	0	1	0	0	1

2.6. Алгебраический сумматор. На рис. 6 представлена одноразрядная схема алгебраического сумматора AddAlg. Ее функционирование описывается таблицей истинности табл. 6. Эта таблица вычисляет сумму $c-2*w = (C + v)$, где C определяется по указанным формулам.

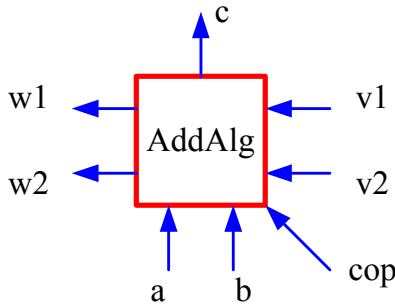


Рис. 6. Одноразрядная схема алгебраического сумматора.

Таблица 6 однократной схемы алгебраического суммирования

Операция	сop	a	b	v2	v1	w2	w1	c
C=A	00	0	0	0	0	0	0	0
		0	1	0	0	0	0	0
		1	0	0	0	0	0	1
		1	1	0	0	0	0	1
C=A+B	01	0	0	0	0	0	0	0
		0	1	0	0	0	0	1
		1	0	0	0	0	0	1
		1	1	0	0	1	1	0
		0	0	1	1	0	1	1
		0	1	1	1	0	0	0
		1	0	1	1	0	0	0
		1	1	1	1	0	0	1
		0	0	0	1	0	0	1
		0	1	0	1	1	1	0
		1	0	0	1	1	1	0
		1	1	0	1	1	1	1
C=A-2*B	10	0	0	0	0	0	0	0
		0	1	0	0	0	1	0
		1	0	0	0	0	0	1
		1	1	0	0	0	1	1
		0	0	0	1	0	0	1
		0	1	0	1	0	1	1
		1	0	0	1	1	1	0
		1	1	0	1	0	0	0
		0	0	1	1	0	1	1
		0	1	1	1	1	0	1
		1	0	1	1	0	0	0
		1	1	1	1	0	1	0
		0	0	1	0	1	1	0
		0	1	1	0	0	0	0
		1	0	1	0	1	1	1
		1	1	1	0	0	0	1
C=A-B	11	0	0	0	0	0	0	0
		0	1	0	0	0	1	1
		1	0	0	0	0	0	1
		1	1	0	0	0	0	0
		0	0	0	1	0	0	1
		0	1	0	1	0	0	0
		1	0	0	1	1	1	0
		1	1	0	1	0	0	1
		0	0	1	1	0	1	1
		0	1	1	1	0	1	0
		1	0	1	1	0	0	0
		1	1	1	1	0	1	1

2.7. Вычитатель с удвоением. На рис. 7 представлена одноразрядная схема сумматора Sub2. Ее функционирование описывается таблицей истинности табл. 7. Эта таблица вычисляет сумму $c-2*w = (a-2*b + v)$. Здесь коды чисел 'v' и 'w' в общем случае не являются кодами по основанию (-2) , а именно, код переноса '10' изображает число 2 (а не число -2) – см. выделенные в таблице строки. Остальные коды являются кодами по основанию (-2) .

Таблица 7 одноразрядной схемы вычитания с удвоением

a	b	v1	v2	w1	w2	c
0	0	0	0	0	0	0
0	1	0	0	0	1	0
1	0	0	0	0	0	1
1	1	0	0	0	1	1
0	0	1	1	0	1	1
0	1	1	1	1	0	1
1	0	1	1	0	0	0
1	1	1	1	0	1	0
0	0	1	0	1	1	0
0	1	1	0	0	0	0
1	0	1	0	1	1	1
1	1	1	0	0	0	1
0	0	0	1	0	0	1
0	1	0	1	0	1	1
1	0	0	1	1	1	0
1	1	0	1	0	0	0

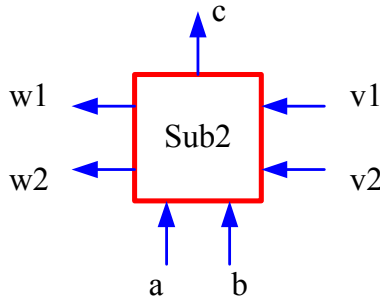


Рис. 7. Одноразрядная схема вычитателя с удвоением.

2.8. Инверсный вычитатель с добавлениями. На рис. 8 представлена одноразрядная такого вычитателя InvSubP. Существует 4 варианта одноразрядной схемы: InvSubP0, InvSubP1, InvSubP2, InvSubP3. Функционирование этих схем описывается таблицами истинности, которые вычисляют величину

$c - 2 * w = (s + v)$, где s зависит от вида схемы и величины cop . Рассмотрим эти 4 варианта.

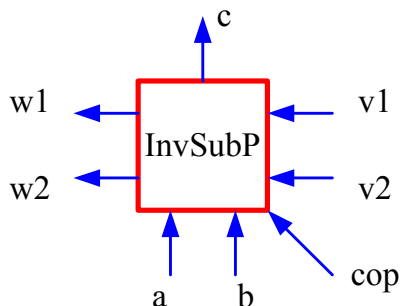


Рис. 8. Одноразрядная схема инверсный вычитатель с добавлением в 0-разряде.

2.8.1. Инверсный вычитатель с добавлением в 0-разряде (InvSubP0). В этом случае таблица истинности имеет вид табл. 8.1 и

$$S = \begin{cases} (a), & \text{если } cop = 0 \\ (-a), & \text{если } cop = 1 \end{cases}$$

Таблица 8.1

a	b	v2	v1	cop	w2	w1	c
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0
1	0	0	0	1	0	1	1

2.8.2. Инверсный вычитатель с добавлением в 1-разряде (InvSubP1). В этом случае таблица истинности имеет вид табл. 8.2 и

$$S = \begin{cases} (-b - 1 + v), & \text{если } cop = 0 \\ (b - 1 + v), & \text{если } cop = 1 \end{cases}$$

Таблица 8.2

a	b	v2	v1	cop	w2	w1	c
0	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0
0	0	0	1	0	0	0	0
0	1	0	1	0	0	1	1
0	0	0	0	1	0	1	1
0	1	0	0	1	0	0	0
0	0	0	1	1	0	0	0
0	1	0	1	1	0	0	1

2.8.3. Инверсный вычитатель с добавлением в разрядах с номерами 2, 4, 6, ... (InvSubP2). В этом случае таблица истинности имеет вид табл. 8.2 и

$$S = \begin{cases} (a + v), & \text{если } \text{cop} = 0 \\ (-a + v), & \text{если } \text{cop} = 1 \end{cases}$$

Таблица 8.3

a	b	v2	v1	cop	w2	w1	c
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
0	0	0	1	0	0	0	1
1	0	0	1	0	1	1	0
0	0	1	1	0	0	1	1
1	0	1	1	0	0	0	0
0	0	0	0	1	0	0	0
1	0	0	0	1	0	1	1
0	0	0	1	1	0	0	1
1	0	0	1	1	0	0	0
0	0	1	1	1	0	1	1
1	0	1	1	1	0	1	0

2.8.4. Инверсный вычитатель с добавлением в разрядах с номерами 3, 5, 7, ... (InvSubP3). В этом случае таблица истинности имеет вид табл. 8.2 и

$$S = \begin{cases} (-b + v), & \text{если } \text{cop} = 0 \\ (b + v), & \text{если } \text{cop} = 1 \end{cases}$$

Таблица 8.4

a	b	v2	v1	cop	w2	w1	c
0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	1
0	1	0	1	0	0	0	0
0	0	1	1	0	0	1	1
0	1	1	1	0	0	1	0
0	0	0	0	1	0	0	0
0	1	0	0	1	0	0	1
0	0	0	1	1	0	0	1
0	1	0	1	1	1	1	0
0	0	1	1	1	0	1	1
0	1	1	1	1	0	0	0

2.9. Вычитатель с добавлениями. На рис. 9 представлена одноразрядная такого вычитателя SubP. Существует 4 варианта одноразрядной схемы: SubP0, SubP1, SubP2, SubP3. Функционирование этих схем описывается таблицами истинности, которые вычисляют величину $c-2^* w = (s + v)$, где s зависит от вида схемы. Рассмотрим эти 4 варианта.

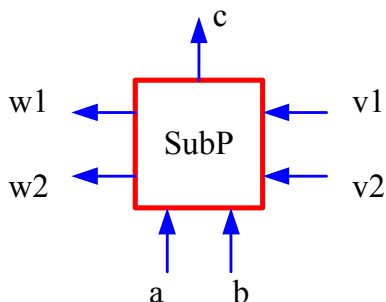


Рис. 9. Одноразрядная схема вычитатель с добавлением.

2.9.1. Вычитатель с добавлением в 0-разряде (SubP0). В этом случае таблица истинности имеет вид табл. 9.1 и $c = (a)$.

Таблица 9.1

a	b	v1	v2	w1	w2	c
0	0	0	0	0	0	0
1	0	0	0	0	0	1

2.9.2. Вычитатель с добавлением в 1-разряде (SubP1). В этом случае таблица истинности имеет вид табл. 9.2 и $c-2^* w = (-b - 1)$.

Таблица 9.2

a	b	v1	v2	w1	w2	c
0	0	0	0	0	1	1
0	1	0	0	0	1	0

2.9.3. Вычитатель с добавлением в разрядах с номерами 2, 4, 6, ... (SubP2). В этом случае таблица истинности имеет вид табл. 9.2 и $c-2^* w = (a + v)$.

Таблица 9.3

a	b	v1	v2	w1	w2	c
0	0	0	0	0	0	0
1	0	0	0	0	0	1
0	0	0	1	0	0	1
1	0	0	1	1	1	0
0	0	1	1	0	1	1
1	0	1	1	0	0	0

2.9.4. Вычитатель с добавлением в разрядах с номерами 3, 5, 7, ... (SubP3). В этом случае таблица истинности имеет вид табл. 9.2 и $c-2^*w = (-b + v)$.

Таблица 9.4

a	b	v1	v2	w1	w2	c
0	0	0	0	0	0	0
0	1	0	0	0	1	1
0	0	0	1	0	0	1
0	1	0	1	0	0	0
0	0	1	1	0	1	1
0	1	1	1	0	1	0

2.10. Вычитатель с уменьшениями. На рис. 10 представлена одноразрядная такого вычитателя SubM. Существует 4 варианта одноразрядной схемы: SubM0, SubM1, SubM2, SubM3. Функционирование этих схем описывается таблицами истинности, которые вычисляют величину $c-2^*w = (s + v)$, где s зависит от вида схемы. Рассмотрим эти 4 варианта.

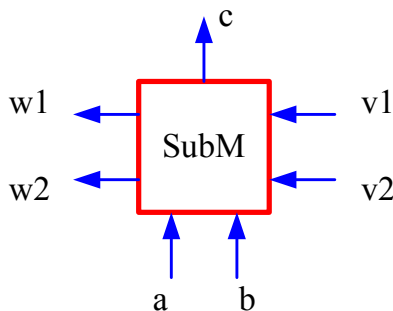


Рис. 10. Одноразрядная схема вычитатель с уменьшением.

2.10.1. Вычитатель с уменьшением в 0-разряде (SubM0). В этом случае таблица истинности имеет вид табл. 10.1 и $c-2^*w = (-a)$.

Таблица 10.1

a	b	v1	v2	w1	w2	c
0	0	0	0	0	0	0
1	0	0	0	0	1	1

2.10.2. Вычитатель с уменьшением в 1-разряде (SubM1). В этом случае таблица истинности имеет вид табл. 10.2 и $c-2^*w=(b+v+1)$.

Таблица 10.2

a	b	v1	v2	w1	w2	c
0	0	0	0	0	0	1
0	1	0	0	1	1	0
0	0	0	1	1	1	0
0	1	0	1	1	1	1

2.10.3. Вычитатель с уменьшением в разрядах с номерами 2, 4, 6, ... (SubM2). В этом случае таблица истинности имеет вид табл. 10.2 и $c-2^*w = (-a + v)$.

Таблица 10.3

a	b	v1	v2	w1	w2	c
0	0	0	0	0	0	0
1	0	0	0	0	1	1
0	0	0	1	0	0	1
1	0	0	1	0	0	0
0	0	1	1	0	1	1
1	0	1	1	0	1	0

2.10.4. Вычитатель с уменьшением в разрядах с номерами 3, 5, 7, ... (SubM3). В этом случае таблица истинности имеет вид табл. 10.2 и $c-2^*w = (b + v)$.

Таблица 10.4

a	b	v1	v2	w1	w2	c
0	0	0	0	0	0	0
0	1	0	0	0	0	1
0	0	0	1	0	0	1
0	1	0	1	1	1	0
0	0	1	1	0	1	1
0	1	1	1	0	0	0

2.11. Знакоопределитель (Seven, Sodd). Одноразрядные схемы знакоопределителя используются в схеме для определения знака числа, представленного М-кодом. Одноразрядный знакоопределитель представлен на рис. 11 и имеет две модификации:

Seven – одноразрядная схема знакоопределителя для разряда с четным номером,

Sodd - одноразрядная схема знакоопределителя для разряда с нечетным номером,

Коды переносов интерпретируются следующим образом:

00 – код имеет нулевое значение,

01 – код имеет положительное значение,

10 – код имеет отрицательное значение.

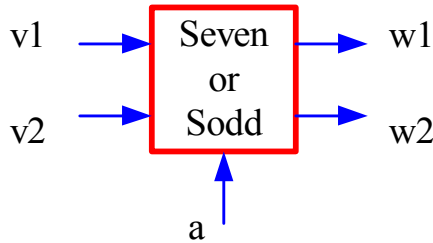


Рис. 11. Одноразрядная знакоопределитель.

Функционирование одноразрядных знакоопределителей описывается табл. 11.1 и 11.2.

Таблица 11.1 для одноразрядной схемы Seven

a	v2	v1	w2	w1
0	0	0	0	0
0	0	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	1	0	1

Эта таблица реализует правило:

$\langle w2, w1 \rangle = \langle v2, v1 \rangle$, если $a = '0'$,

$\langle w2, w1 \rangle = '01'$, если $a = '1'$,

Таблица 11.2 для однократной схемы Sodd

a	v2	v1	w2	w1
0	0	0	0	0
0	0	1	0	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	1	1	1

Эта таблица реализует правило:

$\langle w2, w1 \rangle = \langle v2 \ v1 \rangle$, если $a = \langle 0 \rangle$,

$\langle w2, w1 \rangle = \langle 11 \rangle$, если $a = \langle 1 \rangle$,

3. Многоразрядные схемы алгебраического сложения

3.1. Линейные схемы

Алгоритмы алгебраического сложения при отрицательном основании столь же просты, как и при положительном основании. Покажем это, для чего рассмотрим операцию **обратного сложения**, выполняемую по формуле $a = -b \cdot c$. Из анализа таблицы обратного сложения видно, алгоритм обратного сложения по сложности эквивалентен алгоритму обычного сложения кодов действительных чисел по положительному основанию. В связи с этим обратное сложение следует использовать как базовую операцию алгебраического сложения при отрицательном основании, а обычное сложение, как более сложную операцию, сводить к трем операциям обратного сложения: $a = b + c = -(-b \cdot 0) - (-c \cdot 0)$. Практически операции инвертирования (являющиеся частным случаем обратного сложения) могут выполняться параллельно с основной операцией, что существенно уменьшает время сложения при любых знаках слагаемых и знаках, стоящих перед слагаемыми в формуле алгебраического сложения.

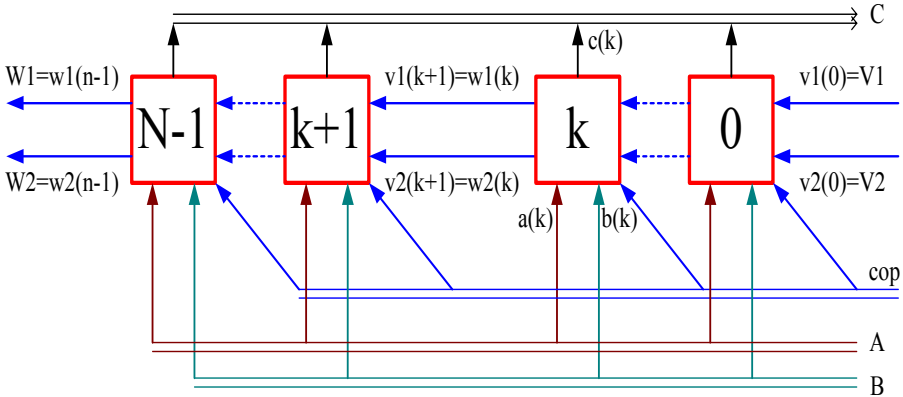


Рис. 1.

В общем случае многоразрядные схемы алгебраического сложения состоят из последовательно соединенных одноразрядных схем – см. рис. 1, где

N - разрядность кодов,

$k = \{0, 1, 2, \dots, n-2, n-1\}$ - номера разрядов и одноразрядных схем,

cop - код операции, общий для всех одноразрядных схем,

$V1, V2$ – разряды входного переноса, изображающие число V ,
 $W1, W2$ – разряды выходного переноса, изображающие число W ,
 A, B – операнды,
 C – результат.

Таблица 1. Линейные многоразрядные схемы алгебраического сложения.

Обозначение	Название	Обозначение одноразрядной схемы	Формула
nInv	Инвертор	Inv	$C = -A$
nInv2	Инвертор-удвоитель	Inv2	$C = \text{cop} * A$, $\text{cop} = (-2, -1, 0, 1)$
nInvAdd	Инверсный сумматор	InvAdd	$C = -A - B$
nAdd	Сумматор	Add	$C = A + B$
nSub	Вычитатель	Sub	$C = A - B$
nSub2	Вычитатель с удвоением	Sub2	$C = A - 2 * B$
nAddAlg	Алгебраический сумматор	AddAlg	$C = A + \text{cop} * B$ $\text{cop} = (-2, -1, 0, 1)$
nSubP	Вычитатель разреженных кодов с добавлением 2	SubP	
nSubM	Вычитатель разреженных кодов с вычитанием 2	SubM	
nInvSubP	Инверсный вычитатель разреженных кодов с добавлением 2	InvSubP	

Многоразрядные схемы, построенные в соответствии с рис. 1, будем называть **линейными**. В частных случаях линейных схем код операции и\или второй операнд могут отсутствовать. Входной перенос V , как правило, равен нулю. Выходной перенос W ,

отличный от нуля, свидетельствует о переполнении. В таблице 1 перечислены линейные многоразрядные схемы алгебраического сложения.

Последние в табл. 1 три схемы вычитателей будут использованы далее при описании кодирования и декодирования. Коды, участвующие в этих вычитаниях, представлены в табл. 2 и называются далее **разреженными** кодами. В табл. 2 символом «*» отмечены разряды, принимающие любое значение. Значение других разрядов фиксировано. При построении вычитателей учитывается то, что в определенных разрядах значения некоторых слагаемых фиксировано (равно 0). В сущности, операнды А и В являются выборкой четных и нечетных разрядов из некоторого единственного операнда. Рассмотрим эти вычитатели подробнее.

Таблица 2.

Bit	...	7	6	5	4	3	2	1	0
A	...	0	*	0	*	0	*	0	*
B	...	*	0	*	0	*	0	*	0
Q	...	0	0	0	0	0	0	1	0

3.2. Вычитатель разреженных кодов с добавлением 2 – nSubP.

Этот вычитатель вычисляет $C=A-B-Q$, где $Q=-2$, т. е. $C=A-B+2$. Он состоит из следующих одноразрядных схем:

SubP0 - блок 0-разряда,

SubP1 - блок 1-разряда,

SubP2 - блок разряда с четным номером (2, 4, 6, ...),

SubP3 - блок разряда с нечетным номером (3, 5, 7,...).

На рис. 2 представлена схема соединения этих одноразрядных схем между собой, где для каждой одноразрядной схемы указан ее тип и номер разряда.

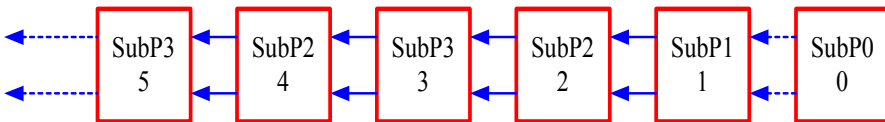


Рис. 2

3.3. Вычитатель разреженных кодов с вычитанием 2 – nSubM.

Этот вычитатель вычисляет $C=-A+B+Q$, где $Q=-2$, т. е.

$C = -A + B - 2$. Он состоит из следующих одноразрядных схем:

SubM0 - блок 0-разряда,

SubM1 - блок 1-разряда,

SubM2 - блок разряда с четным номером (2, 4, 6, ...),

SubM3 - блок разряда с нечетным номером (3, 5, 7, ...).

На рис. 3 представлена схема соединения этих одноразрядных схем между собой, где для каждой одноразрядной схемы указан ее тип и номер разряда.

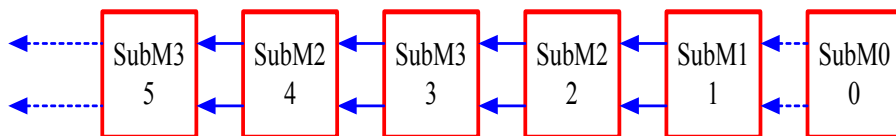


Рис. 3

3.4. Инверсный вычитатель разреженных кодов с добавлением 2 - nInvSubP

Этот вычитатель вычисляет либо $C = A - B - Q$, либо $C = -A + B - Q$, где $Q = -2$, т. е. либо $C = A - B + 2$, либо $C = -A + B + 2$. Он состоит из следующих одноразрядных схем:

InvSubP0 - блок 0-разряда,

InvSubP1 - блок 1-разряда,

InvSubP2 - блок разряда с четным номером (2, 4, 6, ...),

InvSubP3 - блок разряда с нечетным номером (3, 5, 7, ...).

На рис. 4 представлена схема соединения этих одноразрядных схем между собой, где для каждой одноразрядной схемы указан ее тип и номер разряда. В данном вычитателе используется код операции:

- при $cop=0$ выполняется вычитание $C = A - B$,
- при $cop=1$ выполняется вычитание $C = B - A$.

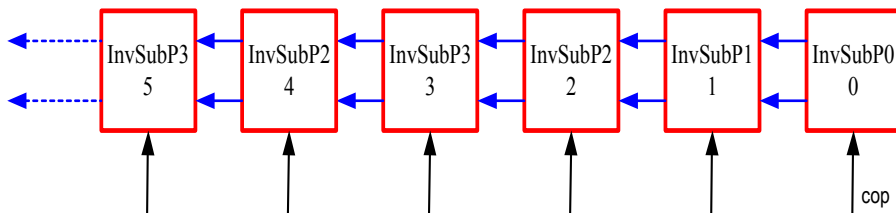


Рис. 4

3.5. Многоярусные схемы

Алгебраические сумматоры могут быть выполнены в виде комбинации сумматоров и инверторов. Общая схема таких алгебраических сумматоров (которые далее называются **многоярусными**) представлена на рис. 5, где показано, что слагаемые A и B через инверторы поступают на входы сумматора, который вычисляет алгебраическую сумму C . Вид алгебраического сложения определяется сигналами $q1$ и $q2$. Перечень многоярусных алгебраических сумматоров приведен в табл. 3.

Таблица 3

Обозначение	Название
nAddAlg3	Трехярусный алгебраический сумматор
nAdd3	Трехярусный сумматор
nSub3	Трехярусный вычитатель

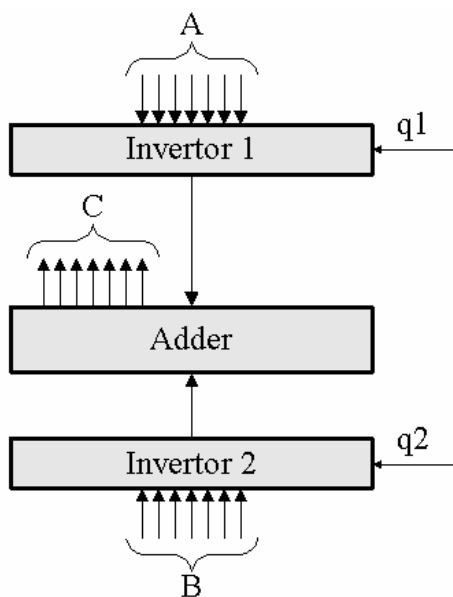


Рис. 5.

Таблица 4.

q2	q1	Операция
11	11	A+B
01	11	A-B
11	01	-A+B
01	01	-A-B
00	01	-A
01	00	-B
00	11	A
11	00	B
11	10	2A+B
01	10	2A-B
00	10	2A
10	11	A+2B
10	01	-A+2B
10	00	2B
10	10	2A+2B

3.6. Трехярусный алгебраический сумматор nAddAlg3 содержит инверторы-удвоители nInv2 и инверсный сумматор InvAdd. Код слагаемого A преобразуется в инверторе 1 по формуле $\bar{A} = q_1 \cdot A$, $q_1 = -2, -1, 0, 1$, а код слагаемого B преобразуется в инверторе 2 по формуле $\bar{B} = q_2 \cdot B$, $q_2 = -2, -1, 0, 1$. Преобразованные слагаемые складываются в сумматоре по формуле $C = -\bar{A} - \bar{B}$. Таким образом, алгебраический сумматор вычисляет $C = (-q_1 \cdot A - q_2 \cdot B)$. В табл. 4 указана связь между выполняемой операцией и кодами инверсий.

3.7. Трехярусный сумматор nAdd3 содержит инверторы-удвоители nInv и инверсный сумматор InvAdd. Код слагаемого A преобразуется в инверторе 1 по формуле $\bar{A} = -A$, а код слагаемого B преобразуется в инверторе 2 по формуле $\bar{B} = -B$. Преобразованные слагаемые складываются в сумматоре по формуле $C = -\bar{A} - \bar{B}$. Таким образом, сумматор вычисляет $C = A + B$.

3.8. Трехярусный вычитатель nSub3 содержит только один инвертор nInv и инверсный сумматор InvAdd. Код слагаемого A преобразуется в инверторе по формуле $\bar{A} = -A$, а код слагаемого B не преобразуется (поступает на вход инверсного сумматора непосредственно). Слагаемые складываются в сумматоре по

формуле $C = -\bar{A} - B$. Таким образом, алгебраический сумматор вычисляет $C = A - B$.

3.9. Знакоопределитель nSign определяет знак числа, представленного М-кодом. Знакоопределитель представлен на рис. 6, где показана схема соединения вышеописанных одnorазрядных блоков Seven и Sodd между собой и с регистром М-кода. При этом приняты следующие обозначения:

N -разрядность знакоопределителя,

A – входной код,

$W1, W2$ – выходные переносы.

Код выходных переносов ($W2, W1$) интерпретируется следующим образом:

00 – код имеет нулевое значение,

01 – код имеет положительное значение,

10 – код имеет отрицательное значение.

Таким образом, если $W2=1$, то $A < 0$, а если $W2=0$, то $A \geq 0$.

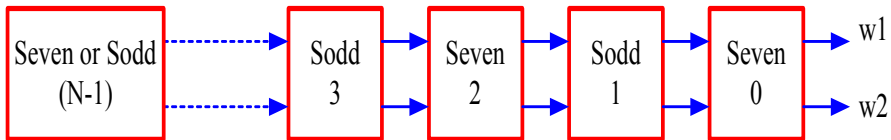


Рис. 6.

Глава 3. Алгебраическое сложение С-кодов

1. Многоразрядные схемы алгебраического сложения
2. Операции алгебраического сложения

1. Многоразрядные схемы алгебраического сложения С-кодов

1.1. Алгебраический сумматор мантисс С-кода (SumM2)

Алгебраический сумматор мантисс комплексного кода с удвоением состоит из двух независимых одинаковых частей – сумматоров действительной и мнимой частей. Каждый из этих сумматоров является N -разрядным алгебраическим сумматором М-кодов – $nAddAlg3$.

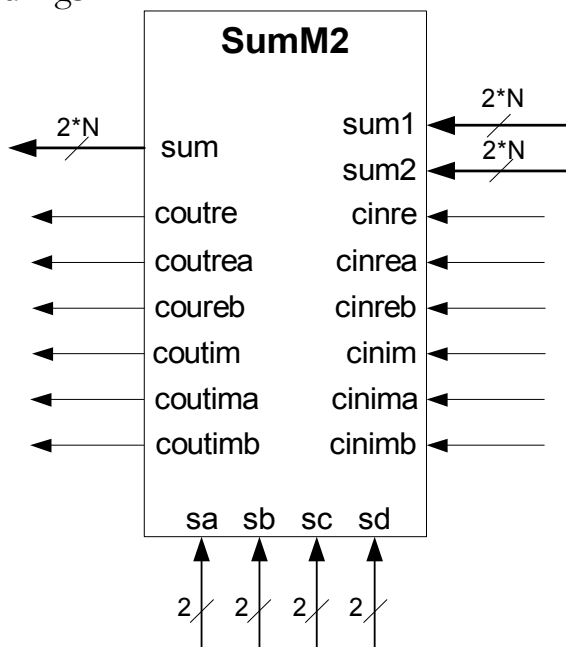


Рис. 1

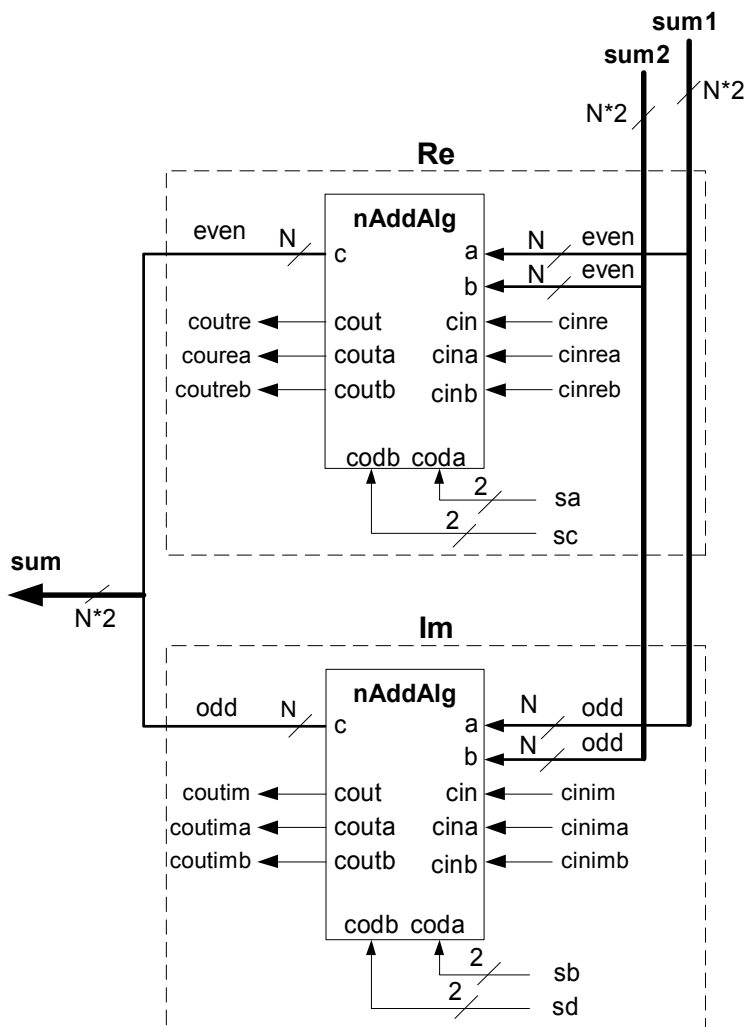


Рис. 2.

На рис. 1 и 2 представлены соответственно условная и полная схемы SumM2. При этом приняты следующие обозначения:

- *sum1*, *sum2* – входы сумматора,
- *sum* – выход сумматора,
- *sa*, *sc* – управляющие коды операций с действительными частями кода *A* и *B* соответственно, обозначенные выше как q_1 , q_2 ,

- sb, sd – управляющие коды операций с мнимыми частями кода A и B соответственно, обозначенные выше как q_1, q_2 ,
- cinre, cinrea, cinreb – входные переносы, поступающие соответственно в реальные части инвертора, сумматора A и сумматора B,
- cinim, cinima, cinimb – входные переносы, поступающие соответственно в мнимые части инвертора, сумматора A и сумматора B,
- coutre, coutrea, coutreb – выходные переносы из реальной части инвертора, сумматора A и сумматора B соответственно,
- coutim, coutima, coutimb – выходные переносы из мнимой части инвертора, сумматора A и сумматора B соответственно.

1.2. Полный алгебраический сумматор мантисс C-кодов (AddFull)

Полный алгебраический сумматор AddFull комплексных кодов изображен на рис. 3. Он производит суммирование комплексных кодов, которые в общем случае состоят из двух частей – кода мантиссы и кода целой части, появляющейся при переполнении. В дальнейшем будем обозначать весь код символом A (или B, C), мантиссу – символом Amant (или Bmant, Cmant), целую часть – символом Aint (или Bint, Cint). Таким образом, AddFull производит алгебраическое сложение кодов (Aint+Amant), (Bint+Bmant) и образует результирующий код (Cint+Cmant). Кроме того, при сложении учитывается входной перенос MantCarry, поступающий в младшие разряды сумматора мантисс. Этот перенос генерируется в блоке GenCarry. Перенос IntCarry, возникающий в старших разрядах сумматора мантиссы, поступает в младшие разряды сумматора целых частей.

Результирующий код Cint всегда записывается в регистр Vmant одновременно с записью суммы мантисс Cmant в регистр результата. Кроме того, перенос IntCarry всегда записывается в регистр Vmant2.

Содержимое регистра **Vmant2** иногда переписывается в блок **GenCarry**, где сохраняется для возможного использования в следующих операциях – см операции **Bini**.

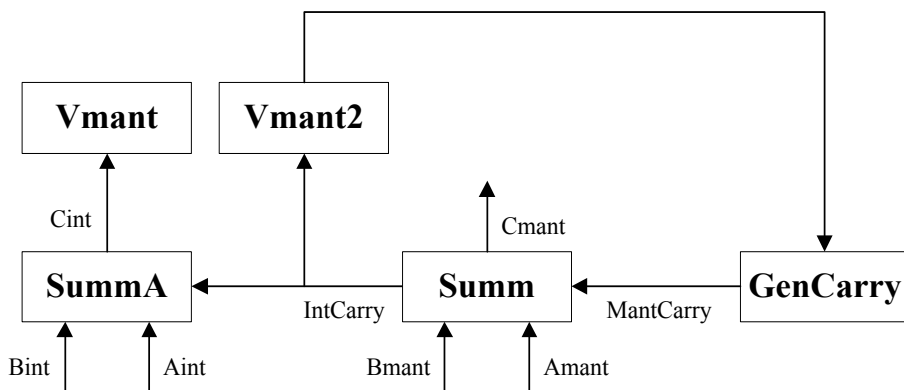


Рис. 3.

Такая организация сумматора позволяет выполнять следующие действия:

- Алгебраическое сложение мантисс с формированием переполнения в регистре **Vmant** для последующей нормализации
- Алгебраическое сложение мантисс, как целых кодов, с формированием переполнения в регистре **Vmant** для выдачи сигнала переполнения
- То же, но с генерацией начального переноса **MantCarry** в блоке **GenCarry** для инкрементных и декрементных операций
- Алгебраическое сложение мантисс, как целых кодов, с запоминанием выходного переноса **IntCarry** в регистре блока **GenCarry**
- Алгебраическое сложение мантисс, как целых кодов, с приемом начального переноса **MantCarry**, запомненного в регистре блока **GenCarry** в предыдущей операции. Это необходимо для организации алгебраического сложения с удвоенной точностью

- Алгебраическое сложение мантисс, как целых кодов, с приемом начального переноса MantCarry, запомненного в регистре блока GenCarry в предыдущей операции, а также с запоминанием выходного переноса IntCarry в регистре блока GenCarry (на следующем такте). Это необходимо для организации алгебраического сложения с утроенной и более точностью
- Алгебраическое сложение кодов, содержащих целую и дробную части

2. Операции алгебраического сложения С-кодов

Далее перечисляются те операции, которые могут быть выполнены на полном алгебраическом сумматоре С-кодов AddFull. Для выполнения операций формируются управляющие коды операций sa, sc, sb, sd и сигналы входного переноса cinre, cinrea, cinreb, cinim, cinima, cinimb, указанные при описании алгебраического сумматора мантисс комплексного кода.

2.1. Унарные операции - Uni

Эти операции состоят в преобразовании одного операнда целого С-кода и выполняются за один такт. Выход за разрядную сетку служит сигналом переполнения. Операции Uni перечислены в табл. 1.

Таблица 1.

№	Обозначение	sA	sB	Наименование
1	UniInv	-1	-1	Инвертирование ($C = -A$)
2	UniCon	1	-1	Сопряженное число ($C = \text{Re}A - \text{Im}A$)
3	UniMir	-1	1	Зеркальное число ($C = -\text{Re}A + \text{Im}A$)
4	UniRe	1	0	Выделение действительной части ($C = \text{Re}A$)
5	UniIm	0	1	Выделение мнимой части ($C = \text{Im}A$)
6	UniRotate90	-1	1	Поворот целого на 90 гр. ($C = A^*j$)
7	UniRotate_90	1	-1	Поворот целого на -90 гр. ($C = -A^*j$)
11	IncrementRe	1	1	Реальный инкремент целого ($C = A + 1$)
12	DecrementRe	1	1	Реальный декремент целого ($C = A - 1$)
13	IncrementIm	1	1	Мнимый инкремент целого ($C = A + j$)
14	DecrementIm	1	1	Мнимый декремент целого ($C = A - j$)
15	IncrementReIm	1	1	Инкременты целого ($C = A + 1 + j$)
16	DecrReIncrIm	1	1	Реальный декремент и мнимый инкремент целого ($C = A - 1 + j$)
17	IncrReDecrIm	1	1	Реальный инкремент и мнимый декремент целого ($C = A + 1 - j$)
18	DecrementReIm	1	1	Декременты целого ($C = A - 1 - j$)
19	UniRokir	1	1	Рокировка

В табл. 1 указана зависимость между видом операции и управляющими кодами операций. В табл. 1а указана зависимость между видом операции и сигналами входного переноса. Именно эта таблица реализуется в блоке GenCarry.

Таблица 1а.

операция	cinre	cinrea	cinreb	cinim	cinima	cinimb
1+A	1	0	0	0	0	0
1-A	0	1	0	0	0	0
A+j	0	0	0	1	0	0
A-j	0	0	0	0	1	0
A+1+j	1	0	0	1	0	0
A-1+j	0	1	0	1	0	0
A+1-j	1	0	0	0	1	0
A-1-j	0	1	0	0	1	0

2.2. Унарные операции с нецелыми кодами - Unf

Эти операции состоят в преобразовании одного операнда C-кода. Они выполняются по следующей схеме:

- Операция с единственным операндом
- Нормализация результата

Эти операции перечислены в табл. 2. В ней указана также зависимость между видом операции и управляющими кодами операций.

Таблица 2.

№	Обозначение	sA	sB	Наименование
1	UnfInv	-1	-1	Инвертирование ($C=-A$)
2	UnfCon	1	-1	Сопряженное число ($C=\text{Re}A-\text{Im}A$)
3	UnfMir	-1	1	Зеркальное число ($C=-\text{Re}A+\text{Im}A$)
4	UnfRe	1	0	Выделение действительной части ($C=\text{Re}A$)
5	UnfIm	0	1	Выделение мнимой части ($C=\text{Im}A$)
6	UnfRotate90	-1	1	$C=j*A$
7	UnfRotate_90	1	-1	$C=-j*A$
13	UnfDivisTwo	-1	-1	$C=A/2$
15	UnfMultJdivisTwo	-1	1	$C=j*A/2$
16	UnfMultMinusJ-divisMinusTwo	1	-1	$C=-j*A/2$

2.3. Бинарные операции с целыми кодами - Bini

При алгебраическом сложении целых С-кодов их реальные и мнимые части складываются независимо. Возможны следующие операции:

$$\text{Re } C = a \text{ Re } A + b \text{ Re } B,$$

$$\text{Im } C = c \text{ Im } A + d \text{ Im } B,$$

где константы a , b , c , d могут принимать значения 1, 0, -1. Эти операции с целыми С-кодами состоят в алгебраическом сложении двух операндов и выполняются за один такт. Выход за разрядную сетку служит сигналом переполнения.

Таблица 3.

№	sA	sB	sC	sD	Наименование
1	1	1	1	1	Сложение ($C=A+B$)
2	1	1	-1	-1	Вычитание ($C=A-B$)
3	-1	-1	-1	-1	Сложение с инверсией ($C=-A-B$)
4	1	0	1	0	Сложение реальных частей ($C=A+B$)
5	1	0	-1	0	Вычитание реальных частей ($C=A-B$)
6	-1	0	-1	0	Сложение реальных частей с инверсией ($C=-A-B$)
7	0	1	0	1	Сложение мнимых частей ($C=A+B$)
8	0	1	0	-1	Вычитание мнимых частей ($C=A-B$)
9	0	-1	0	-1	Сложение мнимых частей с инверсией ($C=-A-B$)
10	1	-1	-1	1	Вычитание сопряженных чисел
11	1	-1	1	-1	Сложение сопряженных чисел
12	1	1	-1	1	Вычитание сопряженного числа
13	1	1	1	-1	Сложение с сопряженным числом

Эти операции перечислены в табл. 3. В ней указана также зависимость между видом операции и управляющими кодами операций.

2.4. Bini с запоминанием переполнения - BiniR

Эти операции выполняются также, как и операции Bini. Отличие заключается в том, что выход за разрядную сетку запоминается в виде кода переполнения для использования в следующей операции.

2.5. Bini с учетом предыдущего переполнения - BiniW.

Эти операции выполняются также, как и операции Bini. Выход за разрядную сетку служит сигналом переполнения. Но, в отличие от Bini и BiniR при алгебраическом сложении учитывается перенос в младшие разряды, который формируется из кода переполнения, полученного в предыдущей операции.

2.6. Bini с учетом предыдущего переполнения и запоминанием переполнения - BiniWR.

Эти операции выполняются также, как и операции Bini. Выход за разрядную сетку запоминается в виде кода переполнения и используется в следующей операции. При алгебраическом сложении учитывается (как и в BiniR) перенос в младшие разряды, который формируется из кода переполнения, полученного в предыдущей операции.

2.7. Bini удвоенной точности.

В этих операциях участвуют операнды, представленные в виде двух целых С-кодов M1 и M2. Предполагается, что M1 содержит младшую часть кода, а M2 содержит старшую часть кода. Алгебраическое сложение выполняется по следующей схеме:

- Алгебраическое сложение целых кодов A1 и B1 с запоминанием переполнения – см. BiniR,
- Алгебраическое сложение целых кодов A2 и B2 с учетом предыдущего переполнения – см. BiniW.

Аналогично могут быть выполнены бинарные операции утроенной, учетверенной и т. д. точности. Рассмотрим, например, сложение с утроенной точностью. Оно выполняется по следующей схеме:

- Алгебраическое сложение целых кодов A1 и B1 с запоминанием переполнения – см. BiniR,
- Алгебраическое сложение целых кодов A2 и B2 с учетом предыдущего переполнения и запоминанием переполнения – см. BiniWR,
- Алгебраическое сложение целых кодов A3 и B3 с учетом предыдущего переполнения см. BiniW.

2.8. Бинарные операции с нецелыми кодами - Binf

Эти операции выполняются с С-кодами по следующей схеме:

- Выравнивание порядков слагаемых,
- Операция алгебраического сложения,
- Нормализация результата. При этом может возникнуть сигнал переполнения.

Операции алгебраического сложения аналогичны операциям Bini. Они перечислены в табл. 3.

Глава 4. Групповой перенос

1. Теория группового переноса
2. Алгебраические сумматоры М-кодов с групповым переносом

1. Теория группового переноса

1.0. Введение.

Групповой перенос при сложении на комбинационном сумматоре служит, как обычно, для ускорения распространения переносов в параллельных сумматорах. С этой целью все разряды сумматора разбиваются на группы по $m > 1$ разрядов (см. рис. 1).

Каждая i -ая группа G_i ($i = 0 \div j$) представляет m -разрядный параллельный сумматор, имеющий входы для кода переноса в младший разряд группы (обозначим этот перенос через T_i) и выходы для кода переноса T'_{i+1} из старшего разряда этой группы.

Младший разряд группы G_i является m_i -ым разрядом сумматора, разряды которого пронумерованы от 0 до $n-1$. Старший разряд группы G_i является $(m_i + m - 1)$ -ым разрядом сумматора. Кроме того, каждая группа G_i (точнее: триггеры разрядов этой группы) нагружена на схему группового переноса GTC_p , вырабатывающую действительный код переноса T_{i+1} в младший разряд следующей группы G_{i+1} . Перенос T_{i+1} вырабатывается на основе анализа слагаемых и переносов T_i и T'_{i+1} .

Перед началом суммирования на все группы подается общий код нулевого переноса (см. цепь сигнала T_0 на рис. 1). В процессе сложения код переноса в данную группу вырабатывается предыдущей группой.

Очевидно, что перенос T_{i+1} может отличаться от переноса T'_{i+1} лишь в том случае, когда комбинация слагаемых в данной

группе такова, что действительный перенос T_{i+1} из группы G_i зависит от величины переноса T_i в эту группу; назовем эти комбинации критическими. Перенос T_{i+1} обязательно отличается от T'_{i+1} , если кроме этого условия $T_i \neq 0$.

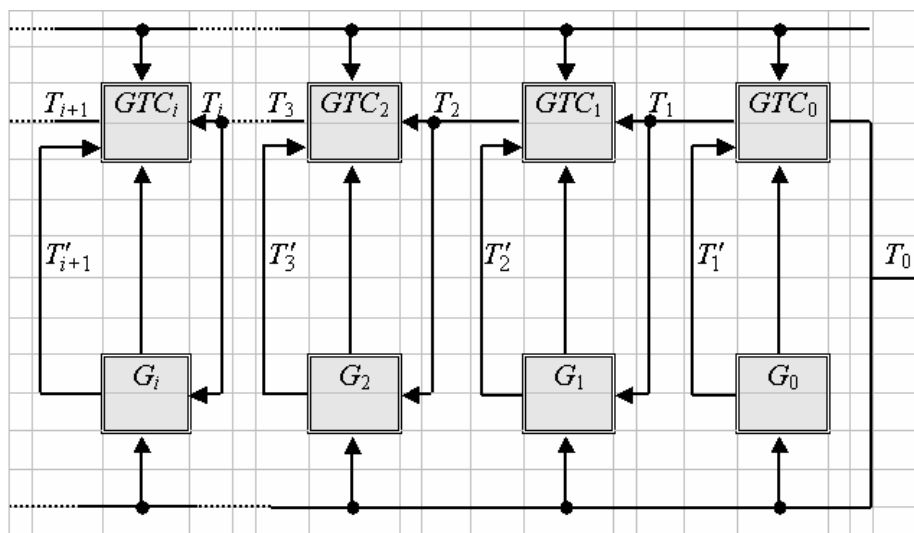


Рис. 1.

При соблюдении этого условия GTC_i должна блокировать перенос T'_{i+1} и выработать перенос T_{i+1} на основе анализа слагаемых и входного переноса T_i ; при несоблюдении этого условия ($T_{i+1} = T'_{i+1}$) GTC_i должна пропускать код переноса T'_{i+1} на выход в качестве кода переноса T_{i+1} .

Из этих общих замечаний следует, что при синтезе GTC_i возникают три задачи:

- (1) Определение критических комбинаций слагаемых.
- (2) Синтез схемы, вырабатывающей выходной перенос на основе анализа критических комбинаций и входного переноса; назовем эту схему «схемой выработки переноса (TGC)» а перенос на ее выходе обозначим через T''_{i+1} .

- (3) Синтез схемы, вырабатывающей действительный перенос T_{i+1} на основе анализа переносов T'_{i+1} и T''_{i+1} ; назовем эту схему «схемой замещения (SC)».

1.1. Определение критических комбинаций.

Обозначим

$K'(a_i) = \alpha_{m-1} \dots \alpha_2 \alpha_1 \alpha_0$ - m -разрядный код i -ой группы первого слагаемого, представляющий число a_i ;

$K'(b_i) = \beta_{m-1} \dots \beta_2 \beta_1 \beta_0$ - m -разрядный код i -ой группы первого слагаемого, представляющий число b_i ;

$Q_i = a_i + b_i$ - i -ый групповой результат (без учета переносов).

Определение критических комбинаций рассмотрим вначале для двухразрядных групп ($m=2$). В этом случае

$$K'(a_i) = \alpha_1 \alpha_0 \text{ и } a_i = -2; -1; 0; 1;$$

$$K'(b_i) = \beta_1 \beta_0 \text{ и } b_i = -2; -1; 0; 1;$$

$$Q_i = a_i + b_i, \text{ т.е. } Q_i = -4; -3; -2; -1; 0; 1; 2.$$

Составим табл. 1, в которой указаны значения переносов T_{i+1} при всевозможных значениях величин Q_i и T_i (значение T_{i+1} находится на пересечении строки и столбца, обозначенных конкретными значениям Q_i и T_i). Из табл. 1 следует, что критическими являются лишь те комбинации разрядов кодов $K'(a_i)$ и $K'(b_i)$, при которых

$$Q_i = -3; -2; -1; 1; 2.$$

Таблица 1.

Q_i	T_i		
	1	0	-1
-4	-1	-1	-1
-3	0	-1	-1
-2	0	0	-1
-1	0	0	0
0	0	0	0
1	1	0	0
2	1	1	0

Назовем эти значения Q_i критическими. Заметим, что при сложении обычных двоичных кодов существует лишь одно критическое значение Q_i , при котором сумма пары разрядов по $\text{mod} 2$ равна 1 для всех пар разрядов данной группы.

При остальных значениях Q_i величина T_i не влияет на величину T_{i+1} . Следовательно, соответствующие комбинации не являются критическими.

Таблица 2.

Q_i	T_i		
	1	0	-1
-4	1	1	1
-3	0	1	1
-2	0	0	1
-1	0	0	0
0	0	0	0
1	0	0	0
2	0	0	0
3	0	0	0
4	0	0	0
5	-1	0	0
6	-1	-1	0
7	-1	-1	-1
8	-1	-1	-1
9	-1	-1	-1
10	-1	-1	-1

Так же составляется и анализируется табл. 2 для трехразрядных групп:

$$K'(a_i) = \alpha_2 \alpha_1 \alpha_0 \text{ и } a_i = -2; -1; \dots; 4; 5;$$

$$K'(b_i) = \beta_2 \beta_1 \beta_0 \text{ и } b_i = -2; -1; \dots; 4; 5;$$

$$Q_i = -4; -3; -2; \dots; 8; 9; 10.$$

Здесь критические значения имеют величину $Q_i = -3; -2; 5; 6$.

Аналогично определяются критические значения Q_i для многоразрядных групп с любым m . Очевидно, что каждому критическому значению Q_i соответствует одна или несколько критических комбинаций слагаемых разрядов. В частности, для

двухразрядных групп критические комбинации, соответствующие критическим значениям Q_i приведены в таблице 3.

Таблица 3.

Критические значения Q_i	Критические комбинации $\alpha_1\alpha_0 + \beta_1\beta_0$
$Q'_i = -3$	10+11 11+10
$Q''_i = -2$	10+00 00+10
$Q'''_i = 1$	11+11 01+00 00+01
$Q_i^v = 2$	01+01

1.2. Синтез логических формул для схемы выработки переноса распадается на два этапа. Рассмотрим их на примере синтеза TGC для двухразрядной группы.

Первый этап состоит в синтезе логических формул сигналов критических значений $Q_i = Q'_i; Q''_i; Q'''_i; Q_i^v$ (сигналы будем обозначать теми же буквами). Эти формулы синтезируются на основе таблицы 3 в виде совершенных дизъюнктивных нормальных форм:

$$\begin{aligned}
 Q'_i &= \alpha_1 \bar{\alpha}_0 \beta_1 \beta_0 \vee \alpha_1 \alpha_0 \beta_1 \bar{\beta}_0 = \alpha_1 \beta_1 (\bar{\alpha}_0 \beta_0 \vee \alpha_0 \bar{\beta}_0); \\
 Q''_i &= \alpha_1 \bar{\alpha}_0 \bar{\beta}_1 \bar{\beta}_0 \vee \bar{\alpha}_1 \bar{\alpha}_0 \beta_1 \bar{\beta}_0 = \alpha_0 \beta_0 (\alpha_1 \bar{\beta}_1 \vee \bar{\alpha}_1 \beta_1); \\
 Q'''_i &= \alpha_1 \alpha_0 \beta_1 \beta_0 \vee \bar{\alpha}_1 \alpha_0 \bar{\beta}_1 \bar{\beta}_0 \vee \bar{\alpha}_1 \bar{\alpha}_0 \bar{\beta}_1 \beta_0 = \\
 &= \alpha_1 \alpha_0 \beta_1 \beta_0 \vee \bar{\alpha}_1 \bar{\beta}_1 (\bar{\alpha}_0 \beta_0 \vee \alpha_0 \bar{\beta}_0); \\
 Q_i^v &= \bar{\alpha}_1 \alpha_0 \bar{\beta}_1 \beta_0.
 \end{aligned}$$

Второй этап заключается в синтезе логических формул для анализа сигналов Q_i и T_i и выработки сигналов T''_{i+1} . При синтезе этих формул используется таблица 1 и один из вариантов кодирования переносов (естественно, выбирается тот вариант кодирования, который использован при синтезе схем групповых

сумматоров G_i). На основе таблицы 1 и выбранного варианта кодирования строится таблица 4, в которой указан вид кода KT_{i+1}'' в зависимости от критического значения Q_i и вида кода KT_i . Для удобства синтеза таблица 4 преобразуется в логическую таблицу 5, в которой используются следующие обозначения:

$KT_i = p_{mi}'' p_{mi}'$ (ибо перенос T_i является переносом в m -ый разряд сумматора);

$KT_{i+1}'' = q_2 q_1$ (ибо перенос T_{i+1}'' не всегда тождественен с переносом T_{i+1} , код которого по аналогии с предыдущим должен иметь вид: $KT_{i+1} = p_{m(i+1)}'' p_{m(i+1)}'$).

Таблица 4.

T_i	1	0	-1
KT_i	01	11	10
$Q_i' = -3$	11	10	10
$Q_i'' = -2$	11	11	10
$Q_i''' = 1$	01	11	11
$Q_i^{iv} = 2$	01	11	11

По таблице 5 составляются формулы для функций q_2 и q_1 :

$$q_2 = Q_i' \vee Q_i'' \vee Q_i''' \cdot p_{mi}'' \vee Q_i^{iv} \cdot \overline{p_{mi}};$$

$$q_1 = Q_i' \cdot \overline{p_{mi}} \vee Q_i'' \cdot p_{mi}' \vee Q_i''' \vee Q_i^{iv}.$$

Таблица 5.

ВХОДЫ			ВЫХОДЫ	
Q_i	p_{mi}''	p_{mi}'	q_2	q_1
Q_i'	0	1	1	1
	1	1	1	0
	1	0	1	0

Входы			Выходы	
Q_i	P''_{mi}	P'_{mi}	q_2	q_1
Q_i''	0	1	1	1
	1	1	1	1
	1	0	1	0
Q_i'''	0	1	0	1
	1	1	1	1
	1	0	1	1
Q_i^{iv}	0	1	0	1
	1	1	0	1
	1	0	1	1

При составлении этих формул использовался тот очевидный факт, что попарные конъюнкции сигналов Q_i всегда равны 0 (т.к. одновременно может быть лишь одно критическое значение Q_i).

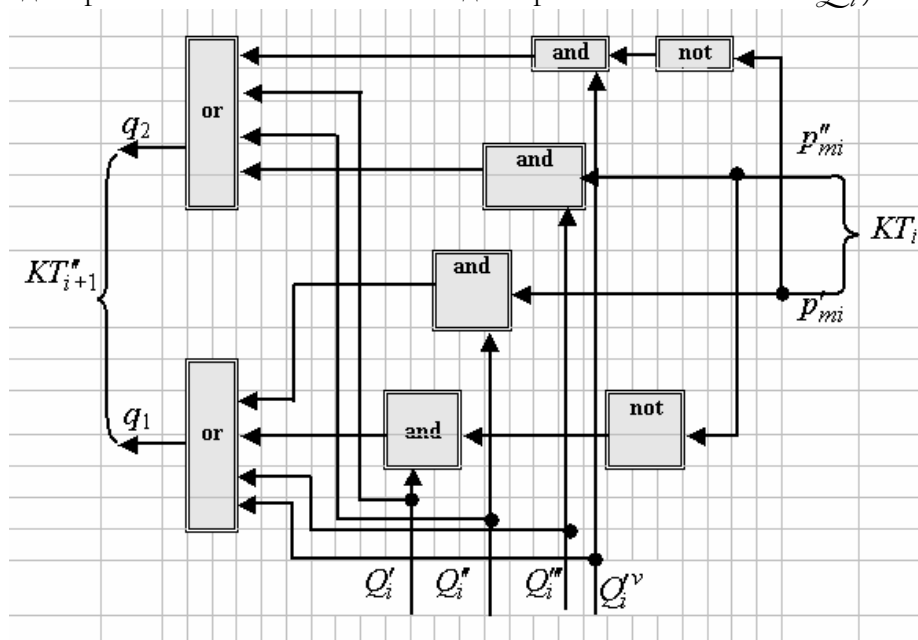


Рис. 2.

На основе полученных формул легко синтезируется схема выработки переноса для двухразрядной группы – см. рис. 2. Совершенно аналогично синтезируется TGC для многоразрядных групп.

1.3. Схема замещения вырабатывает сигнал наличия критических комбинаций

$$h = Q'_i \vee Q''_i \vee Q'''_i \vee Q_i{}^v,$$

который управляет выходными переносами следующим образом:

а) если $h = 0$, то выходной перенос T_{i+1} равен переносу T'_{i+1} из старшего разряда группы G_i ;

б) если $h = 1$, то выходной перенос T_{i+1} равен переносу T''_{i+1} из TGC .

Обозначим: $KT'_{i+1} = q_4 q_3$. Тогда:

$$T''_{mi} = q_4 \cdot \bar{h} \vee q_2 \cdot h; \quad T'_{mi} = q_3 \cdot \bar{h} \vee q_1 \cdot h.$$

Таблица 6.

α_K	β_K	Q_K	T_K	S_K	$K'(S_K)$	C_K	T_{K+1}
0	0	0	0	0	0	0	0
0	1	-1	0	-1	11	1	1
1	0	1	0	1	1	1	0
1	1	0	0	0	0	0	0
0	0	0	-1	-1	11	1	1
0	1	-1	-1	-2	10	0	1
1	0	1	-1	0	0	0	0
1	1	0	-1	1	11	1	1
0	0	0	1	1	1	1	0
0	1	-1	1	0	0	0	0
1	0	1	1	2	110	0	-1
1	1	0	1	1	1	1	0

Последние формулы (включая формулу для h) упрощаются в том случае, если для кодирования переносов выбран тот вариант, в котором код «отсутствующего» переноса равен “00”. Действительно, при этом код KT'_{i+1} на выходе TGC имеет значащие разряды (не отсутствует) лишь при наличии критических комбинаций. Следовательно,

$$h = q_2 \vee q_1;$$

$$T''_{mi} = (\overline{q_2 \vee q_1}) \cdot q_4 \vee q_2;$$

$$T'_{mi} = (\overline{q_2 \vee q_1}) \cdot q_3 \vee q_1.$$

Эти формулы служат для синтеза схемы замещения. Заметим, что при синтезе схем сумматоров по действительному основанию

схема замещения, вырождается в схему логического «ИЛИ» для сигналов T'_{i+1} и T''_{i+1} .

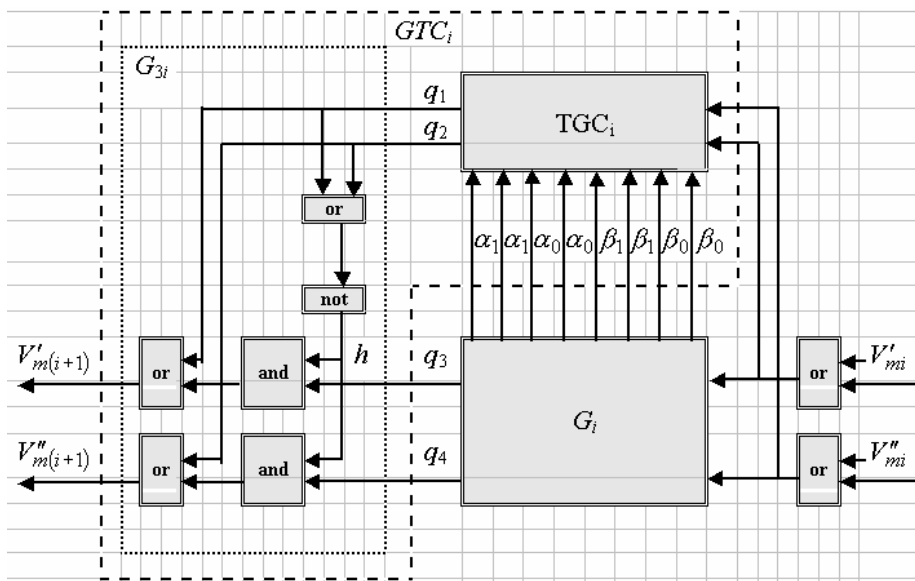


Рис. 3.

На рис. 3 представлена блок-схема одной группы сумматора, в которой используется TGC , изображенная на рис. 2. Схема замещения на блок-схеме изображена полностью. Структура SC не зависит от разрядности группы.

Таблица 7.

A_K	B_K	Q_K	$T'_K = p_{K1} + p_{K2}$		S_K	$K'(S_K)$	C_K	$K'(V'_{K+1}) = p_{K+2} + p_{K+1}$
0	0	0	0	0+0	0	*	0	00
0	1	-1	0		-1	11	1	01
1	0	1	0		1	1	1	00
1	1	0	0		0	0	0	00
0	0	0	1	1+0 или 0+1	1	1	1	00
0	1	-1	1		0	0	0	00
1	0	1	1		2	110	0	11
1	1	0	1		1	1	1	00

0	0	0	2	1+1	2	110	0	11
0	1	-1	2		1	1	1	00
1	0	1	2		3	111	1	11
1	1	0	2		2	110	0	11

2. Алгебраические сумматоры М-кодов с групповым переносом

2.1. Введение.

Применим вышеизложенные рассуждения для синтеза схем группового переноса при сложении М-кодов. Поскольку, как показано выше, алгебраические сумматоры М-кодов могут быть представлены в виде комбинации инверторов и обратных сумматоров, мы ограничимся рассмотрением группового переноса при инвертировании и обратном сложении.

Схема алгебраического сумматора с групповым переносом представлена на рис. 1. На этом рисунке и ниже приняты следующие обозначения:

k - количество разрядов в группе,

N – разрядность кодов,

GTC - k -разрядный групповой блок,

A, B - многоразрядные входы сумматора в целом или k -разрядные входы в схему группового переноса GTC;

C - многоразрядный выход или k -разрядный выход из схемы группового переноса GTC;

нечетный групповой перенос - групповой перенос при k – нечетном;

четный групповой перенос - групповой перенос при k – четном;

Pin – входной перенос в схему группового переноса GTC;

Pout - выходной перенос из схемы группового переноса GTC;

Pgr - выходной перенос, вычисляемый в GTC с учетом входного переноса Pin,

Crit – критическая комбинация; код, содержащий $(k+1)$ разрядов; такая сумма кодов A и B , при которой выходной перенос Pout зависит от входного переноса Pin;

CalcPout – блок вычисления выходного переноса Pout; выходной перенос Pout при наличии критической комбинации зависит от входного переноса Pin; точнее, при k – четном $Pout = Pin$, а при k – нечетном $Pout = \text{not}(Pin)$. При отсутствии критической комбинации выходной перенос $Pout = Pgr$;

CalcPout0 – блок **CalcPout** при k – четном;

CalcPout1 – блок **CalcPout** при k – нечетном;

CompM - блок сравнения $(k+1)$ -разрядных кодов,

h - сигнал наличия критических комбинаций.

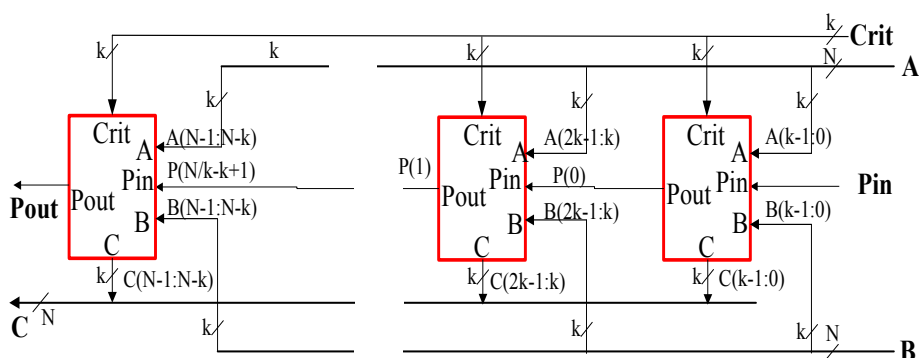


Рис. 1.

2.2. Инверсный сумматор с нечетным групповым переносом.

Этот инверсный сумматор вычисляет $C = -A - B$. В нем используются

- инверсный сумматор k -разрядных кодов **InvAdd**;
- k -разрядный групповой блок ГТС, схема которого изображена на рис. 2.

Внутри блока ГТС используется в данном случае блок **CalcPout** при k – нечетном, т.е. блок **CalcPout1**. Критическая комбинация **Crit** в данном случае определяется по табл. 2 .

Таблица 2.

Число анализируемых разрядов k	$2^{k-1}(-1)^k$	$Crit_{пред}$	Критическая комбинация $Crit = Crit_{пред} + 2^{k-1}(-1)^k$	
3	-4	1	-3	1101
4	8	-3	5	0101
5	-16	5	-11	110101
6	32	-11	21	010101
7	-64	21	-43	11010101
...

Сумма кодов A и B в группе без учета переноса в группу вычисляется сумматором **InvAddK0** и сравнивается с критической

комбинацией Crit в блоке сравнения CompM. Его выход $h=1$ свидетельствует о том, что в данную группу поступила критическая комбинация слагаемых.

Сумма кодов A и B в группе с учетом переноса в группу вычисляется сумматором **InvAddK1**.

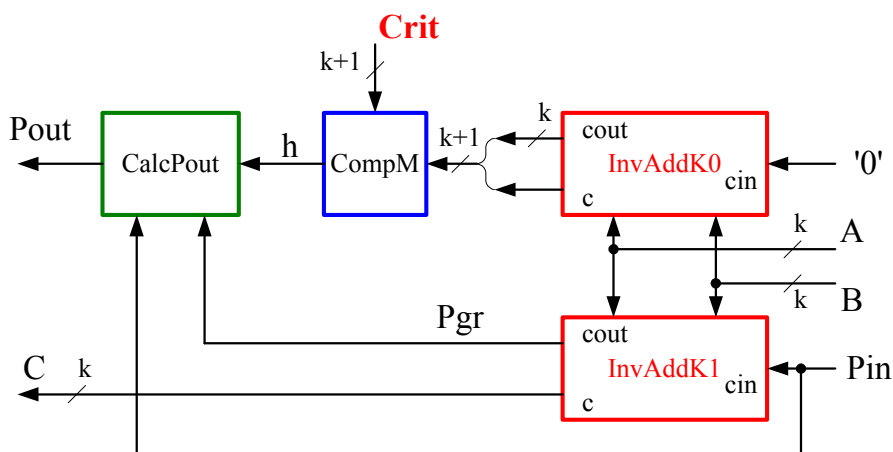


Рис. 2.

Ускорение вычислений при групповом переносе объясняется на рис. 3, где

- X - групповой блок GTC, в котором отсутствует критическая комбинация и при $Pin = 0$ возник перенос $Pout=1$,
- $Y1, Y2, Y3, Y4, \dots$ - подряд расположенные групповые блоки GTC, в которых присутствует критическая комбинация;
- Z - групповой блок GTC, в котором отсутствует критическая комбинация.

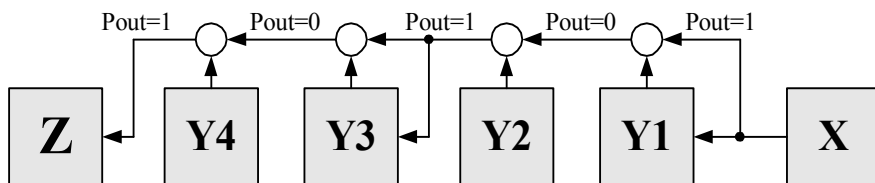


Рис. 3.

Перенос $Pout$, возникший в X , распространяется через $Y1, Y2, Y3, Y4, \dots$, меняя значение после каждого из этих блоков

(см. пояснение к блоку CalcPout1). Одновременно с этим перенос $P_{out} = 1$ поступает на входы блоков Y_1, Y_3, \dots и Z . Общая задержка распространения переносов равна задержке в сумматоре блока X , задержке в элементе CalcPout1 блоков $Y_1, Y_2, Y_3, Y_4, \dots$ и задержке в сумматоре блока Z . Полагая, что задержка в каждом разряде сумматора равна задержке в элементе CalcPout1 и равна τ , находим, что общая задержка $\vartheta = (2k + m) \cdot \tau$, где m – количество блоков в цепочке $Y_1, Y_2, Y_3, Y_4, \dots$. Отсюда следует, что максимальная задержка распространения переносов

$$\vartheta = \left(2k + \frac{N}{k} - 2 \right) \cdot \tau. \quad (1)$$

Например, при $N=27$ и $k=3$ имеем: $\vartheta = 13 \cdot \tau$, т.е. достигается ускорение в 2 раза.

2.3. Инверсный сумматор с четным групповым переносом.

Этот инверсный сумматор вычисляет $C = -A - B$. В нем используются

- инверсный сумматор k -разрядных кодов **InvAdd**;
- k -разрядный групповой блок GTC, схема которого изображена на рис. 2.

Внутри блока GTC используется в данном случае блок CalcPout при k – четном, т.е. блок CalcPout0. Критическая комбинация Crit в данном случае определяется по табл. 3.

Таблица 3.

Число анализируемых разрядов k	$2^{k-1}(-1)^k$	Crit _{пред}	Критическая комбинация $Crit = Crit_{пред} + 2^{k-1}(-1)^k$	
3	-4	1	-3	1101
4	8	-3	5	0101
5	-16	5	-11	110101
6	32	-11	21	010101
7	-64	21	-43	11010101
...

Сумма кодов A и B в группе без учета переноса в группу вычисляется сумматором **InvAddK0** и сравнивается с критической комбинацией Crit в блоке сравнения CompM. Его выход $h=1$

свидетельствует о том, что в данную группу поступила критическая комбинация слагаемых.

Сумма кодов A и B в группе с учетом переноса в группу вычисляется сумматором **InvAddK1**.

Ускорение вычислений при групповом переносе объясняется на рис. 4, где

X - групповой блок ГТС, в котором отсутствует критическая комбинация и при $P_{in}=0$ возник перенос $P_{out}=1$,

$Y1, Y2, Y3, Y4, \dots$ - подряд расположенные групповые блоки ГТС, в которых присутствует критическая комбинация;

Z - групповой блок ГТС, в котором отсутствует критическая комбинация.

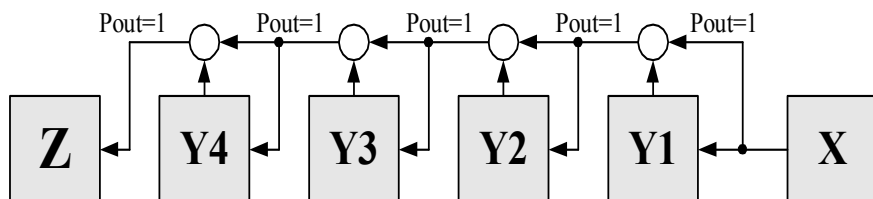


Рис. 4.

Перенос **Pout**, возникший в X , распространяется через $Y1, Y2, Y3, Y4, \dots$ (см. пояснение к блоку **CalcPout0**). Одновременно с этим перенос $P_{out}=1$ поступает на входы блоков $Y1, Y2, Y3, Y4, \dots$ и Z . Максимальная задержка распространения переносов, по-прежнему, оценивается формулой (1). Например, при $N=24$ и $k=4$ имеем: $\vartheta = 12 \cdot \tau$, т.е. достигается ускорение в 2 раза.

2.4. Инвертор с нечетным групповым переносом.

Этот инвертор вычисляет $C=-A$. В нем используются

- инвертор k -разрядных кодов **InvK**;
- k -разрядный групповой блок ГТС, схема которого изображена на рис. 5.

Внутри блока **ГТС** используется в данном случае блок **CalcPout** при k – нечетном, т.е. блок **CalcPout1**. Критическая комбинация **Crit** в данном случае содержит k разрядов и определяется по табл. 5.

Для ее вычисления не требуется дополнительной схемы. Инверсия кода A с учетом переноса в группу вычисляется инвертором **InvK**. Максимальная задержка распространения переносов вычисляется в этом случае также, как и для инверсного сумматора.

Таблица 5.

Число анализируемых разрядов k	$2^{k-1}(-1)^k$	$\text{Crit}_{\text{пред}}$	Критическая комбинация $\text{Crit} = \text{Crit}_{\text{пред}} + 2^{k-1}(-1)^k$	
3	-4	1	-3	111
4	8	-3	5	1111
5	-16	5	-11	11111
6	32	-11	21	111111
7	-64	21	-43	1111111
...

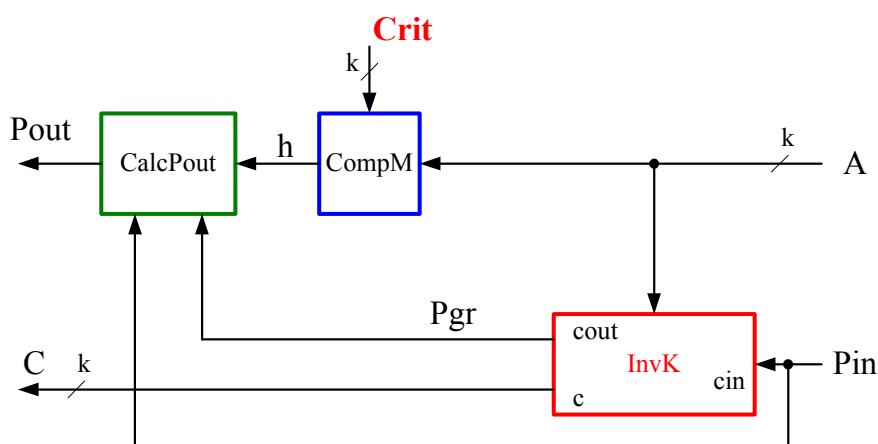


Рис. 5.

2.5. Инвертор с четным групповым переносом.

Этот инвертор вычисляет $C = -A$. В нем используются

- инвертор k -разрядных кодов **InvK**;
- k -разрядный групповой блок **GTC**, схема которого изображена на рис. 5.

Внутри блока **GTC** используется в данном случае блок **CalcPout** при k – четном, т.е. блок **CalcPout0**. Критическая комбинация Crit в данном случае содержит k разрядов и определяется по табл. 6. Для ее вычисления не требуется дополнительной схемы. Инверсия кода

A с учетом переноса в группу вычисляется инвертором **InvK**. Максимальная задержка распространения переносов вычисляется в этом случае также, как и для инверсного сумматора.

Таблица 6.

Число анализируемых разрядов k	$2^{k-1}(-1)^k$	$\text{Crit}_{\text{пред}}$	Критическая комбинация $\text{Crit} = \text{Crit}_{\text{пред}} + 2^{k-1}(-1)^k$	
3	-4	1	3	111
4	8	-3	-5	1111
5	-16	5	11	11111
6	32	-11	-21	111111
7	-64	21	43	1111111
...

Глава 5. Кодирование и декодирование

1. Устройства кодирования и декодирования действительных чисел
2. Операции кодирования и декодирования комплексных чисел

1. Устройства кодирования и декодирования действительных чисел

1.1. Прекодер Р-кода в М-код – PreCoder.

Прекодер Р-кода в М-код распределяет разряды Р-кода A на две группы – группу A_{even} четных разрядов (0, 2, 4, ...) и группу A_{odd} нечетных разрядов (1, 3, 5, ...). Таким образом, кодер образует из кода A коды A_{even} и A_{odd} . Заметим, что эти коды в дальнейшем поступают на два входа алгебраического сумматора М-кодов. Этот сумматор вычисляет либо $(A_{even} - A_{odd})$, либо $(A_{odd} - A_{even})$ в зависимости от значения некоторого управляющего сигнала $s = \{0, 1\}$ соответственно. Прекодер представлен в табл. 1, где для кода A указаны номера разрядов, а для кодов A_{even} и A_{odd} знаком «=» указаны разряды, значения которых совпадают со значением соответствующего разряда кода A . Цифрой «0» указано определенное значение разряда этих кодов.

Таблица 1.

A	...	7	6	5	4	3	2	1	0
A_{even}	...	0	=	0	=	0	=	0	=
A_{odd}	...	=	0	=	0	=	0	=	0

1.2. Кодер положительного Р-кода в М-код - CoderPM.

Этот кодер преобразует Р-код положительного числа в М-код этого числа. Его схема представлена на рис. 1, где

N - разрядность кодера,

$Meven$ – одноразрядная схема кодирования для разряда с четным номером,

$Modd$ - одноразрядная схема кодирования для разряда с нечетным номером,

A – входной Р-код,

C – выходной М-код.

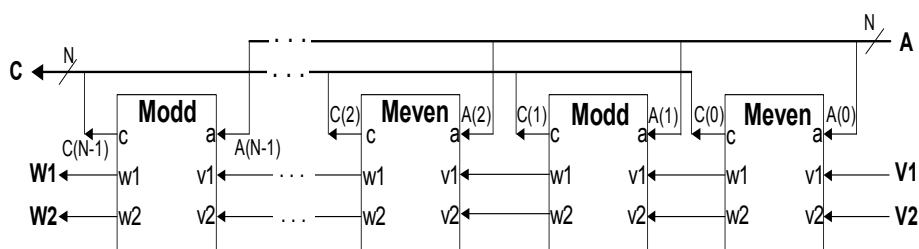


Рис. 1.

По-существу преобразование заключается в том, что из кода, составленного из четных разрядов Р-кода вычитается код, составленный из нечетных разрядов Р-кода, а вычитание выполняется по правилам вычитания М-кодов. Одноразрядные схемы $Meven$ и $Modd$ представлены на рис. 2. Их функционирование описывается таблицей истинности табл. 2. и 3 соответственно.

Таблица 2 одноразрядной схемы кодера для четного разряда

a	v1	v2	w1	w2	c
0	0	0	0	0	0
1	0	0	0	0	1
0	0	1	0	0	1
1	0	1	1	1	0
0	1	1	0	1	1
1	1	1	0	0	0

Таблица 3 одноразрядной схемы кодера для нечетного разряда

a	v1	v2	w1	w2	c
0	0	0	0	0	0
1	0	0	0	1	1
0	0	1	0	0	1
1	0	1	0	0	0
0	1	1	0	1	1
1	1	1	0	1	0

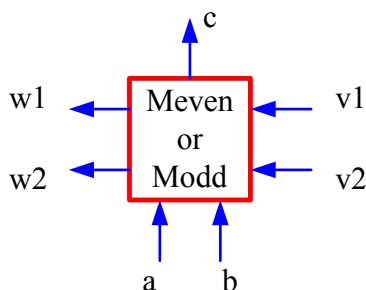


Рис. 2. Одноразрядная схема кодера.

1.3. Кодер Р-кода в М-код с добавлением - ExpCoder.

Этот кодер преобразует Р-код в М-код и прибавляет к результату «2». Как будет ясно из дальнейшего, такой кодер необходим для кодирования экспоненты, поэтому он назван ExpCoder. Рассмотрим для определенности 8-разрядный Р-код. Кодер для этого кода представлен на рис. 3.

Часть E Р-кода, а именно разряды (6,5,4,3,2,1,0), подается на вход кодера, а 7-ой знаковый разряд Р-кода $sE = \{0,1\}$ подается на управляющий вход кодера. Последний вычисляет М-код E_m по формуле

$$E_m = \begin{cases} (E_{even} - E_{odd} + 2) & \text{if } sE = 0; \\ (-E_{even} + E_{odd} + 2) & \text{if } sE = 1. \end{cases}$$

Это вычисление выполняется на блоке «Вычитатель с добавлением-nSubP». Для выделения четных и нечетных разрядов перед входами кодера включен прекодер PreCoder, который образует коды E_{even} и E_{odd} . Знаковый разряд sE Р-кода является кодом операции cor для вычитателя nSubP. Разрядность М-кода E_m равна 10, а

разрядность части E Р-кода равна 7. Поэтому переполнение при кодировании отсутствует.

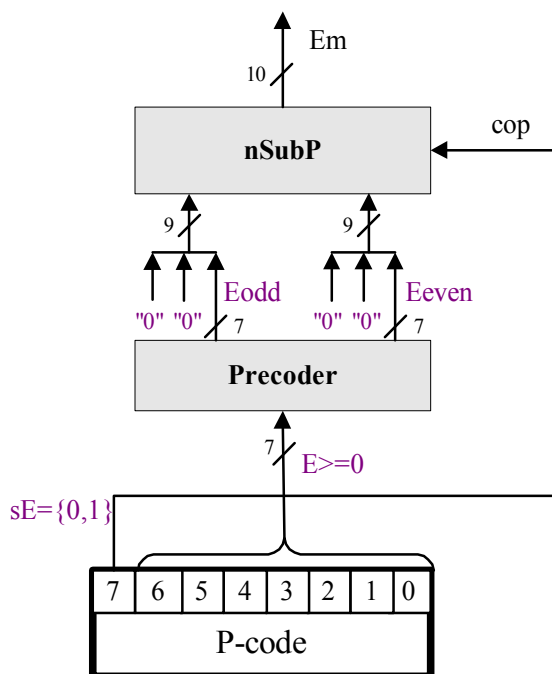


Рис. 3.

1.4. Декодер М-кода в Р-код – DecoderMP.

Этот декодер преобразует М-код некоторого числа в Р-код этого числа. Его схема представлена на рис. 4, где

N – разрядность декодера,

$Deven$ – одноразрядная схема декодирования для разряда с четным номером,

$Dodd$ – одноразрядная схема декодирования для разряда с нечетным номером,

A – входной М-код,

C – выходной Р-код,

COP – код операции,

W – выходной перенос.

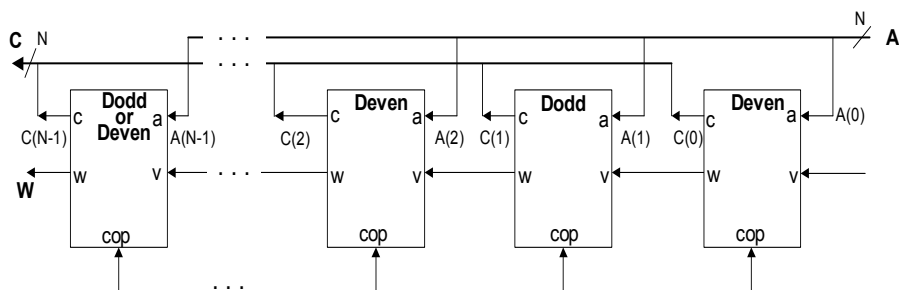


Рис. 4.

По-существу, преобразование заключается в том, что

- если М-код представляет *положительное* число, то из кода, составленного из *четных* разрядов М-кода вычитается код, составленный из *нечетных* разрядов М-кода (в этом случае $\text{cop}=0$),
- если М-код представляет *отрицательное* число, то из кода, составленного из *нечетных* разрядов М-кода вычитается код, составленный из *четных* разрядов М-кода (в этом случае $\text{cop}=1$), а вычитание выполняется по правилам вычитания Р-кодов.

Одноразрядные схемы Deven и Dodd представлены на рис. 4. Их функционирование описывается таблицей истинности табл. 4. и 5 соответственно.

Таблица 4 одноразрядной схемы декодера для четного разряда

	cop	a	v	w	c
Even - Odd	0	0	0	0	0
	0	1	0	0	1
	0	0	1	1	1
	0	1	1	0	0
Odd - Even	1	0	0	0	0
	1	1	0	1	1
	1	0	1	1	1
	1	1	1	1	0

Эта таблица вычисляет $(-2w+c)$ $\left\{ \begin{array}{l} (a-v), \text{ если } \text{cop} = 0 \\ (-a-v), \text{ если } \text{cop} = 1 \end{array} \right.$

Таблица 5 одноразрядной схемы декодера для нечетного разряда

	cop	a	v	w	c
Even - Odd	0	0	0	0	0
	0	1	0	1	1
	0	0	1	1	1
	0	1	1	1	0
Odd - Even	1	0	0	0	0
	1	1	0	0	1
	1	0	1	1	1
	1	1	1	0	0

Эта таблица вычисляет $(-2w+c) = \begin{cases} (-a-v), & \text{если } \text{cop} = 0 \\ (a-v), & \text{если } \text{cop} = 1 \end{cases}$

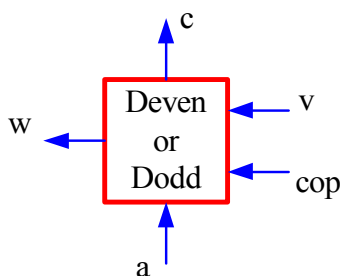


Рис. 5. Одноразрядная схема декодера.

1.5. Блок декодирования М-кода в Р-код – mDecoderMP

Декодер mDecoderMP управляется кодом операции cop. В связи с этим блок декодирования в целом должен (кроме описанного выше декодера DecoderMP) содержать еще и знакоопределитель М-кода nSign. При этом схема блока декодирования принимает вид рис. 6.

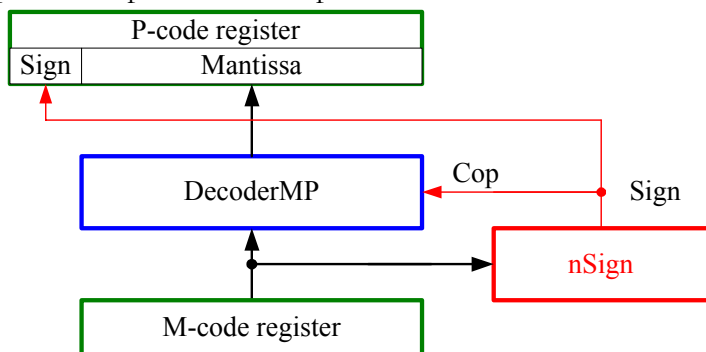


Рис. 6. Блок декодирования.

1.6. Распределитель частей кода – Partitioning.

Распределитель частей кода преобразует действительную и мнимую части комплексного кода A в два М-кода действительных чисел $\text{Re}A$ и $\text{Im}A$. При этом мнимая часть передается со сдвигом на 1 разряд влево. Распределитель кода представлен в табл. 6, где для кода A указаны номера разрядов, а для кодов $\text{Re}A$ и $\text{Im}A$ указаны номера разрядов кода A , которые перемещаются распределителем в данный разряд.

Таблица 6.

A	...	7	6	5	4	3	2	1	0
$\text{Re}A$...	14	12	10	8	6	4	2	0
$\text{Im}A$...	15	13	11	9	7	5	3	1

2. Операции кодирования и декодирования комплексных чисел

2.0. Введение

Рассмотрим операции кодирования и декодирования, которые выполняются с использованием вышеописанных блоков. Эти операции перечислены в табл. 1. Далее эти операции описываются подробнее.

Таблица 1.

№	Обозначение	Наименование
1	CodingInt	Кодирование пары целых Р-кодов в целый С-код.
2	DecodingInt	Декодирование целого С-кода в пару целых Р-кодов
3	CodingFloatTwo	Кодирование пары Р-кодов в пару С-кодов
4	DecodingFloat	Декодирование С-кода в пару Р-кодов
5	CodingFloat	Кодирование пары Р-кодов в С-код
7	FloatToWholeFraction	Выделение целой и дробной части С-кода
8	FloatToWhole	Выделение целой части С-кода
9	FloatToFraction	Выделение дробной части С-кода
10	ExpToInt	Преобразование экспоненты в целый С-код
11	ExpToFloat	Преобразование экспоненты в С-код
12	RealToExp	Преобразование целого С-кода в экспоненту
13	FloatToInt	Преобразование С-кода «from Float to Integer»
14	IntToFloat	Преобразование С-кода «from Integer to Float»

2.1. Кодирование пары целых Р-кодов в целый С-код - CodingInt

Кодирование Р-кода в М-код выполняется на сумматоре, который выполняет кодирование сразу двух Р-кодов одновременно. Для этого два Р-кода преобразуются блоком **ExpanderTwoTradInt** в четыре слагаемых М-кода или, что одно и то же, в два слагаемых С-кода. В схеме преобразования выполняется прием только 25-ти младших разряда целого Р-кода (разряды 24, 23,..., 2, 1, 0). Остальные старшие разряды игнорируются (предполагается, что они равны 0). Знаки целого Р-кода принимаются из старшего 31-го разряда. Эти знаки и определяют вид алгебраического сложения. В результате образуется целый комплексный код. Обозначим:

S_m - знак мантииссы Р-кода (разряд 31),

A_{even} - код четных разрядов кода А,

A_{odd} - код нечетных разрядов кода А,

B_{even} - код четных разрядов кода В,

B_{odd} - код нечетных разрядов кода В.

Блок **ExpanderTwoTradInt** изображен на рис. 1. Он формирует из этих кодов два С-кода - **OutEint1**, **OutEint2**. При этом

$$\text{Re}(\text{OutEint1}) = A_{even},$$

$$\text{Im}(\text{OutEint1}) = B_{even},$$

$$\text{Re}(\text{OutEint2}) = A_{odd},$$

$$\text{Im}(\text{OutEint2}) = B_{odd}.$$

После сложения кодов **OutEint1** и **OutEint2** в регистре результата образуются М-коды чисел А и В, причем М-код числа А находится в четных разрядах результирующего кода, а М-код числа В находится в нечетных разрядах результирующего кода. Знак операции определяется по следующему правилу (например, для А):

$$A = A_{even} - A_{odd}, \text{ if } S_m = 1,$$

$$A = -A_{even} + A_{odd}, \text{ if } S_m = -1.$$

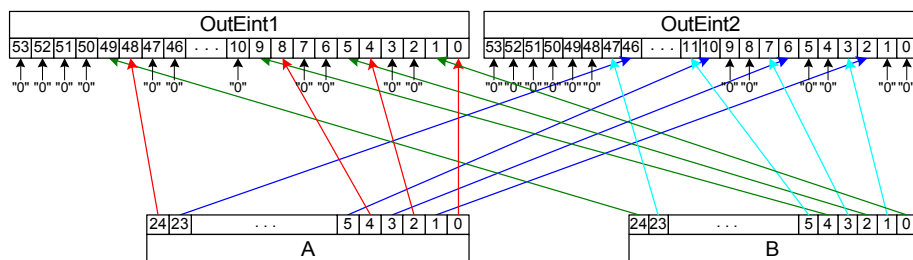


Рис. 1.

2.2. Декодирование целого комплексного кода в пару целых Р-кодов - **DecodingInt**

Для декодирования используется декодер **DecoderInt** - см. рис. 2, где

Im, Re – мнимая и реальная части комплексного кода,

DecSim – знак Р-кода мнимой части,
 DecSre – знак Р-кода реальной части,
 outDPIIm – мантисса мнимой части,
 outDPIRe – мантисса реальной части,
 Sim – знак мнимой части,
 Sre – знак реальной части,
 inReg – комплексный код.
 outDiIm – Р-код мнимой части,
 outDiRe – Р-код реальной части.

Целый комплексный С-код разбивается в блоке Partitioning на два М-кода реальной и мнимой части. Каждая из этих частей декодируется в отдельном декодере М-кода DecoderMP. На управляющие входы этих декодеров подаются знаки соответствующих частей комплексного кода. Результаты декодирования в виде Р-кодов записываются в выходные регистры. При этом заполняются только 32 младших разряда этих регистров.

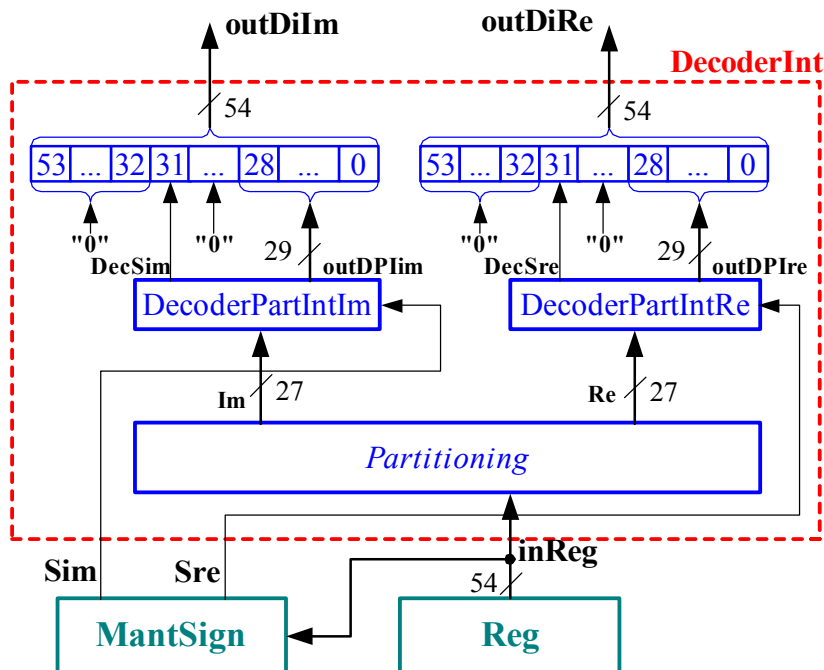


Рис. 2.

2.3. Кодирование пары Р-кодов в пару С-кодов - CodingFloatTwo

Кодирование мантисс Р-кода в М-код выполняется на сумматоре, который выполняет кодирование сразу двух Р-кодов одновременно. Для этого два Р-кода преобразуются блоком ExpanderTwoTradFloat в четыре слагаемых М-кода. Для предотвращения переноса в целую часть кода исходные коды сдвигаются в этом блоке на два разряда вправо. При этом экспонента числа увеличивается на 2. Обозначим:

M - мантисса Р-кода (разряды 22-0),

E - экспонента Р-кода (разряды 29-23),

S_m - знак мантиссы Р-кода (разряд 31),

S_e - знак экспоненты Р-кода (разряд 30).

L_e - значение младшего разряда экспоненты Р-кода (разряд 23).

A_{even} - код четных разрядов кода А,

A_{odd} - код нечетных разрядов кода А,

B_{even} - код четных разрядов кода В,

B_{odd} - код нечетных разрядов кода В.

Блок ExpanderTwoTradFloat изображен на рис. 3. Он формирует из этих кодов два С-кода - OutEfloat1, OutEfloat2. При этом

$$\text{Re}(\text{OutEfloat1}) = A_{even},$$

$$\text{Im}(\text{OutEfloat1}) = B_{even},$$

$$\text{Re}(\text{OutEfloat2}) = A_{odd},$$

$$\text{Im}(\text{OutEfloat2}) = B_{odd}.$$

До преобразования каждый Р-код представлял величину $2^{S_e \cdot E} \cdot S_m \cdot M$, а после кодирования каждый М-код представляет величину $(-2)^{(E+2)}(-1)^{L_e} S_m \frac{A}{4}$. Таким образом,

- мантисса перед кодированием сдвигается на два разряда вправо (сдвинутые разряды не пропадают, т.к. мантисса М-кода имеет разрядность 27, а мантисса Р-кода имеет разрядность 23);

- при кодировании мантисса умножается на величину $(-1)^{L_e} S_m = \pm 1$;
- экспонента при кодировании увеличивается на 2.

После сложения кодов OutEfloat1 и OutEfloat2 в регистре результата образуются М-коды чисел А и В, причем М-код числа А находится в четных разрядах результирующего кода, а М-код числа В находится в нечетных разрядах результирующего кода. Знак операции определяется по следующему правилу (например, для А):

$$A = A_{even} - A_{odd}, \text{ if } (-1)^{L_e} S_m = 1,$$

$$A = -A_{even} + A_{odd}, \text{ if } (-1)^{L_e} S_m = -1.$$

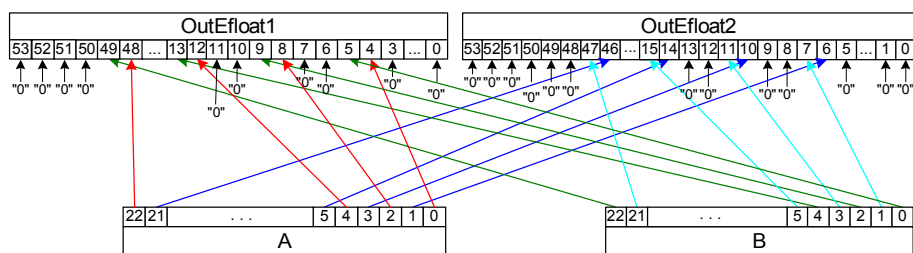


Рис. 3.

2.4. Декодирование С-кода в пару Р-кодов - DecoderFloat

Для декодирования используется декодер DecoderFloat - см. рис. 4, где

Im, Re – мнимая и реальная части комплексного кода,
 DecSim – предварительный знак Р-кода мнимой части,
 DecSre – предварительный знак Р-кода реальной части,
 TrSim – знак Р-кода мнимой части,
 TrSre – знак Р-кода реальной части,
 outDPIim – мантисса мнимой части,
 outDPIre – мантисса реальной части,
 DecS – знак экспоненты Р-кода,
 OutDE –экспонента Р-кода.,
 Sim – знак мнимой части,
 Sre – знак реальной части,
 inReg – комплексный код,

inExp – экспонента комплексного кода,
 S – знак экспоненты Р-кода.
 outDfIm – Р-код мнимой части,
 outDfRe – Р-код реальной части,
 V – сигнал переполнения.

Комплексный С-код в блоке Partitioning разбивается на два М-кода реальной и мнимой части. Каждая из этих частей декодируется декодерами DecoderPartFloatRe и DecoderPartFloatIm, выполненными в виде mDecoderMP. На управляющие входы этих декодеров подаются знаки соответствующих частей комплексного кода из блока MantSign. Результаты декодирования в виде Р-кодов записываются в выходные регистры.

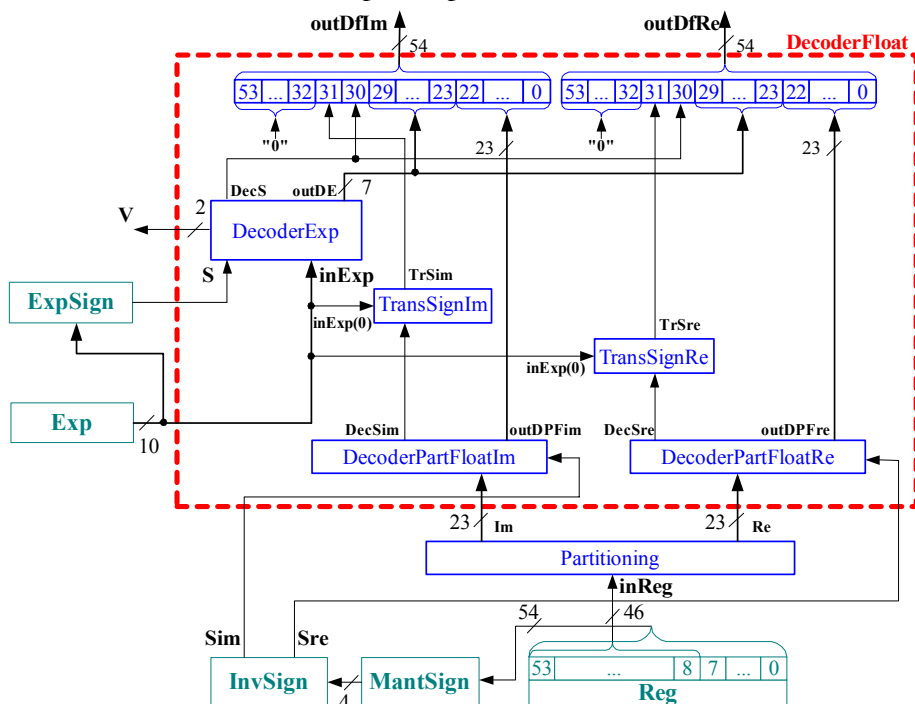


Рис. 4.

Экспонента комплексного кода как М-код декодируется в отдельном декодере экспоненты **DecoderExp**. На управляющий вход этого декодера подается знак экспоненты комплексного кода из блока **ExpSign**. Результат декодирования в виде Р-кода записываются одновременно в два выходных регистра. Знаковые разряды Р-кода образуются в двух преобразователях знака

TransSignRe и TransSignIm. При декодировании экспоненты может возникнуть переполнение, т.к. разрядность экспоненты комплексного кода превышает разрядность экспоненты традиционного кода.

Рассмотрим блок TransSign. На его входе присутствуют

- предварительный знак части (реальной или мнимой) мантиисы DecS, полученный в декодере mDecoderMP,
- младший бит экспоненты С-кода inExp(0).

Знак Р-кода части (реальной или мнимой) мантиисы определяется в блоке TransSign по формуле $TrS = \text{Xor}(\text{DecS}, \text{inExp}(0))$.

2.5. Кодирование пары Р-кодов в С-код - CodingFloat

Такое кодирование выполняется в два этапа:

- кодирование пары Р-кодов в пару С-кодов
- сложение пары С-кодов, полученных на первом этапе.

Второй этап необходим, т.к. пара комплексных кодов, полученных на первом этапе, может иметь различные экспоненты.

2.7. Выделение целой и дробной части из комплексного кода - FloatToWholeFraction

В табл. 7 рассматривается код мантиисы одной из частей комплексного кода. Она представляет собой код действительного числа по основанию (-2). В таблице указана степень основания при различных значениях экспоненты. Точка, разделяющая целую и дробную части, располагается после разряда с нулевым весом. В таблице цветом выделены целые и дробные части.

Таблица 7.

Экспонента	Степень основания (-2)									
0	<i>k</i>	-1	-2	-3		-E		-k		-27
<i>E</i> > 0	<i>E</i> + <i>k</i>	E-1	E-2	E-3		0		E-k		E-27

Отсюда следует, что в коде с нулевой экспонентой нет целой части. В коде с экспонентой $E > 0$ целая часть содержится в левых E разрядах, а дробная часть содержится в правых $(27-E)$ разрядах. Таким образом, алгоритм выделения целой и дробной части имеет следующий вид:

- При $27 > E > 0$ целая часть образуется сдвигом исходной мантиссы на $(27-E)$ разрядов вправо; дробная часть образуется сдвигом исходной мантиссы на E разрядов влево; экспонента дробной части равна нулю.
- При $E = 27$ целая часть равна исходному коду, а дробная часть равна нулю; экспонента дробной части также равна нулю.
- При $E \leq 0$ целая часть отсутствует, а дробная часть равна исходному числу.
- При $E > 27$ выделение целой и дробной частей невозможно (переполнение целой части).

2.8. Выделение целой части из С-кода - FloatToWhole

Эта операция является частным случаем операции FloatToWholeFraction и выполняется по следующему алгоритму:

- При $27 > E > 0$ целая часть образуется сдвигом исходной мантиссы на $(27-E)$ разрядов вправо.
- При $E = 27$ целая часть равна исходному коду.
- При $E \leq 0$ целая часть отсутствует.
- При $E > 27$ выделение целой и дробной частей невозможно (переполнение целой части).

2.9. Выделение дробной части из С-кода - FloatToFraction

Эта операция является частным случаем операции FloatToWholeFraction и выполняется по следующему алгоритму:

- При $27 > E > 0$ дробная часть образуется сдвигом исходной мантиссы на E разрядов влево; экспонента дробной части равна нулю.
- При $E = 27$ дробная часть равна нулю; экспонента дробной части также равна нулю.
- При $E \leq 0$ дробная часть равна исходному числу.
- При $E > 27$ выделение целой и дробной частей невозможно (переполнение целой части).

2.10. Преобразование экспоненты в целый С-код - ExpToInt

Это преобразование заключается в том, что код экспоненты записывается в младшие разряды действительной части кода мантиссы.

2.11. Преобразование экспоненты в С-код - ExpToFloat

Вначале экспонента преобразуется в целый комплексный код, а затем целый комплексный код преобразуется в комплексный код Float. Оба этих преобразования рассмотрены выше.

2.12. Преобразование действительного целого числа в экспоненту - RealToExp

Это преобразование заключается в том, что 10 младших разрядов действительной части кода мантииссы записываются в код экспоненты.

2.13. Преобразование С-кода “from Float to Integer” - FloatToInt

Это преобразование эквивалентно операции FloatToWhole - выделению целой части из С-кода

2.14. Преобразование С-кода “from Integer to Float” - IntToFloat

Вначале такого преобразования в регистр экспоненты С-кода записывается число 27 – разрядность частей мантииссы. После этого целый С-код в регистре мантииссы и код в регистре экспоненты могут рассматриваться как комплексный код с прежним значением. Далее выполняется нормализация полученного кода, т.к. несколько старших разрядов целого кода могли быть нулевыми.

Глава 6. Специальные логические операции

1. Нормализация
2. Выравнивание экспонент 9
3. Сдвигатели
4. Компараторы

1. Нормализация

В результате некоторой арифметической операции может возникнуть ненормализованный код или переполнение разрядной сетки. При переполнении С-кода могут появиться еще 4 разряда в целой части кода – см. также описание алгебраического сумматора С-кодов. Структура мантиссы С-кода с переполнением показана на рис. 1, где во второй строке указана нмерация разрядов, а в первой строке – нумерация пар разрядов. Пара разрядов является значащей, если хотя бы один разряд в ней равен «1».

1		2		3		4		...	28		29	
57	56	55	54	53	52	51	50	...	3	2	1	0
Overflow				Mantissa								

Рис. 1.

В табл. 1 указан характер денормализации в зависимости от номера старшей значащей пары. В этой таблице

N - номер значащей пары разрядов,

Msu1_out – признак характера денормализации.

Msign – признак направления сдвига при нормализации

Таблица 1.

N	Msu1_out	Характер денормализации	Msign	Характер сдвига
0	30	Нулевой код	“10”	Нулевой код
1	2	Переполнение	“01”	Сдвиг вправо
2	1	Переполнение	“01”	Сдвиг вправо
3	0	Нормализованный код	“00”	Нет сдвига
>3	N-3	Ненормализованный код	“11”	Сдвиг влево на $(2*(N-3))$ разрядов влево

На рис. 2 и 3 представлены условная и полная схемы анализатора старшей значащей пары разрядов – MSU (Most Significant Unit). При этом приняты следующие обозначения:

DetMSU - определитель номера N,

MSU1 - указатель Msu1_out,

MSU2 - указатель Msign,

CoderPMN - преобразователь Р-кода в М-код Msu3_out,

SummatorMN – сумматор М-кодов с выходом Msu4_out,

InMsu – входной нормализуемый код.

MSU определяет номер N старшей пары значащих разрядов и преобразует его в несколько выходных кодов. Наличие «1» в одном из разрядов, объединенных в пару, определяются с помощью элементов OR. Выходы всех этих элементов соединены со входом блока CoderMSU. Выход последнего определяет номер старшего элемента OR с единичным сигналом на выходе. Этот номер образуется в виде Р-кода. Другими словами, он определяет номер N старшей пары разрядов, в одном из которых имеется «1». Номер N на выходе CoderMSU принимает значения от 0 до 29 и поступает на входы трех других блоков – MSU1, MSU2, CoderPMN. Последний вычисляет М-код числа N. Сумматор SummatorMN вычисляет число $(N-3)$ разрядов, на которые нужно сдвинуть нормализуемый код.

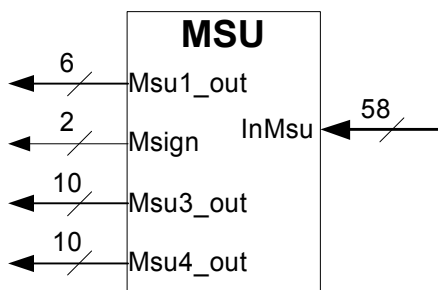


Рис. 2.

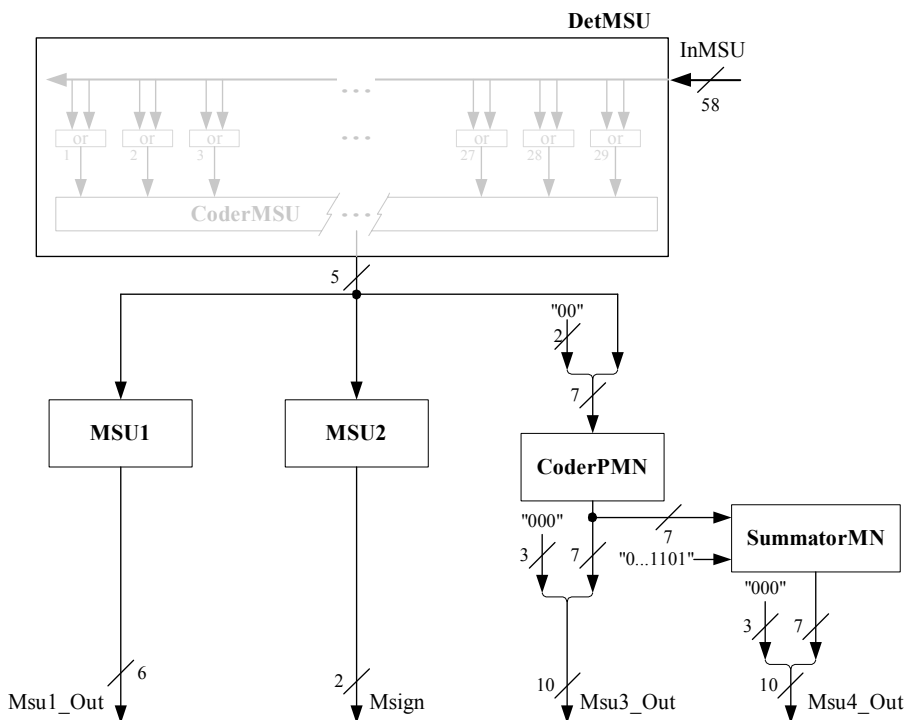


Рис. 3.

Рассмотрим теперь блок анализа для нормализации AnlNormal. Он описывается табл. 2, где

VExp - состояние регистра переполнения,

OutNT - код действия.

Таблица 2.

Msign	VExp	OutNT	Действие
0 0	* *	0 0 0	Нормализация не нужна
1 0	* *	0 0 1	Нулевой результат
0 1	0 0	0 1 0	Сдвиг вправо из-за переполнения
1 1	0 0	0 1 1	Сдвиг влево из-за денормализации
* 1	1 1	0 0 1	Нулевой результат
* 1	0 1	1 0 0	Переполнение

2. Выравнивание экспонент

Перед алгебраическим сравнением необходимо выровнять экспоненты слагаемых. Обозначим:

A, B –слагаемые C -коды,

C – результирующий C -код,

mA, mB, mC – мантиссы кодов A, B, C соответственно,

eA, eB, eC – экспоненты кодов A, B, C соответственно,

V_{exp} – признак переполнения при вычитании экспонент,

причем $V_{exp}=(0,1,-1)$ и M -код

$K(V_{exp})=(00,01,11)$ соответственно,

eAB – разность $(eA-eB)$, представленная в M -кодом (7 бит)

$shiftal$ – количество разрядов, на которые нужно сдвинуть одно из слагаемых (5 бит),

$align$ – тип выравнивания.

В табл. 1 перечислены возможные варианты.

Таблица 1.

V_{exp}	eAB	$shiftal$	$align$	выравнивание
0	$eAB > 26$	0	001	$eC=eA, mB=0$
0	$eAB < -26$	0	000	$eC=eB, mA=0$
0	$eAB = 0$	0	100	$eC=eA$
0	$eAB \geq -26$ and $eAB \leq -1$	$2 \cdot eAB$	011	$eC=eB, mA \rightarrow shift$
0	$eAB \leq 26$ and $eAB \geq 1$	$2 \cdot eAB$	010	$eC=eA, mA \rightarrow shift$
1	-	0	001	$eC=eA, mB=0$
-1	-	0	000	$eC=eB, mA=0$

Блок вычисления признака выравнивания $AlignmentAB$ показан на рис. 1. Он содержит декодер $DecoderM$ M -кода разности eAB и определитель знака этого числа $ExpSign$. На их выходах присутствуют P -код модуля $|eAB|$ и знак $sign(eAB)$. Кроме того, блок $AlignmentAB$ содержит компаратор $AlignmentComp$, который вырабатывает признаки $alignComp0=1$ при выполнении условия $|eAB|=0$ и $alignComp26=1$ при выполнении условия $|eAB|>26$. В

результате блок AlignmentAB вырабатывает $shiftal = |eAB|$ и в блоке TabAlign реализует табл. 2 для получения сигнала align.

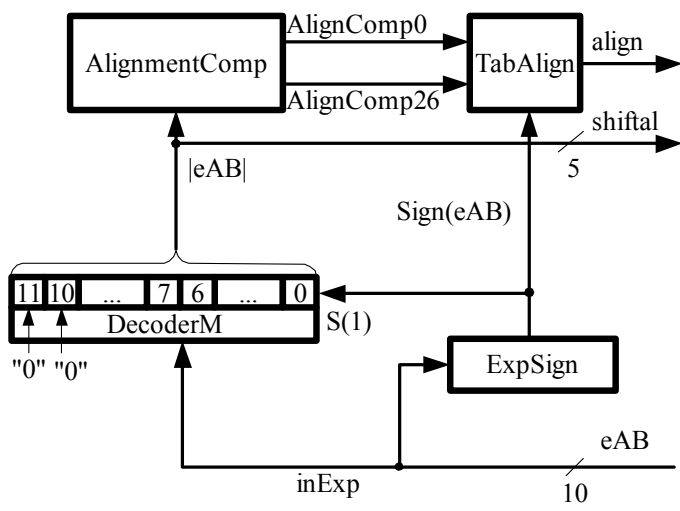


Рис. 1.

Таблица 2.

Vexp	sign(eAB)	AlignComp0	AlignComp26	align
0	1	0	1	001
0	-1	0	1	000
0	1	1	0	100
0	-1	0	0	011
0	1	0	0	010
1	-	-	-	001
-1	-	-	-	000

3. Сдвигатели

3.1. Сдвигатель мантиссы вправо – ShiftRigth

Условная схема сдвигателя мантиссы вправо представлена на рис. 1, а полная схема - на рис. 2. При этом приняты следующие обозначения:

A – сдвигаемый код,

D – управляющий сигнал,

Q – результат.

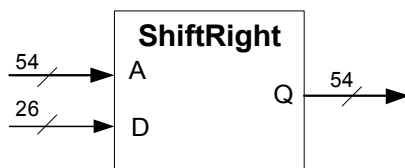


Рис.1

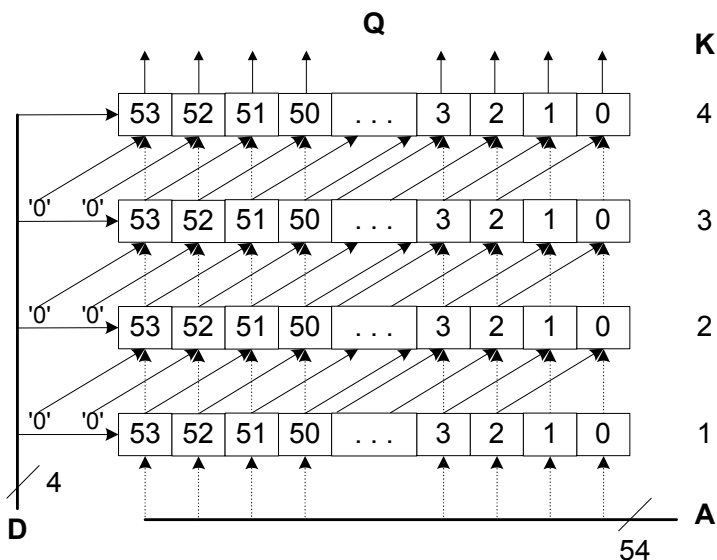


Рис.2

Сдвигатель состоит из 26 линейных схем. Каждая линейная K-схема ($K=1,2,3,4,\dots$) передает входной код в следующую линейную (K+1)-схему. Передача выполняется без изменения или со сдвигом на (2^K) разрядов вправо в зависимости от значения управляющего сигнала. Управляющим сигналом является значение одного из

разрядов управляющего регистра *D*. Каждая линейная схема состоит из 54 одноразрядных схем. На рис. 3 представлена одноразрядная схема и таблица истинности, описывающая ее функционирование. На этой фигуре

- s** - управляющий сигнал,
- a** - значение одноименного разряда предыдущей схемы,
- q** - значение разряда предыдущей схемы,
- r** - выходной сигнал.

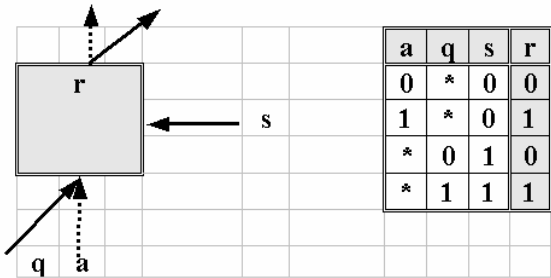


Рис. 3.

Для сдвига на $2 \cdot M$ разрядов вправо в управляющий регистр *D* должен быть записан соответствующий код *OutRomShift*. В табл. 1 указано это соответствие.

Таблица 1.

M	OutRomShift
1	10000000000000000000000000000000
2	11000000000000000000000000000000
3	11100000000000000000000000000000
24	11111111111111111111111111111000
25	11111111111111111111111111111100
26	11111111111111111111111111111110

3.2. Сдвигатель мантиссы влево – ShiftLeft

Этот сдвигатель аналогичен предыдущему. Отличие заключается только в схеме соединения линейных схем – см. рис. 4.

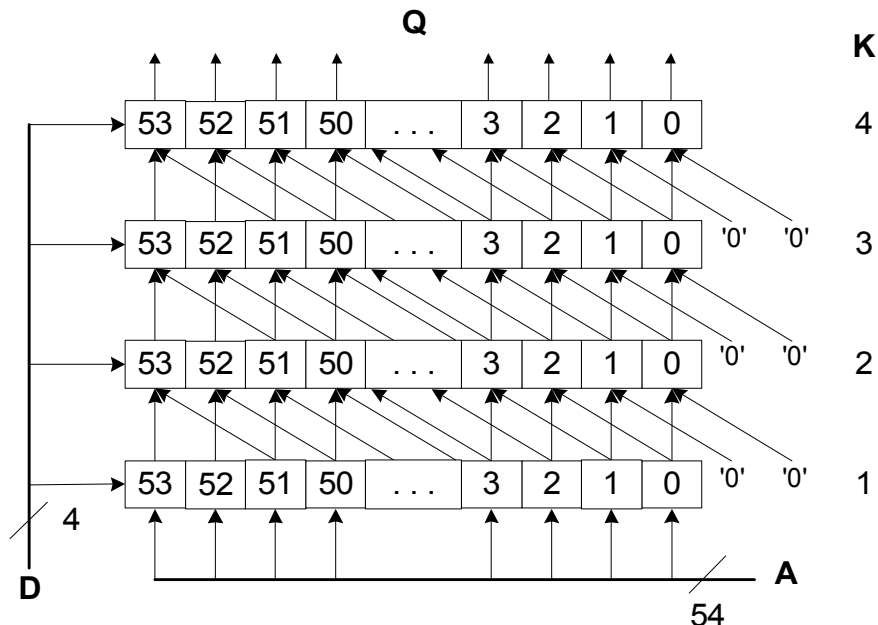


Рис. 4

3.3. Сдвигатель мантиссы вправо - ShFastR.

На рис. 5 представлена схема ShFastR. При этом приняты следующие обозначения:

Mux1, Mux2, Mux4, Mux8, Mux16, MuxOut –
мультиплексоры,

ShR1 - сдвигатель на пару разрядов вправо,

ShR2 - сдвигатель на 2 пары разрядов вправо,

ShR4 - сдвигатель на 4 пары разрядов вправо,

ShR8 - сдвигатель на 8 пар разрядов вправо,

ShR16 - сдвигатель на 16 пар разрядов вправо.

A – сдвигаемый код (54 разряда),

D – управляющий сигнал (14 бит).

Q - результат (54 разряда).

Разряды управляющего кода D открывают определенные информационные входы мультиплексоров. Коды D построены таким образом, что передача исходного кода A от мультиплексора к мультиплексору приводит в конечном счете к сдвигу кода A на необходимое число разрядов (от 0 до 27). Для сдвига на $2 \cdot M$ разрядов вправо в управляющий регистр D должен быть записан

соответствующий код OutRomShiftFast. В табл. 2 указано это соответствие.

Таблица 2.

М	OutRomShiftFast
1	01000000000001
2	01100000000010
3	01100000000101
25	11010101000001
26	11010101100010
27	11010101100101

3.4. Сдвигатель мантиссы влево - ShFastL.

На рис. 6 представлена схема этого сдвигателя, где

ShL1 - сдвигатель на пару разрядов влево,

ShL2 - сдвигатель на 2 пары разрядов влево,

ShL4 - сдвигатель на 4 пары разрядов влево,

ShL8 - сдвигатель на 8 пар разрядов влево,

ShL16 - сдвигатель на 16 пар разрядов влево.

В остальном этот сдвигатель аналогичен предыдущему.

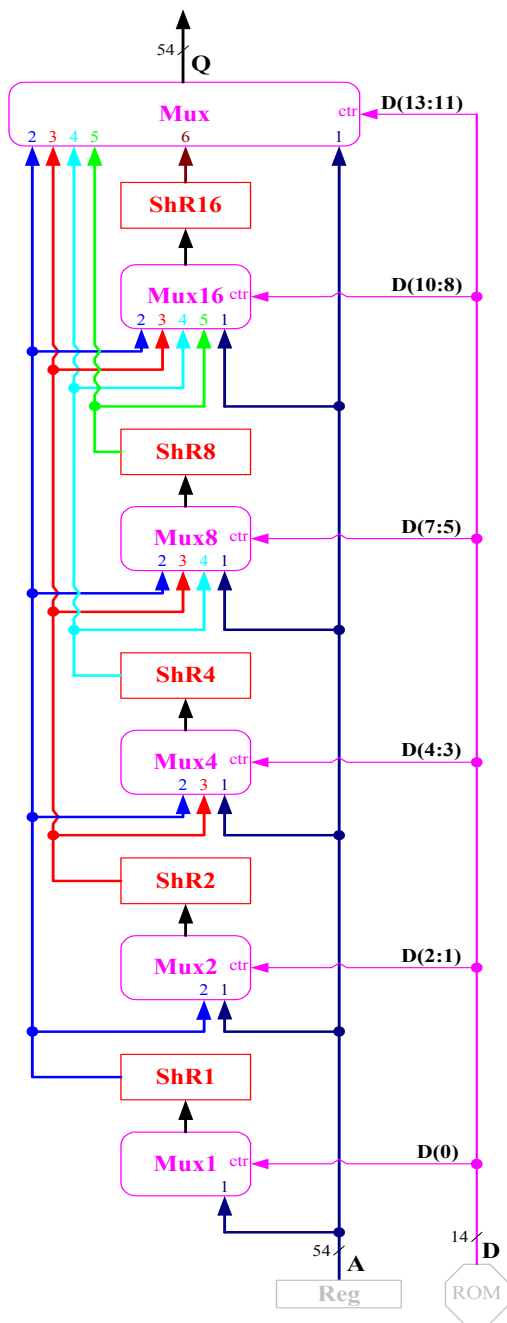


Рис. 5.

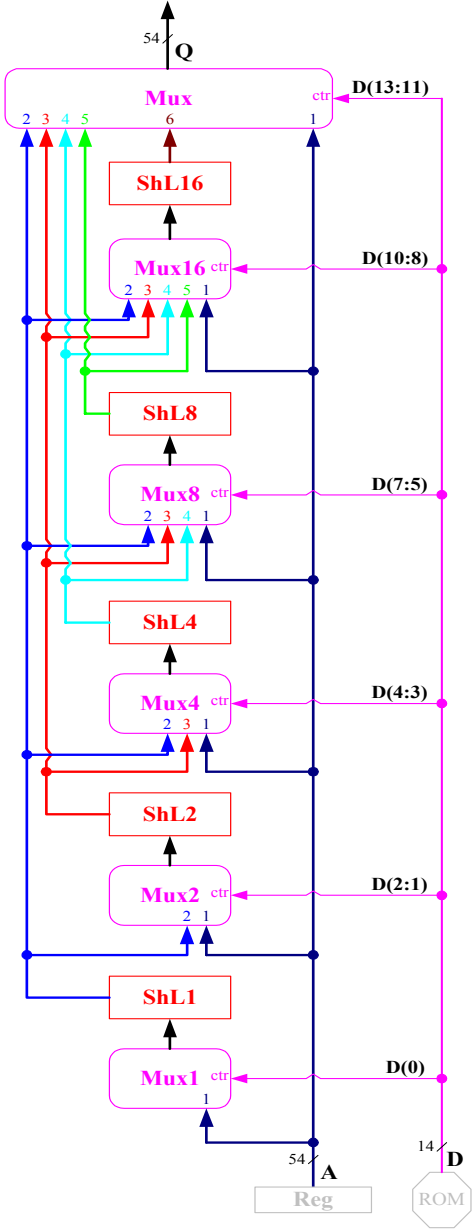


Рис. 6.

4. Компараторы

4.1. Блок сравнения С-кодов по модулю – CompModLfull.

На рис. 1 представлена схема блока сравнения С-кодов по модулю, где

A, B – входы, L -разрядные С-коды,

OutCompMod – выход (2 бита),

CalcModulLA – вычислитель квадрата модуля A ,

CalcModulLB – вычислитель квадрата модуля B .

Вычислитель квадрата модуля описан в разделе 5.2.5. Блок IfThen реализует правило:

if OutA > OutB then OutCompMod = 01

if OutA < OutB then OutCompMod = 11

if OutA = OutB then OutCompMod = 00

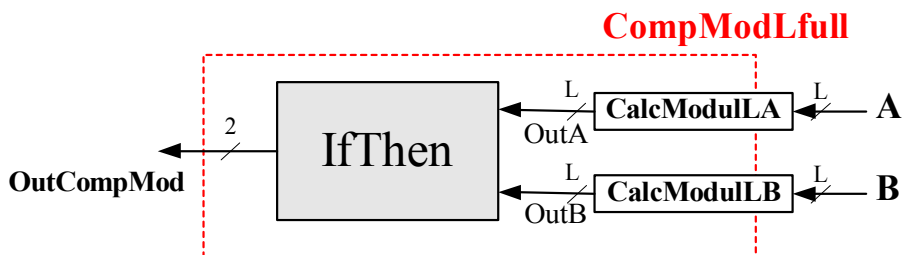


Рис. 1.

4.2. Блок сравнения М-кодов по длине – CompLen.

Сравнение М-кодов по длине заключается в следующем:

- сравниваются старшие разряды двух кодов;
- если они равны, то выполняется переход к следующей паре;
- если они не равны, то выявляется тот код, в котором сравниваемый разряд равен «1».

На рис. 2 представлена представлена одноразрядная схема Cmp блока CompLen, а табл. 1 является ее таблицей истинности и описывает указанный алгоритм. При этом приняты следующие обозначения:

a, b - разряды сравниваемых кодов,

$v1, v2$ - входные переносы,

$w1, w2$ - выходные переносы.

Код переносов ' $w2 w1$ ' интерпретируется следующим образом:

- 00 – коды сравнимы по длине,
- 01 – $a > b$ по длине,
- 11 – $a < b$ по длине.

Табл. 1 вычисляет код ‘w2 w1’ следующим образом:

‘w2 w1’ = $\left\{ \begin{array}{l} \text{‘v2 v1’ if ‘v2 v1’} \neq \text{‘00’} \\ \text{‘00’ if ‘v2 v1’} = \text{‘00’ and } a = b \\ \text{‘01’ if ‘v2 v1’} = \text{‘00’ and } a > b \\ \text{‘11’ if ‘v2 v1’} = \text{‘00’ and } a < b \end{array} \right.$

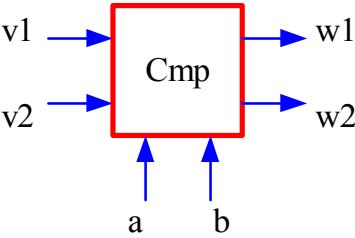


Рис. 2. Одноразрядная схема блока CompLen.

Таблица 1.

a	b	v2	v1	w2	w1
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	0	0	1
1	1	0	0	0	0
0	0	0	1	0	1
0	1	0	1	0	1
1	0	0	1	0	1
1	1	0	1	0	1
0	0	1	1	1	1
0	1	1	1	1	1
1	0	1	1	1	1
1	1	1	1	1	1

Полная схема CompLen. представлены на рис. 3, где
A, B – N-разрядные входные коды,
V1, V2 – входной перенос, равный нулю,
W1, W2 – выходной перенос.

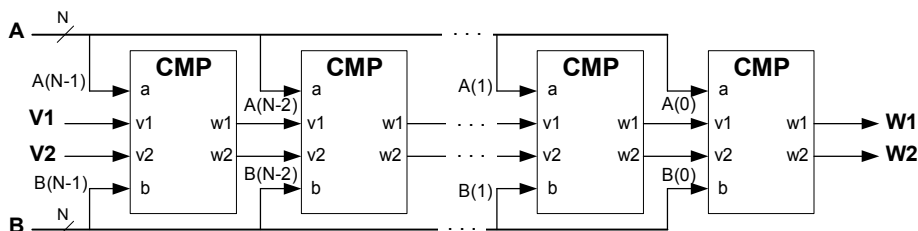


Рис. 3.

Выходной перенос интерпретируется следующим образом:

$W1\ W2 = '00'$ if $A = B$

$W1\ W2 = '01'$ if $A > B$

$W1\ W2 = '11'$ if $A < B$

Можно заметить, что сравнение М-кодов по длине равносильно сравнению по модулю чисел, представленных М-кодами.

4.3. Операции сравнения

Рассмотрим операции сравнения, которые могут быть выполнены на описанных выше устройствах.

4.3.1. Сравнение М-кодов по длине - CompareLen

Эта операция выполняется непосредственно блоком CompLen.

4.3.2. Сравнение С-кодов по модулю - CompareMod

Эта операция заключается в сравнении модулей двух комплексных чисел, представленных кодами мантисс. Операция выполняется непосредственно блоком CompModulLong.

4.3.3. Сравнение по длине кодов действительных и мнимых частей - CompareLenReIm

Операция выполняется непосредственно блоком CompLen, на входы которого подаются части С-кода.

4.3.4. Сравнение по модулю действительных и мнимых частей - CompareModReIm

Операция выполняется непосредственно блоком CompModulLong, на входы которого подаются части С-кода. В силу замечания к схеме CompLen результаты операций CompareLenReIm и совпадают CompareModReIm.

4.3.5 Сравнение по длине комплексных кодов с плавающей точкой - CompareModFloat

Эта операция эквивалентна *грубому* сравнению комплексных чисел по модулю. Она может быть использована в тех случаях, когда высокая точность сравнения не нужна, а важна скорость сравнения, например, при выборе ведущего элемента для решения системы линейных уравнений методом Гауса-Зейделя. Операция выполняется следующим образом: вначале сравниваются экспоненты, а при их равенстве сравниваются по модулю мантииссы.

4.3.6. Сравнение экспонент - CompareExp

Операция заключается в сравнении экспонент по модулю. Результат сравнения определяется по знаку разности экспонент.

Глава 7. Умножение

1. Метод умножения М-кодов
2. Матричные умножители
3. Сложение группы кодов
4. Формирование группы слагаемых при умножении
5. Составные умножители
6. Операции умножения М-кодов и С-кодов

1. Метод умножения М-кодов

Умножение любых позиционных кодов состоит из циклов «сдвиг-сложение». Поэтому для его реализации достаточно построить автомат, управляющий последовательным включение сумматора и сдвигателя. Рассмотрим вначале простейший метод умножения, для которого достаточно иметь один сумматор. На рис. 1 приведена традиционная схема последовательного умножения, которая применима и для умножения М-кодов: $C=A*B$. В начале умножения множитель A записывается в регистр $RegA$, множимое B - в регистр $RegB$, а в регистр $RegQ$ частичного произведения записывается «0». На каждом шаге

- анализируется m младших разрядов $RegB$; комплексное значение этих разрядов обозначим через h ;
- в сумматоре Sum вычисляется комплексный код числа $S = h * RegA + RegQ$;
- комплексный код S с выхода сумматора Sum через сдвигатель $ShiftRigth-1$ записывается в $RegC$ со сдвигом на m разрядов вправо;
- код из $RegC$ пересылается в $RegQ$;
- комплексный код из регистра $RegB$ через сдвигатель $ShiftRigth-2$ записывается в $RegW$ со сдвигом на m разрядов вправо;

- код из RegW пересылается в RegB.

Количество m анализируемых на каждом шаге разрядов определяет быстродействие умножения и сложность сумматора. Например, при $m=2$ число $h = \{1, 0, -1, -2\}$ и сумматор должен выполнять сложение, вычитание и вычитание удвоенного числа.

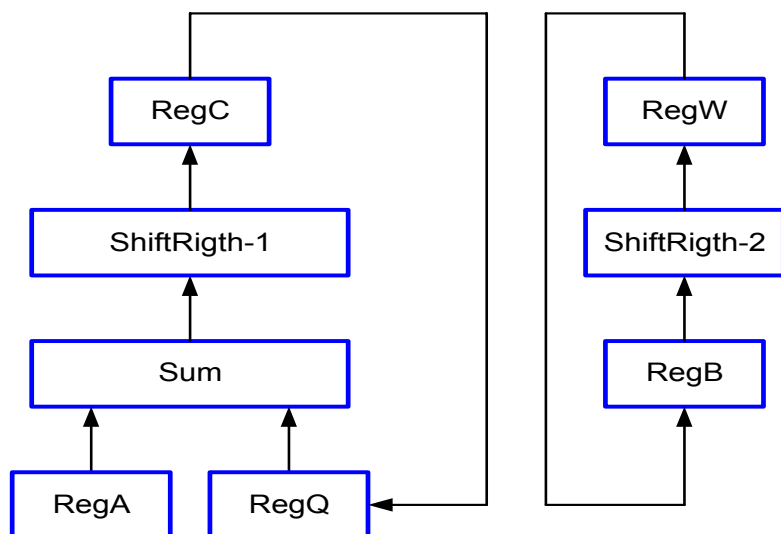


Рис. 1.

Для ускорения умножения применяют более сложные комбинационные схемы – комбинационные умножители, содержащие несколько сумматоров и выполняющие умножение за один такт (без промежуточных сигналов управления).

Ниже рассматриваются различные варианты комбинационных умножителей. Различаются а) матричные умножители и б) составные умножители. Этим термином мы будем называть устройства, в которых умножение можно рассматривать как формирование группы слагаемых с последующим сложением этой группы. Для того и другого существует несколько методов. При этом для построения составного умножителя можно воспользоваться любым сочетанием методов формирования группы слагаемых и сложения этой группы.

2. Матричные умножители.

В матричных умножителях сумматоры включаются последовательно (выход одного присоединен к первому входу следующего), а множимое подключено ко второму входу всех сумматоров. Каждый сумматор (назовем его линейным, поскольку он составляет одну линейку матрицы) находит сумму предыдущего результата C и числа $A * m$, где A – множимое, m – значение одного или двух разрядов кода множителя. Если используется один разряд, то говорят о матричном умножении на один разряд. Если используются два разряда, то говорят о матричном умножении на два разряда. Таким образом, даже при умножении на один разряд $m = \{0, 1\}$ каждый сумматор выполняет еще и функции мультиплексора.

При умножении на два разряда для Р-кодов $m = \{0, 1, 2, 3\}$. Следовательно, линейный сумматор для Р-кодов либо обнуляет число A , либо принимает его без изменений, либо принимает его со сдвигом на один разряд влево, либо утраивает его. Последнее действие усложняет матричный умножитель Р-кодов настолько, что он не находит практического применения.

При умножении на два разряда М-кодов $m = \{-2, -1, 0, 1\}$. Следовательно, линейный сумматор для М-кодов либо принимает число A со сдвигом на один разряд влево, либо инвертирует его, либо обнуляет его, либо принимает его без изменений. Далее устройства матричного умножения на два разряда рассматриваются подробно.

2.1. Инверсный матричный умножитель М-кодов - MultMatrShort.

Здесь используется матричная схема умножения на два разряда. Если множимое A содержит N разрядов, а множитель B содержит M разрядов, то матричная схема содержит $M/2$ сумматоров с разрядностью $(N+2)$. На каждый сумматор поступает частичное произведение, вычисленное предыдущим сумматором, и множимое. При этом на вход следующего сумматора поступает N старших разрядов с выхода предыдущего сумматора. Таким образом, частичное произведение сдвигается на 2 разряда влево. Каждый сумматор $k=0, 1, 2, \dots, (M/2)$ вычисляет величину

$$P_{k+1} = -(-4 \cdot P_k + A \cdot b_k) \quad (1)$$

или

$$P_{k+1} = 4 \cdot P_k - A \cdot b_k,$$

где P_k – частичное произведение, $P_0 = 0$,

$b_k = \{-2, -1, 0, 1\}$ – значение пары разрядов

инвертированного множимого, содержащего M разрядов.

Окончательное произведение равно

$$P = -A \cdot \sum_{k=0}^{(M/2-1)} b_k \cdot 4^k$$

или

$$P = -A \cdot B.$$

На рис. 1 представлена схема MultMatrShort. При этом приняты следующие обозначения:

AdderM2NShort – алгебраический сумматор M -кодов с удвоением, отличающийся от трехярусного алгебраического сумматора **nAddAlg3**, описанного в разделе 2.3.6 только тем, что в нем инвертор 2 выполнен как инвертор **nInv**, а не как инвертор **nInv2**. Таким образом этот сумматор вычисляет $C = (-q_1 \cdot A + B)$, где $q_1 = -2, -1, 0, 1$.

A – вход – N разрядов,

B – вход – M разрядов,

C – результат – $(N+M)$ разрядов.

2.2. Матричный квадратер M -кодов – QuadrMatr.

Этот квадратер отличается от умножителя MultMatrShort только тем, что $M=N$ и оба его входа объединены и поэтому он вычисляет $P = -A^2$.

2.3. Матричный умножитель M -кодов – MultMatr.

Этот умножитель отличается от умножителя MultMatrShort только тем, что на входе B дополнительно включен инвертор M -кодов **nInv**, и поэтому он вычисляет $P = A \cdot B$

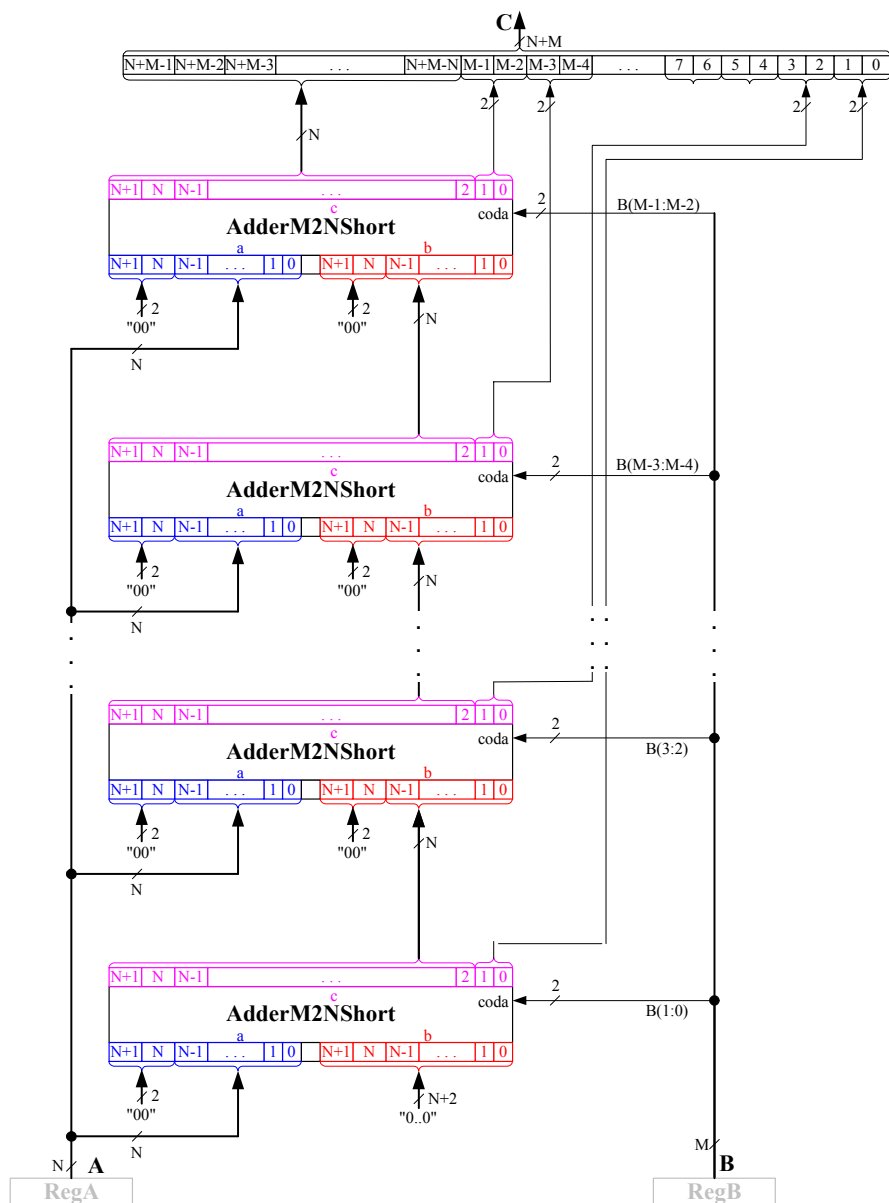


Рис. 1.

2.4. Матричный умножитель C-кодов - FloatMultMatr.

Этот умножитель состоит из 4-х матричных умножителей M-кодов **MultMatr** и двух алгебраических сумматоров **mAddAlg** - см. рис. 2.

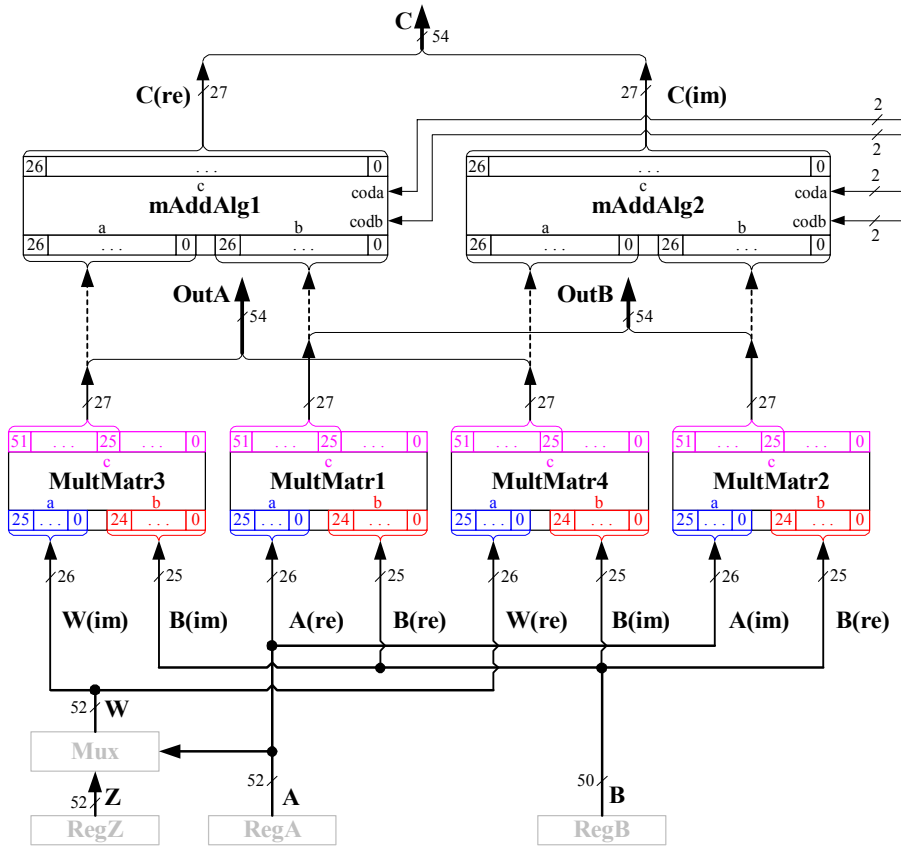


Рис. 2.

2.5. Матричный квадратор С-кодов - CalcModuLL

На рис. 3 представлена схема квадратора С-кодов, который вычисляет $C = |A|^2$, где A – входной L -разрядный С-код, а C – выходной L -разрядный Р-код.

Квадратор содержит разделитель Partitioning комплексного кода на реальную и мнимую части (см. раздел 3.1.6) и два квадратора QuadrMatr (см. раздел 5.2.2), которые вычисляют величины $-(\text{Re}(A))^2$ и $-(\text{Im}(A))^2$. Инверсный сумматор InvAdd вычисляет обратную сумму этих величин, т.е. квадрат модуля $|A|^2$. Декодер DecoderMP преобразует М-код этой величины в Р-код.

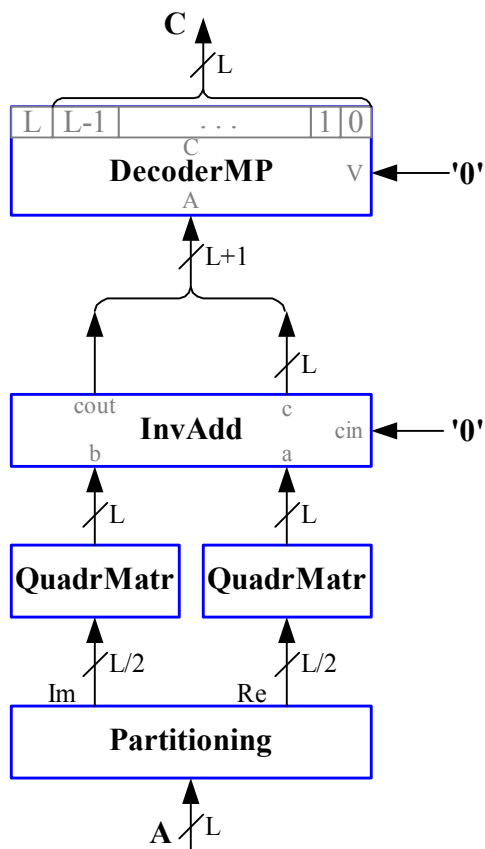


Рис. 3.

3. Сложение группы кодов

Рассмотрим группу Р-кодов и обозначим через N количество кодов в группе. Будем вычислять сумму этих кодов.

Первый метод заключается в том, что для сложения используется $(N-1)$ многоразрядных сумматоров. Обозначим через n номер кода и сумматора. При этом для кодов $n=(1,N)$, а для сумматоров $n=(1,(N-1))$. Первый сумматор складывает 1-ый и 2-ой коды, а каждый n -ый сумматор, где $n=(2,(N-1))$, складывает $(n+1)$ -ый код с результатом, полученным на n -ом сумматоре.

Второй метод заключается в том, что строится *бинарное* дерево сумматоров. Первый ярус, состоящий из $(N/2)$ сумматоров, складывает пары исходных кодов. Второй ярус, состоящий из $(N/4)$ сумматоров, складывает пары результатов, полученных на предыдущем ярусе, и т.д. Очевидно, это дерево состоит из $\log(N)$ ярусов и содержит $(N-1)$ сумматоров. На рис. 1 представлено дерево для обратного сложения 13-ти М-кодов, которое состоит из 4-х ярусов содержит 11 инверсных сумматоров.

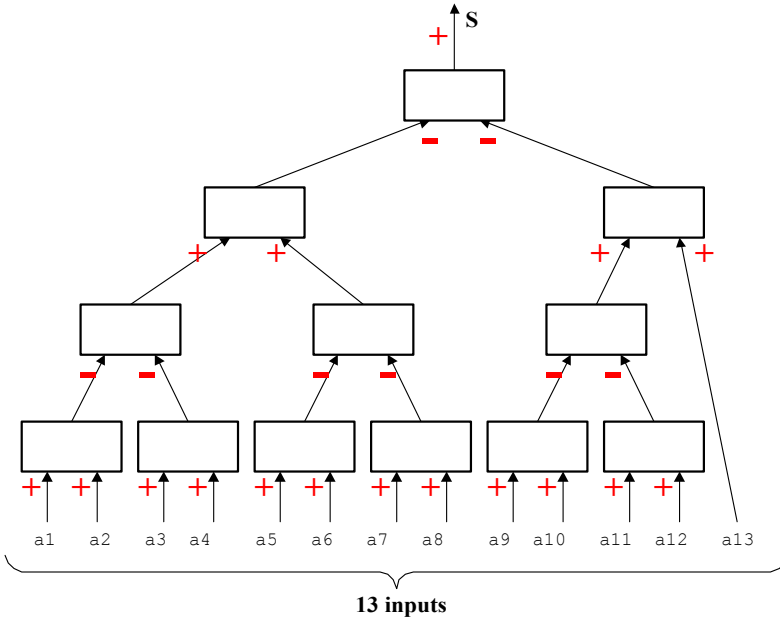


Рис. 1. Бинарное дерево для 13-и М-кодов.

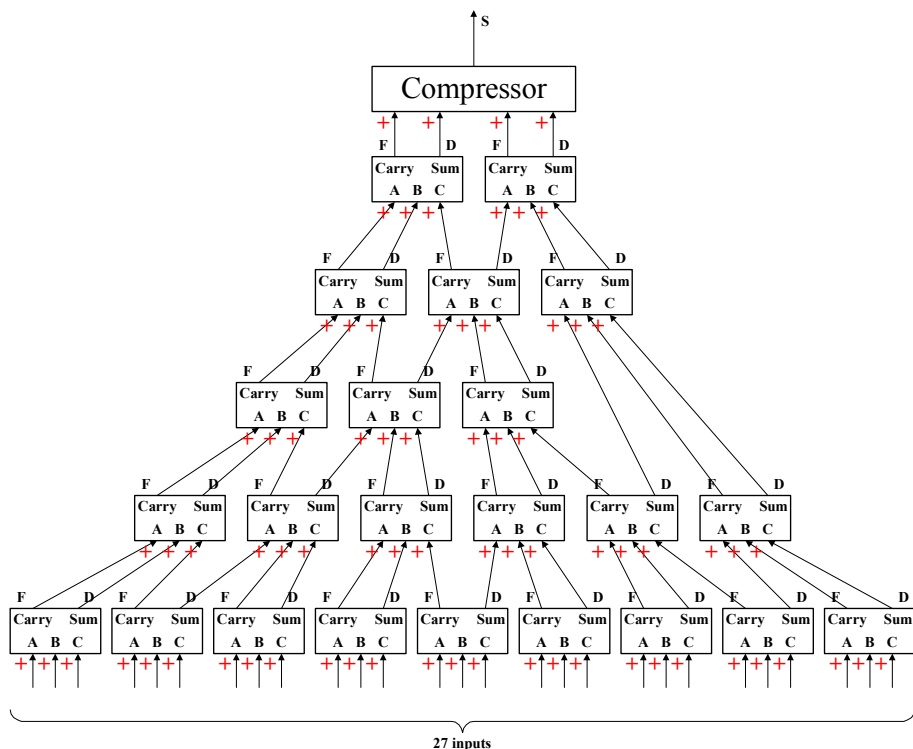


Рис. 2. Дерево Уоллиса для 27-ми Р-кодов.

Третий метод, называемый методом Уоллиса (Wallace) [7, 8], использует многоразрядные специальные сумматоры без переносов. Для удобства дальнейшего изложения назовем такое устройство полусумматором. Полусумматор состоит из одноразрядных полусумматоров, имеет три многоразрядных входа A, B, C и два многоразрядных выхода $D = \text{sum}$, $E = \text{carry}$. На одноименных разрядах первого и второго выходов D, E образуются код суммы трех одноименных разрядов на входах A, B, C . Важно отметить, что переносы при этом не распространяются. Таким образом, полусумматор вычисляет $(F + D) = (A + B + C)$, где $F = 2 * E$. Иначе говоря, полусумматор преобразует три кода в два кода с тем же числовым значением. С применением полусумматоров строится так называемое дерево Уоллиса для сложения группы Р-кодов – см. рис. 2. Первый ярус полусумматоров складывает тройки кодов. Этот ярус содержит $N/3$ полусумматоров и образует $2 * N/3$ кодов. Следующий ярус содержит $2 * N/9$ полусумматоров и образует $4 * N/9$ кодов и т.д.

Последний ярус содержит четырехвходовой сумматор, который принято называть компрессором. На выходе компрессора образуется искомая сумма S . На рис. 2 представлено дерево Уоллиса для сложения 27-ми кодов. Оно состоит из 6-ти ярусов и содержит 23 полусумматора и 1 компрессор. Для сравнения заметим, что для сложения 27-ми кодов матрица сумматоров первого метода состоит из 26-ти ярусов содержит 26 сумматоров.

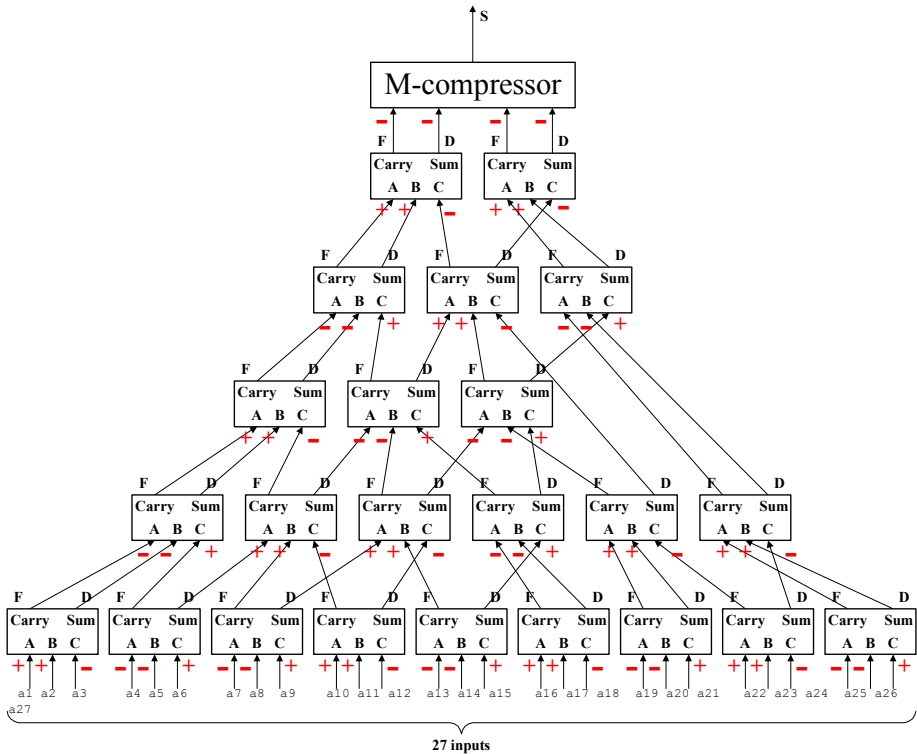


Рис. 3. Дерево Уоллиса для 27-ми М-кодов.

Распространим метод Уоллиса на М-коды. Для этого заменим полусумматоры Р-кодов на инверсные полусумматоры М-кодов, а 4-хвходовой сумматор Р-кодов заменим на 4-хвходовой инверсный сумматор М-кодов, который будем называть (по аналогии с предыдущим компрессором) компрессором М-кодов или М-компрессором. Инверсный полусумматор состоит из одноразрядных инверсных сумматоров, имеет три многоразрядных входа A , B , C и

два многоразрядных выхода $D=sum$, $E=carry$. На одноименных разрядах первого и второго выходов D , E образуется М-код алгебраической суммы $(-a-b+c)$ трех одноименных разрядов на входах A , B , C . Важно отметить, что переносы при этом не распространяются. Таким образом, инверсный полусумматор вычисляет $(F+D)=(-A-B+C)$, где $F=-2*E$. Иначе говоря, инверсный полусумматор (также, как и полусумматор Р-кодов) преобразует три кода в два кода с тем же числовым значением.

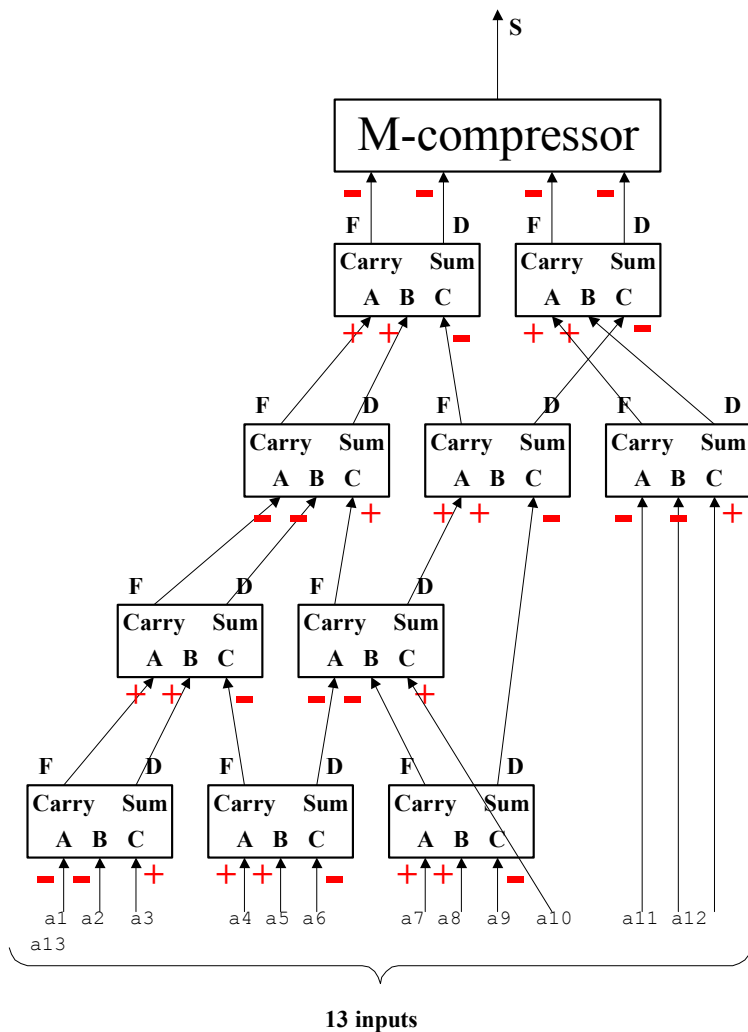


Рис. 4. Дерево Уоллиса для 13-ти М-кодов.

Дерево Уоллиса для сложения 27-ми М-кодов изображено на рис.3. На выходе М-компрессора в этом случае образуется алгебраическая сумма

$$S=(a1+a2+a6+a9+a10+a11+a15+a16+a17+a21+a22+a23+a27+ -a3-a4-a5-a7-a8-a12-a13-a14-a18-a19-a20-a24-a25-a26). \quad (1)$$

В эту сумму входит 13 положительных слагаемых и 14 отрицательных слагаемых. Дерево Уоллиса и в этом случае состоит из 6-ти ярусов и содержит 23 инверсных полусумматора и 1 М-компрессор.

Дерево Уоллиса для сложения 13-ти М-кодов изображено на рис. 4. На выходе М-компрессора в этом случае образуется алгебраическая сумма

$$S=(a3+a4+a5+a7+a8+a10+a13-a1-a2-a6-a9-a10-a11-a12). \quad (2)$$

В эту сумму входит 7 положительных слагаемых и 6 отрицательных слагаемых. Дерево Уоллиса и в этом случае состоит из 5-ти ярусов и содержит 10 инверсных полусумматора и 1 М-компрессор.

4. Формирование группы слагаемых при умножении

Можно указать три метода формирования группы слагаемых при умножении.

Первый метод состоит в том, что множимое подключается ко входу N мультиплексоров, каждый из которых управляется одним из разрядов N -разрядного множителя. Тем самым образуется N слагаемых (часть из которых равна нулю).

Второй метод состоит в следующем. Формируются коды A^*m , где m – значение двухразрядного кода. Для Р-кодов $m = \{0, 1, 2, 3\}$, для М-кодов $m = \{-2, -1, 0, 1\}$. Следовательно, для создания Р-кодов чисел A^*m требуется 1 сумматор, а для создания М-кодов чисел A^*m требуется только 1 инвертор (число $(-2A)$ образуется сдвигом на один разряд влево). Далее каждая четверка кодов чисел A^*m подключается ко входам мультиплексоров. Каждый мультиплексор управляется парой разрядов N -разрядного множителя и пропускает на выход число A^*m , где m – значение данной пары. Тем самым образуется $N/2$ слагаемых (часть из которых равна нулю). Аналогично может быть организовано умножение на 3 и более разрядов.

Третий метод далее будем называть попарным произведением. Он применим только для М-кодов и состоит в следующем. Слагаемые коды формируются из фрагментов, а каждый фрагмент представляет собой произведение пар разрядов сомножителей. Если разрядность сомножителей равна N , то количество фрагментов равно $(N*N/4)$. В табл. 1 показана величина и разрядность фрагмента $c=a*b$ для М-кодов. В этой таблице

$Sa=(-2a_1+a_0)$ - значение пары разрядов $a=(a_1, a_0)$,

$Sb=(-2b_1+b_0)$ - значение пары разрядов $b=(b_1, b_0)$,

$Sc=Sa*Sb$ – значение фрагмента.

Видно, что разрядность фрагмента не превышает 3. Фрагмент (a,b) , образованный произведением a -пары разрядов первого сомножителя на b -пару разрядов второго сомножителя, и фрагмент (b,a) будем называть сопряженными. Обратную сумму двух сопряженных фрагментов будем называть двойным фрагментом.

Очевидно, разрядность двойного фрагмента не превышает 4. Как будет показано далее, из этого следует, что общая разрядность двойных фрагментов меньше общей разрядности слагаемых, полученных по первому методу, а время формирования двойных фрагментов равно времени задержки в трехразрядной схеме. Заметим, что для Р-кодов разрядность суммы двойных фрагментов может быть равна 6. Отсюда и следует, что данный метод применим только для М-кодов.

Итак, рассмотрим *двойные фрагменты* (в дальнейшем - $\Delta\Phi$) и их положение в разрядной сетке – см. рис. 1, где в клетке на пересечении строки и столбца с номерами пар разрядов указан $\Delta\Phi$ и видно его положение относительно горизонтальной разрядной сетки. Эта таблица построена для перемножения 26-разрядных М-кодов. После перестановки некоторых фрагментов по вертикали образуется рис. 2, из которого следует, что $\Delta\Phi$ составляют в совокупности 13 регистров, разрядность которых равна 28. Схему, построенную по рис. 2, будем называть прекомпрессором двойных фрагментов или $\Delta\Phi$ -прекомпрессором. Он содержит 91 двойной фрагмент. Вообще, количество фрагментов в $\Delta\Phi$ -прекомпрессоре $f=N*N/8$.

Таблица 1.

№	a			b			c= a*b			
	a1	a0	Sa	b1	b0	Sb	Sc=Sa*Sb	c2	c1	c0
1	0	0	0	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0	0
3	1	0	-2	0	0	0	0	0	0	0
4	1	1	-1	0	0	0	0	0	0	0
5	0	0	0	0	1	1	0	0	0	0
6	0	1	1	0	1	1	1	0	0	1
7	1	0	-2	0	1	1	-2	0	1	0
8	1	1	-1	0	1	1	-1	0	1	1
9	0	0	0	1	0	-2	0	0	0	0
10	0	1	1	1	0	-2	-2	0	1	0
11	1	0	-2	1	0	-2	4	1	0	0
12	1	1	-1	1	0	-2	2	1	1	0
13	0	0	0	1	1	-1	0	0	0	0
14	0	1	1	1	1	-1	-1	0	1	1
15	1	0	-2	1	1	-1	2	1	1	0
16	1	1	-1	1	1	-1	1	0	0	1

Схема для формирования 4-разрядного кода фрагмента реализует табл. 1 и по объему сопоставима с одноразрядным сумматором. Схема для формирования 4-разрядного кода двойного фрагмента представлена на рис. 3, где

frag1 и frag2 - схемы для формирования кода фрагмента,

InvAdd - трехразрядный инверсный сумматор М-кодов.

Эта схема по объему сопоставима с 4-разрядным инверсным сумматором М-кодов. Таким образом, объем ДФ-прекомпрессора равен объему $f \cdot 4 = N \cdot N / 2$ одноразрядных схем. ДФ-прекомпрессор позволяет вдвое уменьшить количество складываемых М-кодов за время задержки в 4-разрядной схеме.

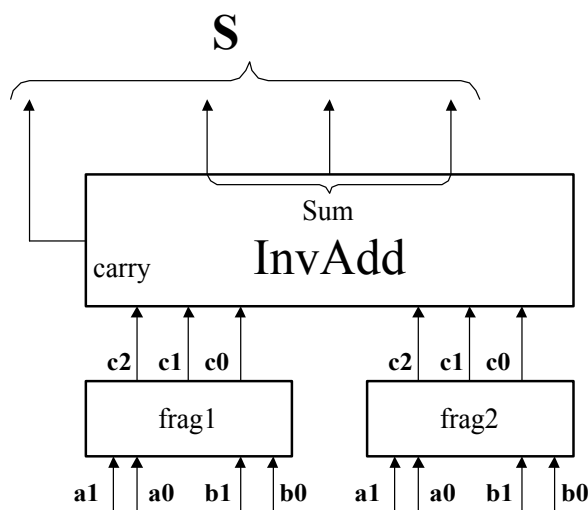


Рис. 3.

Схема двойного фрагмента для квадратора существенно упрощается, т.к. в этом случае значения сопряженных фрагментов совпадают и поэтому вместо инверсного сложения достаточно сделать сдвиг на один разряд влево. Эта схема представлена на рис. 4. Ее объем (как отмечалось) сопоставим с одноразрядным сумматором. Таким образом, объем ДФ-прекомпрессора для квадратора (в дальнейшем - *ДФК-прекомпрессор*) равен объему $f = N \cdot N / 8$ одноразрядных схем. Этот ДФК-прекомпрессор позволяет вдвое уменьшить количество складываемых М-кодов за время задержки в одноразрядной схеме. Итак, ДФК-прекомпрессор в 4

раза меньше и в 4 раза быстрее, чем ДФ-прекомпрессор. Мы воспользуемся этими преимуществами ДФК-прекомпрессора ниже для построения умножителей из квадраторов.

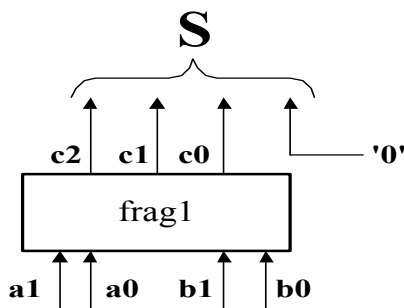


Рис. 4.

5. Составные умножители

Для построения составного умножителя можно воспользоваться любым сочетанием методов формирования группы слагаемых и сложения этой группы.

5.1. Умножитель М-кодов с деревом Уоллиса

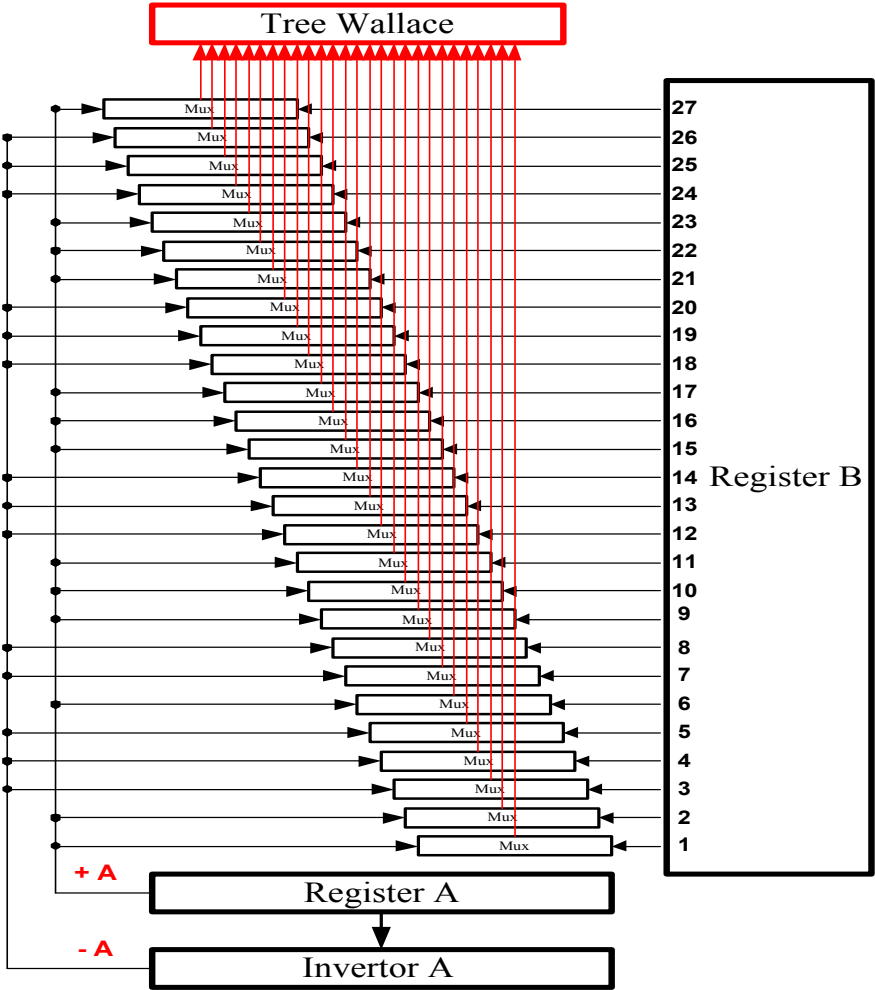


Рис. 1.

В этом умножителе используется первый метод формирования группы слагаемых и третий метод (метод Уоллиса) сложения группы кодов. На рис. 1 изображен умножитель 27-разрядных М-кодов с деревом Уоллиса, представленным на рис. 3.3. Мультиплексоры управляются разрядами множителя *B* и пропускают на вход дерева

числа A с выхода регистра $(+A)$ или числа $(-A)$ с выхода инвертора. Выбор между числами $(+A)$ или $(-A)$ выполняется в соответствии с формулой (1). Дальнейшее сложение 27-ми кодов на дереве Уоллиса описано выше.

5.2. Умножитель М-кодов с ДФ-прекомпрессором

В этом умножителе используется ДФ-прекомпрессор для формирования группы слагаемых и дерево (бинарное или Уоллиса) для сложения группы кодов – см. рис. 2.

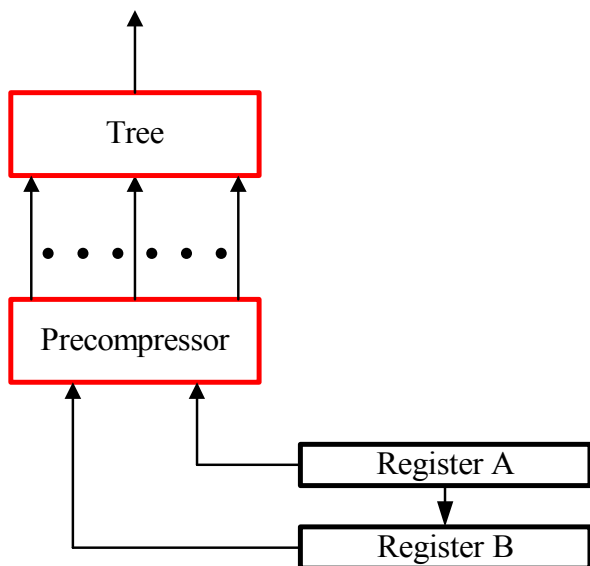


Рис. 2.

Важно отметить, что при использовании дерева Уоллиса между некоторыми выходами ДФ-прекомпрессора и входами дерева Уоллиса должны быть включены инверторы. Для примера на рис. 3 показаны эти соединения в умножителе 26-разрядных М-кодов, где используется дерево Уоллиса, изображенное на рис. 3.4, и ДФ-прекомпрессор, представленный на рис. 4.2.

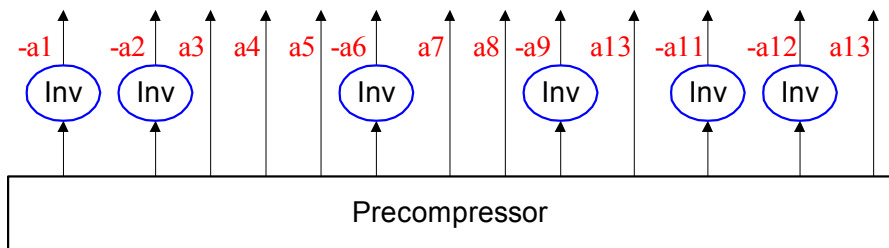


Рис. 3.

5.3. Умножитель М-кодов на квадраторах

Известно, что $a \cdot b = \left(\frac{a+b}{2}\right)^2 - \left(\frac{a-b}{2}\right)^2$. Кроме того, выше

показано, что ДФК-прекомпрессор в 4 раза меньше и в 4 раза быстрее, чем ДФ-прекомпрессор для умножения. Поэтому целесообразно строить умножитель М-кодов по данной формуле с использованием ДФК-прекомпрессоров.

5.4. Умножитель С-кодов на квадраторах

Рассмотрим следующие соотношения:

$$x = a + jb, \cdot y = c + jd, z = x \cdot y, \operatorname{Re} z = ac - bd, \operatorname{Im} z = cb + ad.$$

Рассмотрим, далее, числа

$$p_1 = (a+b) \cdot c = \left(\frac{a+b+c}{2}\right)^2 - \left(\frac{a+b-c}{2}\right)^2,$$

$$p_2 = (d+c) \cdot b = \left(\frac{d+c+b}{2}\right)^2 - \left(\frac{d+c-b}{2}\right)^2,$$

$$p_3 = (d-c) \cdot a = \left(\frac{d-c+a}{2}\right)^2 - \left(\frac{d-c-a}{2}\right)^2,$$

Очевидно,

$$\operatorname{Re} z = p_1 - p_2, \operatorname{Im} z = p_1 + p_3. \quad (1)$$

Для дальнейшего обозначим:

$$q_1 = \left(\frac{a+b+c}{2}\right)^2, \quad q_2 = \left(\frac{a+b-c}{2}\right)^2,$$

$$q_3 = \left(\frac{d+c+b}{2}\right)^2, \quad q_4 = \left(\frac{d+c-b}{2}\right)^2,$$

$$q_5 = \left(\frac{d-c+a}{2}\right)^2, \quad q_6 = \left(\frac{d-c-a}{2}\right)^2,$$

Выше показано, что ДФК-прекомпрессор в 4 раза меньше и в 4 раза быстрее, чем ДФ-прекомпрессор. Поэтому целесообразно строить умножитель С-кодов по формуле (1) с использованием ДФК-прекомпрессоров. Схема такого умножителя показана на рис. 4, где Inv – инверторы, q – квадраторы, остальные элементы – инверсные сумматоры. Основной объем этой схемы составляют 6 квадраторов вместо 4-х умножителей в ранее рассмотренном варианте.

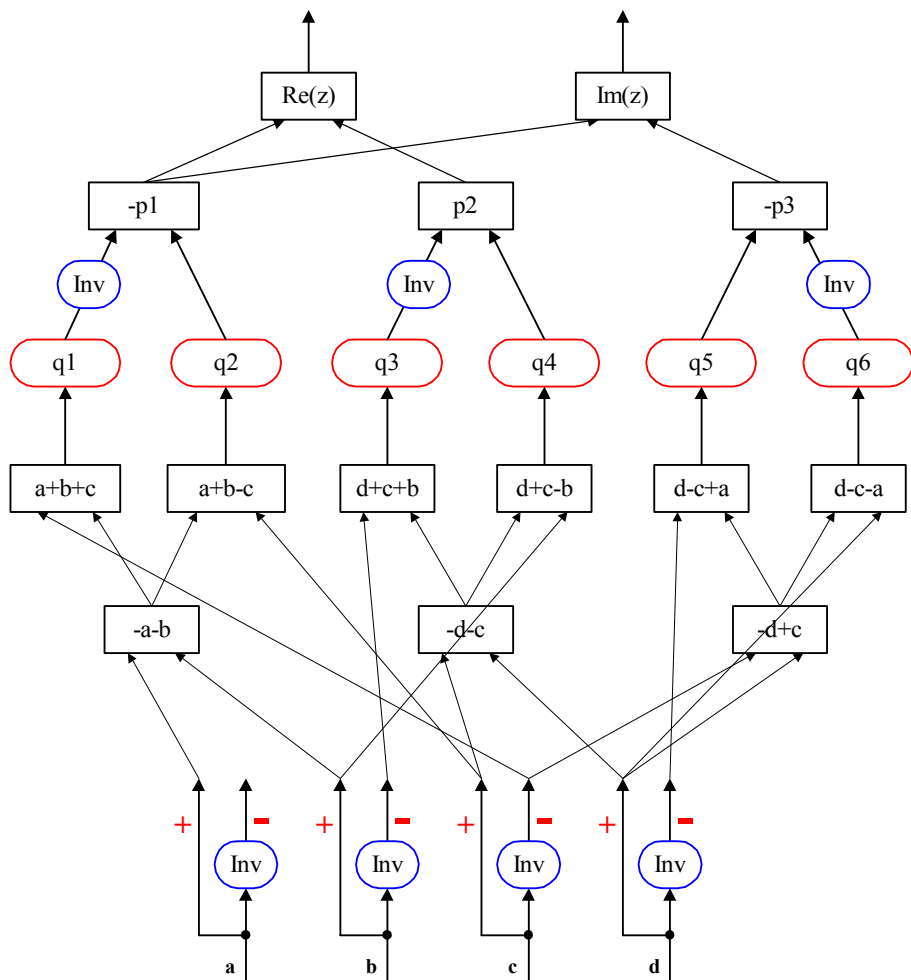


Рис. 4.

6. Операции умножения М-кодов и С-кодов

Рассмотрим операции умножения, которые могут быть выполнены с использованием матричного умножителя комплексных кодов FloatMultMatr, который состоит из 4-х матричных умножителей М-кодов MultMatr1,2,3,4 и двух алгебраических сумматоров М-кодов mAddAlg1,2. Реализация FloatMultMatr описана в разделе 2.4. На этом устройстве может быть выполнено несколько операций. Они перечислены в табл. 1.

Таблица 1.

№	Наименование	Обозначение
0	=== Умножение комплексных чисел	
1	Умножение комплексных чисел	MultMatrFloat
2	Квадрат комплексного числа	SquareMatrFloat
3	Квадрат модуля комплексного числа	SquareModuleMatrFloat
4	Центроаффинное преобразование комплексных чисел	CentroAffineMatrFloat
5	Аффинное преобразование комплексных чисел	AffineMatrFloat
6	Быстрое преобразование Фурье	ButterflyMatrFloat
7	Скалярное произведение комплексных чисел	ScalMatrFloat
8	Быстрое преобразование Фурье комплексных чисел (вариант 2)	InvButterflyMatrFloat
10	=== Умножение целых комплексных чисел	
11	Умножение целых комплексных чисел	MultMatrInt
12	Квадрат целого комплексного числа	SquareMatrInt
13	Квадрат модуля целого комплексного числа	SquareModuleMatrInt
14	Центроаффинное преобразование целых комплексных чисел	CentroAffineMatrInt
15	Аффинное преобразование целых целых комплексных чисел	AffineMatrInt

№	Наименование	Обозначение
16	Быстрое преобразование Фурье целых комплексных чисел	ButterflyMatrInt
17	Скалярное произведение целых комплексных чисел	ScalMatrInt
18	Быстрое преобразование Фурье целых комплексных чисел (вариант 2)	InvButterflyMatrInt
20	=== Умножение действительных чисел	
21	Умножение действительных чисел	MultRealMatrFloat
22	Квадрат действительного числа	SquareRealMatrFloat
23	Детерминант 2*2-матрицы действительных чисел	DetMatrFloat
24	Умножение 2*2-матрицы на 2-вектор для действительных чисел	MatVecMatrFloat
30	=== Умножение целых действительных чисел	
31	Умножение целых действительных чисел	MultRealMatrInt
32	Квадрат целого действительного числа	SquareRealMatrInt
33	Детерминант 2*2-матрицы целых действительных чисел	DetMatrInt
34	Умножение 2*2-матрицы на 2-вектор для целых действительных чисел	MatVecMatrInt
35	Умножение четырех пар целых действительных чисел	MultFourMatrInt

Далее эти операции описываются подробнее. В табл. 1а показано для определенной операции

P - номер этой операции в табл. 1,

U - номер множителя M-кодов MultMatr1,2,3,4,

U1 - множимое на входе U-умножителя,

U2 – множитель на входе U-умножителя,

SumRe – произведения, которые образуются на выходах умножителей IntMultMatr и поступают на входы сумматора реальных частей mAddAlg1,

SumIm – произведения, которые образуются на выходах умножителей IntMultMatr и поступают на входы сумматора мнимых частей mAddAlg2.

Таблица 1а.

P	W	U	U1	U2	SumRe	SumIm
1	A	1	ReA	ReB	M1=ReA*ReB	
		2	ImA	ReB		M2=ImA*ReB
		3	ReA	ImB		M3=ReA*ImB
		4	ImA	ImB	M4=ImA*ImB	
2	A	1	ReA	ReA	M1=ReA*ReA	
		2	ImA	ReA		M2=ImA*ReA
		3	ReA	ImA		M3=ReA*ImB
		4	ImA	ImA	M4=ImA*ImA	
3	A	1	ReA	ReA	M1=ReA*ReA	
		2	ImA	ReA		0
		3	ReA	ImA		0
		4	ImA	ImA	M4=ImA*ImA	
4	Z	1	ReA	ReB	M1=ReA*ReB	
		2	ImA	ReB		M2=ImA*ReB
		3	ReW	ImB		M3=ReW*ImB
		4	ImW	ImB	M4=ImW*ImB	
7	A	1	ReA	ReB	M1=ReA*ReB	
		2	ImA	ReB		0
		3	ReA	ImB		0
		4	ImA	ImB	M4=ImA*ImB	
14	Z	1	ReA	ReB	M1=ReA*ReB	
		2	ImA	ReB		M2=ImA*ReB
		3	ReW	ImB		M3=ReW*ImB
		4	ImW	ImB	M4=ImW*ImB	
21	A	1	ReA	ReB	M1=ReA*ReB	
		2	0	ReB		0
		3	ReA	0		0
		4	0	0	0	
22	A	1	ReA	ReA	M1=ReA*ReA	
		2	0	ReA		0
		3	ReA	0		0
		4	0	0	0	
23	A	1	ReA	ReB	0	

		2	ImA	ReB		$M2=ImA*ReB$
		3	ReA	ImB		$M3=ReA*ImB$
		4	ImA	ImB	0	
24	Z	1	ReA	ReB	$M1=ReA*ReB$	
		2	ImA	ReB		$M2=ImA*ReB$
		3	ReW	ImB		$M3=ReW*ImB$
		4	ImW	ImB	$M4=ImW*ImB$	
35	A	1	ReA	ReB	$M1=ReA*ReB$	
		2	ImA	ReB		$M2=ImA*ReB$
		3	ReA	ImB		$M3=ReA*ImB$
		4	ImA	ImB	$M4=ImA*ImB$	

6.1. Умножение комплексных чисел

На входах умножителя присутствуют коды чисел A, B. Умножитель в этом случае выполняет операцию $C=A*B$, причем $ReC=M1-M4$, $ImC=M2+M3$. Произведение оказывается сдвинутым на 2 разряда. Поэтому экспонента результата вычисляется по формуле $ExpC=ExpA+ExpB+1$.

6.2. Квадрат комплексного числа

Эта операция отличается от операции «Умножение комплексных чисел» только тем, что вначале операции устанавливается $A=B$ и вычисляется произведение $C=A*A$.

6.3. Квадрат модуля комплексного числа

Эта операция отличается от возведения в квадрат тем, что здесь множители $M2$ и $M3$ не участвуют в операции, а сумматор вычисляет $C = ReC = M1 + M4 = (Re A)^2 + (Im A)^2$.

6.4. Центроаффинное преобразование

Итак, на входах умножителя присутствуют коды чисел A, B, W. Умножитель в этом случае выполняет центроаффинное преобразование $C=ReB*A+ImB*W$, причем $ReC=M1-M4$, $ImC=M2+M3$. В этом случае $W=Z$. Следовательно, $C=ReB*A+ImB*Z$.

6.7. Скалярное умножение комплексных чисел.

Это вычисление выполняется по формуле: $S=ac+bd$. Операндами команды являются комплексные числа $a+jb$ и $d+jc$, а результатом является действительное число. Здесь $a=\text{Re}A$, $b=\text{Im}A$, $c=\text{Re}B$, $d=\text{Im}B$.

Заметим, что операции «Детерминант» и «Скалярное произведение» аналогичны операции умножения. Она использует два операнда – комплексные числа $a+jb$, $d+jc$ и образует результат $x+ju$, где $x=ad-bc$, $y=ac+bd$. Действительная часть совпадает с результатом операции «Детерминант», а мнимая часть совпадает с результатом операции «Скалярное произведение». Отдельно выполняемые операции и позволяют в некоторых случаях повысить точность результата (а именно, тогда, когда действительная и мнимая части существенно отличаются по величине и меньшая часть может исчезнуть в результате нормализации).

6.11. Умножение целых комплексных чисел

Эта операция аналогична умножению комплексных чисел. Отсутствуют операции с экспонентами. В этом случае вычисляется произведение $C=A*B$

6.12. Квадрат целого комплексного числа

Эта операция отличается от операции «Умножение целых комплексных чисел» только тем, что вначале операции устанавливается $A=B$. В этом случае вычисляется произведение $C=A*A$.

6.13. Квадрат модуля целого комплексного числа

Эта операция отличается от возведения в квадрат целого комплексного числа тем, что здесь множители $M2$ и $M3$ не участвуют в операции, а сумматор вычисляет $C = \text{Re } C = M1 + M4 = (\text{Re } A)^2 + (\text{Im } A)^2$.

6.14. Центроаффинное преобразование целого комплексного числа

Центроаффинное преобразование заключается в том, что действительная и мнимая части некоторого комплексного числа умножаются на различные комплексные числа. Эта операция отличается от операции «Умножение целых комплексных чисел» тем, что

- вначале операции вводится еще один аргумент Z ,
- множитель A умножается на реальную часть множителя B ,
- множитель Z умножается на мнимую часть множителя B ,
- после окончания умножения в регистрах D , E образуются следующие числа:

$$D = \text{Re}(D) + \text{Im}(D) = \text{Re}(A) * \text{Re}(B) + \text{Im}(A) * \text{Re}(B) = A * \text{Re}(B)$$

$$E = \text{Re}(E) + \text{Im}(E) = \text{Re}(Z) * \text{Im}(B) + \text{Im}(Z) * \text{Im}(B) = Z * \text{Im}(B)$$

Окончательно произведение вычисляется на последнем такте суммирования по формуле

$$C = \{\text{Re}(C)\} + \{\text{Im}(C)\} = \{\text{Re}(D) - \text{Im}(E)\} + \{\text{Im}(D) + \text{Re}(E)\}$$

Итак, на входах матричного умножителя присутствуют коды чисел A , B , W . Умножитель в этом случае выполняет центроаффинное преобразование $C = \text{Re}B * A + \text{Im}B * W$, причем $\text{Re}C = M1 - M4$, $\text{Im}C = M2 + M3$. В данной операции $W = Z$. Таким образом, $C = \text{Re}B * A + \text{Im}B * Z$.

6.15. Аффинное преобразование целого комплексного числа

Эта операция отличается от операции «Центроаффинное преобразование целого комплексного числа» тем, что

- вначале операции вводится еще один аргумент F ,
- после окончания центроаффинного преобразования выполняется еще одно сложение результата центроаффинного преобразования M с комплексным числом F .

6.16. Быстрое преобразование Фурье целых комплексных чисел - «Бабочка»

Это – базовая операция для быстрого преобразования Фурье. В этой операции участвуют целые комплексные коды A , B , Z . Разрядность кодов A , Z равна 12. Здесь вычисляются следующие величины:

$$Q = A * B, C = Q - Z, D = Q + Z.$$

Таким образом, операция выполняется по следующей схеме

- умножение целых
- вычитание
- сложение

Длительность операции бабочка на два такта превышает длительность умножения.

6.21. Умножение действительных чисел

В этой операции множимое находится в регистре А, а множитель - в регистре В. Произведение оказывается сдвинутым на 2 разряда. Поэтому экспонента результата вычисляется по формуле $E_{\text{пр}}C = E_{\text{пр}}A + E_{\text{пр}}B + 1$.

6.22. Квадрат действительного числа

Эта операция отличается от операции «Умножение действительных чисел» только тем, что вначале операции устанавливается $A=B$.

6.23. Ддетерминант квадратной матрицы 2*2 с действительных чисел.

Это вычисление выполняется по формуле

$$\text{Det}2 = \det \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc.$$

Операндами этой команды являются комплексные числа $a+jb$ и $d+jc$, а результатом является действительное число. Здесь $a=\text{Re}A$, $b=\text{Im}A$, $c=\text{Re}B$, $d=\text{Im}B$.

6.24. Умножение действительной квадратной матрицы 2*2 на действительный вектор-2.

Умножение действительной матрицы 2*2 вида $A = \begin{vmatrix} a & c \\ b & d \end{vmatrix}$ на действительный вектор-2 вида $M = \begin{vmatrix} s \\ t \end{vmatrix}$ дает в результате действительный вектор-2 вида $Z = \begin{vmatrix} x \\ y \end{vmatrix}$. Здесь $x=as+ct$, $y=bs+dt$.

Операндами этой команды являются комплексные числа $a+jb$, $d+jc$, $s+jt$, а результатом является комплексное число $x+ju$.

Заметим, что эта операция аналогична операции центроафинного преобразования. Она использует три операнда – комплексные числа $a+jb$, $c+jd$, $s+jt$ (где $a=\text{Re}A$, $b=\text{Im}A$, $c=\text{Re}B$, $d=\text{Im}B$, $s=\text{Re}W$, $t=\text{Im}W$) и образует результат $x+ju$, где $x=as-ct$, $y=bs+dt$. Отличие заключается только в знаке перед членом ct .

6.31. Умножение целых действительных чисел

В этой операции вычисляется произведение целых действительных чисел, представленных М-кодами, за один такт. При этом множимое в регистре А содержит 24 разряда (12 разрядов в действительной части), а множитель в регистре В содержит 22 разряда (11 разрядов в действительной части).

6.32. Квадрат целого действительного числа

Эта операция отличается от предыдущей только тем, что множимое и множитель равны и содержат 22 разряда (11 разрядов в действительной части).

6.35. Умножение 4-х пар целых действительных чисел

Эта операция отличается от центроаффинного преобразования тем, что $W=A$ и результат из умножителей выдается непосредственно на выходы С и D. Сумматор не используется. Итак,

$$\text{Re}C = M1 = \text{Re}A * \text{Re}B$$

$$\text{Im}C = M2 = \text{Im}A * \text{Re}B$$

$$\text{Re}D = M3 = \text{Im}A * \text{Im}B$$

$$\text{Im}D = M4 = \text{Re}A * \text{Im}B$$

Здесь исходные числа $\text{Re}(A)$, $\text{Im}(A)$, $\text{Re}(B)$, $\text{Im}(B)$ и результаты $\text{Re}(D)$, $\text{Im}(D)$, $\text{Re}(C)$, $\text{Im}(C)$ являются М-кодами действительных чисел.

Глава 8. Деление

1. Метод деления
2. Декомпозиции и композиции
3. Алгоритм обращения и деления

1. Метод деления

Деление позиционных кодов действительных чисел по отрицательному основанию во многом аналогично обычному делению и состоит из циклов ‘сдвиг-вычитание-сравнение’. Отличие состоит только в правилах выполнения сравнения. Для действительных положительных чисел сравнение производится по знакам предыдущего и последующего остатков. Для кодов действительных чисел по отрицательному основанию знак остатка может быть определен по четности/нечетности номера старшего значащего разряда. Соответственно сравнение знаков остатков можно выполнить сравнением четности старших значащих разрядов. Сравнение по знаку можно было бы заменить сравнением по модулю. Для действительных чисел это одно и то же, ибо сохранение знака остатка свидетельствует об уменьшении его модуля. Ниже рассматривается алгоритм деления, основанный на сравнении длин двоичных кодов.

Итак, необходимо найти частное $Z = V/W$ от деления чисел - делимого и делителя, представленных своими M -кодами $\langle V \rangle$ и $\langle W \rangle$ соответственно. Рассмотрим алгоритм деления, обозначив

P - предыдущий остаток,

S - следующий остаток,

v, w, p, s - длина кодов V, W, P, S соответственно.

Алгоритм 1.

1. Выполняется сдвиг делителя W на b разрядов влево при $b > 0$ или вправо при $b < 0$, где $b = v - w$.
2. Предыдущему остатку P присваивается значение делимого V , а частному Z - значение 0.
3. Сдвинутый делитель W вычитается из предыдущего остатка P . Разность является следующим остатком S .

4. Сравниваются размеры p и s с целью определения нового значения частного, сдвинутого делителя и величины остатка для следующего цикла деления. Эти данные определяются в соответствии с табл. 1.
5. Выполняется переход к пункту 3 с новыми значениями сдвинутого делителя и предыдущего остатка.

Таблица 1

Условие	Результат сравнения		
	Частное	h	Предыдущий остаток для следующего цикла
$s < p$	увеличивается на единицу h -го разряда	увеличивается на единицу	$P=S$
$s = p$	увеличивается на единицу h -го разряда	остается без изменений	$P=S$
$s > p$	остается без изменений	увеличивается на единицу	$P=P$

Деление производится до получения заданного числа разрядов в частном, которое образуется суммированием единиц h -го разряда, полученных в пункте 4. Важно отметить, что случай ' $s=p$ ' может повторяться, то-есть делитель может вычитаться более одного раза, не сдвигаясь. При определенном выборе соотношения между модулями делителя и делимого количество вычитаний не превышает 2. В связи с этим деление может выполняться в два этапа:

- декомпозиция делимого, т.е. накопление единиц в разрядах частного;
- композиция, т.е. сложение единиц в разрядах частного.

2. Декомпозиции и композиции

Декомпозиция для деления состоит в разложении данного комплексного числа Z по формуле $\frac{1}{16Z} = \sum a_h Z(-2)^h$.

Композиция состоит в вычислении по формуле $\frac{1}{16Z} = \sum a_h (-2)^h$. При аппаратной реализации этих операций используются сумматор, сдвигатель, компаратор (блок сравнения длин CompLen , описанный выше) и два регистра для накопления разрядов $a_h = \{0,1,2\}$ частного, которые мы будем обозначать как Acc1 и Acc2 . При описании алгоритма декомпозиции приняты следующие обозначения:

Z – данное число,

E_{prev} – предыдущее значение результата,

E_{next} – следующее значение результата,

Compar – функция сравнения комплексных чисел по модулю (или по длине, что эквивалентно), которая выполняется компаратором и возвращает значение «1», если $E_{\text{next}} \leq E_{\text{prev}}$ и – «0», если $E_{\text{next}} > E_{\text{prev}}$,

$R = a_h = \{0,1,2\}$ – текущее значение разряда частного,

Accum – функция записи в данный разряд регистров частного; эта функция устанавливает текущие разряды регистров Acc1 и Acc2 по следующему правилу:

R	Acc1	Acc2
0	0	0
1	1	0
2	0	1

DZ – приращение,

H – текущий номер разряда, $H_{\min} \leq H \leq H_{\max}$,

Iter – счетчик итераций.

Схема алгоритма декомпозиции при делении S -кода действительного числа DecompDevisReal представлена на рис. 1.

Композиция CompDevis эквивалентна сложению кодов, полученных в двух регистрах Acc1 и Acc2 , как С-кодов, по формуле: $W = \langle \text{Acc1} \rangle - 2 \cdot \langle \text{Acc2} \rangle$.

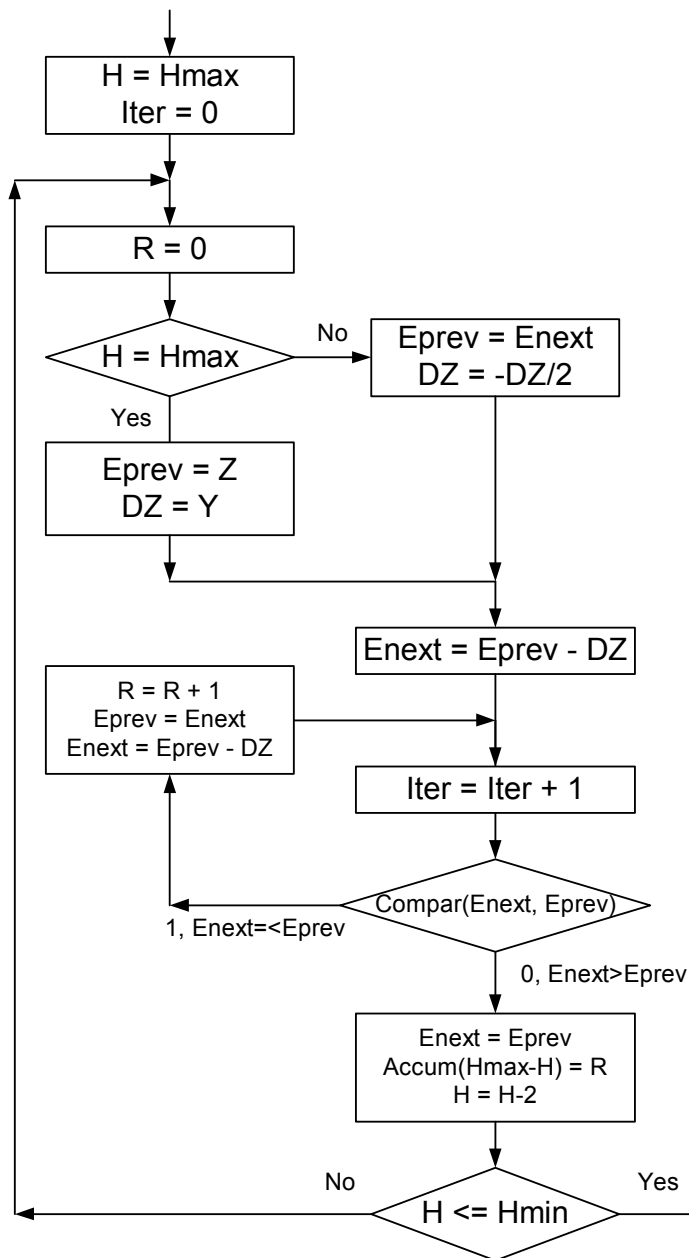


Рис. 1.

3. Алгоритм обращения и деления

Вначале рассмотрим обращение числа Z , которое состоит в следующем:

1. Декомпозиция `DecompDivisReal`.
2. Композиция `CompDivis`.

При этом определяется число $\frac{1}{16Z}$.

Теперь рассмотрим деление чисел $V = \frac{Y}{Z}$. Пусть делитель

$Z = (-2)^k \cdot M$, где M - мантисса, k - экспонента. Деление целесообразно выполнять по следующему алгоритму:

1. Определение числа $W = \frac{1}{16 \cdot M}$, т.е. обращение числа M .
2. Умножение $V = (-2)^{-k+4} Y \cdot W$.

Часть 2. Коды комплексных чисел и векторов

Аннотация

Рассматривается малоизвестная теория позиционного кодирования комплексных чисел и векторов. Описываются аппаратно-реализуемые алгоритмы кодирования, декодирования, арифметических операций, вычисления элементарных функций и др. Приводятся схемы основных вычислительных блоков. Описываются программы моделирования в системе MATLAB.

Содержание

Предисловие \ 2-4

Глава 1. Позиционные коды комплексных чисел и векторов \ 2-8

Глава 2. Точность кодирования \ 2-36

Глава 3. Поразрядные арифметические операции \ 2-59

Глава 4. Алгоритмы кодирования и декодирования комплексных чисел \ 2-81

Глава 5. Умножение \ 2-85

Глава 6. Метод «цифра за цифрой» \ 2-103

Глава 7. Деление \ 2-134

Глава 8. Вычисление функций комплексного переменного \ 2-155

Глава 9. Решение уравнений \ 2-181

Глава 10. Моделирование устройства извлечения квадратного корня из комплексных чисел \ 2-193

Глава 11. Моделирование устройства для потенцирования и логарифмирования комплексных чисел \ 2-220

Глава 12. Моделирование устройства для вычисления одного корня степенного полинома \ 2-242

Предисловие

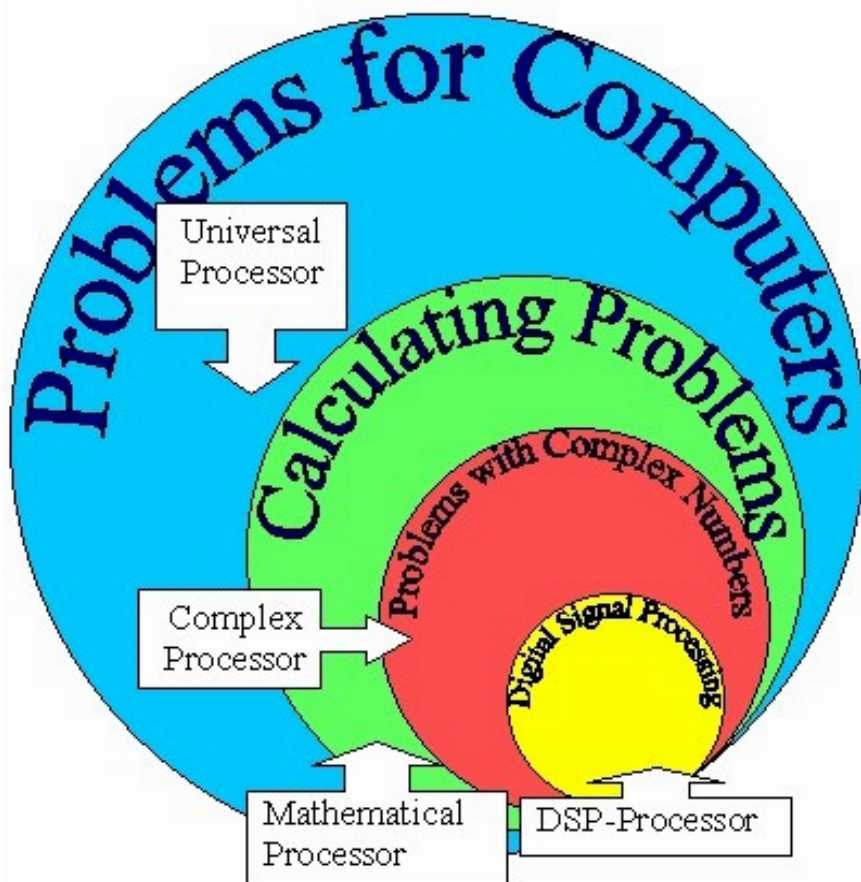
В этой части книги рассматривается теория и применение позиционного кодирования комплексных чисел и векторов. Предпочтение, отдаваемое именно позиционным кодам, объясняется тем, что с ними очень просто выполняются арифметические операции. Так, вне зависимости от объекта кодирования сложение позиционных кодов связано с распространением переносов от младших разрядов к старшим, а умножение и деление состоит из сдвигов и алгебраических сложений. Метод «цифра-за-цифрой» вообще применим только в сочетании с позиционной системой кодирования.

Далее показано, что с позиционными кодами комплексных чисел и векторов выполнимы операции алгебраического сложения, векторного, скалярного и специального умножения, деления комплексных чисел и векторов. Это может быть использовано при построении процессоров, оперирующих с комплексными числами и векторами в целом. Такой процессор требует более простого алгоритма для решения задач, а при данном алгоритме работает по более короткой программе и обладает повышенным быстродействием. Для оценки этих величин можно указать, например, что программа векторного умножения векторов, заданных тремя числами, содержит 6 операций умножения и 3 операции вычитания.

Укажем некоторые области применения комплексных чисел: телекоммуникация, цифровая фильтрация, обработка изображений, обработка сигналов в радарх/сонарах, распознавание сигналов, спектральный анализ, инженерная графика в системах автоматизированного проектирования, навигационные системы, компьютерные игры, медицинская диагностика, проектирование электромеханических систем, управление энергосистемами, многоканальные цифровые системы автоматического регулирования. Вообще, вычислительная математика в большей своей части является математикой комплексных чисел. Поэтому универсальный компьютер для вычислений — это компьютер, оперирующий с комплексными числами. Этот факт не учитывается практикой по нескольким причинам:

- Программы позволяют завуалировать базовые возможности компьютера (на самом деле даже арифметические операции можно было бы заменить на более простые – счетчик и сравнение с нулем),
- Существуют специализированные компьютеры для решения вычислительных задач, требующих значительного быстродействия,
- Не было предложено комплексной компьютерной арифметики.

Арифметическое устройство для операций с комплексными числами САУ превращает компьютер, оснащенный им, в универсальный математический компьютер. Место математического компьютера в общем круге компьютерных задач иллюстрируется следующей диаграммой.



Вторая часть книги содержит **12** глав. В **главе 1** рассматривается теория позиционного кодирования комплексных чисел и векторов. Описывается метод поиска оснований кодирования и предлагаются различные основания кодирования. Выделяется несколько вариантов построения двоичных кодов комплексных чисел и векторов. В **главе 2** исследуются вопросы точности представления комплексных чисел кодами, построенными выше. В **главе 3** рассматриваются операции, которые можно выполнять на схемах с распространением переноса. Формулируются два алгоритма таких операций и рассматриваются многочисленные частные случаи операций с различными кодами. В **главе 4** предлагаются алгоритмы для преобразования чисел из одной системы кодирования в другую. В **главе 5** рассматриваются методы, алгоритмы и схемы умножения чисел и векторов, в т. ч. скалярного, векторного и специального умножения векторов. Для последнего случая строится специальная алгебра. В **главе 6** известный для кодов действительных чисел метод ‘цифра за цифрой’ обобщается на позиционные коды комплексных чисел. Приводится описание специализированных схем. В **главе 7** рассматриваются методы, алгоритмы и схемы деления комплексных чисел, а также деления векторов в рассмотренной при умножении специальной алгебре. В **главе 8** строятся алгоритмы и схемы для аппаратного вычисления элементарных функций комплексного переменного. В **главе 9** рассматриваются алгоритмы решения алгебраических, трансцендентных, дифференциальных уравнений с использованием специализированных операций комплексной арифметики.

Главы 10-11 содержат описание моделей некоторых устройств в системе MATLAB. В **главе 10** рассматривается моделирование устройства извлечения квадратного корня из комплексных чисел. Показано, что по количеству тактов предлагаемый алгоритм сравним с алгоритмами вычисления элементарных функций действительного аргумента. В **главе 11** описывается устройство для потенцирования и логарифмирования комплексных чисел. Приведенной в этих главах информации (вместе с некоторыми специальными схемами, описанными в главе 6) достаточно для практической реализации указанных устройств. В **главе 12** описывается модель устройства для вычисления одного корня степенного полинома. Эта глава служит иллюстрацией способа, которым можно воспользоваться для моделирования устройств,

предназначенных для аппаратной реализации сложных вычислений методом «цифрв-за-цифрой».

Как показано в части 1, некоторая композиция кодов по отрицательному основанию может рассматриваться как единый код комплексного числа. Поэтому часть 1 может рассматриваться как описание аппаратуры для арифметического устройства, оперирующего с комплексными числами. Определенные дополнения - специализированные схемы для реализации метода "цифра-за-цифрой" описываются в **главе 6**.

Глава 1. Позиционные коды комплексных чисел и векторов

1.1. О методе позиционного кодирования

В этом разделе будут рассмотрены позиционные коды многомерных векторов Z , основанные на их представлении в виде разложения

$$Z = \sum_m r_m f(\rho, m), \quad (1)$$

где

m - номер разряда,

ρ - основание кодирования, число или вектор,

$f(\rho, m)$ - базовая функция номера и основания,

r - разряд разложения, число или вектор, принимающий значения из ограниченного множества

$$A_R = \{a_0, a_1, a_2, \dots, a_j, \dots, a_{R-1}\}, \quad \text{содержащего } R$$

различных величин a_j .

Позиционный код вектора Z , соответствующий этому разложению, имеет вид

$$K(Z) = \dots \sigma_m \dots,$$

где σ_m - цифра, обозначающая величину r_m . Формула (1) включает операции сложения и умножения. Для существования алгоритмов операций с такими разложениями (или, что одно и то же, с позиционными кодами) сложение и умножение должны быть ассоциативными и коммутативными, а также подчиняться дистрибутивному закону. Следовательно, для возможности позиционного кодирования некоторого множества объектов это множество должно составлять кольцо. Такому требованию удовлетворяет множество действительных чисел и множество

многомерных векторов, в котором определены операции сложения и умножения на число. Для действительных чисел позиционные системы известны. Для указанного множества векторов ниже будет построена позиционная система счисления с действительным основанием.

Множество комплексных чисел составляет кольцо и для него также будут построены позиционные системы счисления по действительному и комплексному основаниям.

Для построения позиционной системы счисления многомерных векторов по векторному основанию должна быть определена операция умножения векторов, подчиняющаяся вышеперечисленным законам. Другими словами, должна быть определена алгебра в многомерном векторном пространстве. Это сделано ниже.

Вначале рассмотрим два способа кодирования, а затем перейдем к более общему и строгому описанию метода позиционного кодирования.

1.2. Два способа синтеза кодов КОМПЛЕКСНЫХ ЧИСЕЛ

Позиционные коды комплексных чисел могут быть получены некоторой композицией кодов действительных чисел по отрицательному основанию. Здесь и далее j - мнимая единица.

Пусть X_α и X_β - действительные числа, заданные двоичными разложениями по основанию $\rho = -2$, то есть

$$X_\alpha = \sum_{(m)} \alpha_m \rho^m, \quad X_\beta = \sum_{(m)} \beta_m \rho^m.$$

Этим разложениям соответствуют коды

$$K(X_\alpha) = \alpha_m, \quad K(X_\beta) = \beta_m.$$

Существуют два способа объединения этих двух кодов в единый код комплексного числа. Первый из них заключается в том, что пара разрядов α_m и β_m обозначается одной цифрой σ_m . При этом образуется код

$$K(Z) = \dots \sigma_m \dots$$

комплексного числа $Z = X_\alpha + j \cdot X_\beta$ по основанию $\rho = -2$ с разрядами, принимающими одно из четырех значений:

$$\sigma_m \in \{0, 1, j, 1+j\}.$$

Рассмотрим комплексную функцию от действительного целого аргумента m :

$$\rho_2 = \begin{cases} (-2)^{m/2} & \text{if } m - \text{even} \\ j(-2)^{m-1/2} & \text{if } m - \text{odd} \end{cases} \quad (2)$$

При этом рассматриваемый код комплексного числа по основанию (-2) с комплексными значениями разрядов может рассматриваться

как код комплексного числа по основанию ρ_2 с двоичными разрядами. Этому коду соответствует разложение комплексного числа в виде $Z = \sum_m (\sigma_m \rho_2)$. где двоичные разряды

$$\sigma_m = \begin{cases} \alpha_m & \text{if } m - \text{even} \\ j \cdot \beta_m & \text{if } m - \text{odd} \end{cases}. \quad \text{Для иллюстрации запишем коды}$$

некоторых характерных чисел в этой этой системе: $K(2) = 10100$, $K(-2) = 100$, $K(-1) = 101$, $K(j) = 10$, $K(-j) = 1010$, $K(2j) = 101000$.

Второй способ объединения этих двух кодов в единый код комплексного числа состоит в построении последовательности чередующихся разрядов α_m и β_m :

$$\dots \beta_{m+1} \alpha_{m+1} \beta_m \alpha_m \beta_{m-1} \alpha_{m-1} \dots$$

Обозначим $\alpha_m = \sigma_{2m}$, $\beta_m = \sigma_{2m+1}$ и перепишем указанную последовательность в ином виде:

$$\dots \sigma_{k+3} \sigma_{k+2} \sigma_{k+1} \sigma_k \sigma_{k-1} \sigma_{k-2}$$

где $k=2m$. Эта последовательность является двоичным кодом

$$K(Z) = \dots \sigma_m \dots$$

некоторого комплексного числа Z . Можно показать (и это будет сделано ниже), что код, полученный таким способом, является двоичным кодом по основанию $\rho = \pm j\sqrt{2}$, а закодированное число $Z = X_\alpha + \rho \cdot X_\beta$.

Таким образом, некоторые композиции двоичных кодов действительных чисел по основанию $\rho = -2$ образуют коды комплексных чисел. При выполнении алгебраического сложения комплексных чисел такие коды можно рассматривать как простую совокупность кодов действительных чисел и выполнять одноименную операцию с каждой парой действительных чисел независимо. В то же время с такими кодами выполнимы операции умножения и деления. При этом операции умножения и деления состоят, как обычно, из циклов 'сдвиг-сложение'.

1.3. Метод кодирования точек многомерного пространства

Метод кодирования точек многомерного евклидова пространства должен устанавливать некоторое соответствие между этими точками и кодами из некоторого множества. Это соответствие, вообще говоря, может быть не взаимно-однозначным. Но для возможности однозначного декодирования каждому коду должна соответствовать только одна точка кодируемого пространства. В то же время даже ограниченная область пространства содержит несчетное множество точек. Следовательно, множество соответствующих кодов также несчетно и среди них должны быть коды с бесконечным числом разрядов (**бесконечные коды**). Однако в практике вычислений могут использоваться только **конечные коды**, а множество конечных кодов ограничено.

Для того, чтобы в этих условиях сохранить соответствие между кодами и точками пространства, естественно ограниченную кодируемую область G разбить на ограниченное множество областей δ определенного размера и конфигурации так, чтобы каждая точка области G находилась в одной из областей δ . Тогда между множеством конечных кодов и множеством областей δ можно установить взаимно-однозначное соответствие.

Такой способ кодирования точек многомерного пространства является приближенным. Действительно, всем точкам $Z_j \in \delta_i$ соответствует единственный код K_i . Однако при декодировании кода K_i образуется единственная точка Z_i . Обозначим радиус-вектор точки Z символом \bar{Z} . Разность $\Delta Z_j = |\bar{Z}_j - \bar{Z}_i|$ определяет абсолютную погрешность кодирования точки Z_j .

В качестве иллюстрации рассмотрим рис. 1, где изображена область Z_j двумерного пространства, разбитая на области δ .

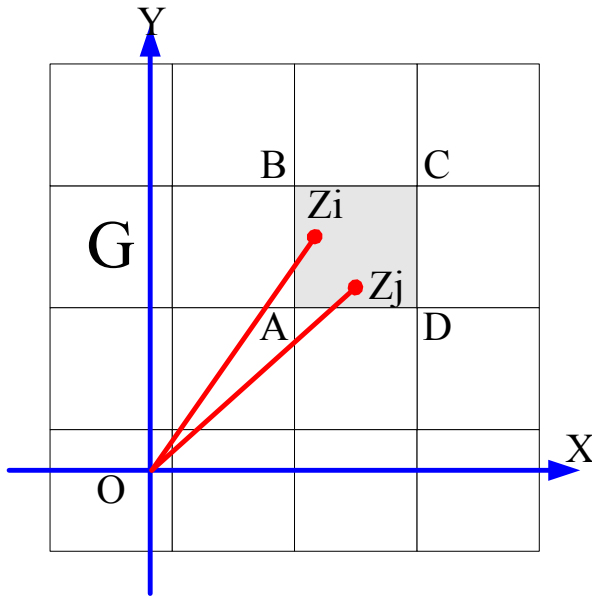


Рис. 1. Кодирование двумерной области

На этом рисунке выделена область $\delta_i = ABCD$, причем области δ_i принадлежит также ее нижняя (AD) и правая (CD) границы. В области δ_i выделена базисная точка Z_i и некоторая точка $Z_j \in \delta_i$. Длина отрезка ΔZ_j характеризует абсолютную погрешность кодирования точки Z_j .

Итак, изложенный принцип кодирования точек многомерного пространства заключается в следующем:

- ограниченная область G кодируемого пространства разделяется на ограниченное множество равных областей δ_i ($i=1, 2, \dots, N$), причем $G = \bigcup \delta_i$ и $\delta_i \cap \delta_j = \emptyset$ при $i \neq j$;
- определяется множество конечных кодов K_i ($i=1, 2, \dots, N$);
- между областями и кодами устанавливается взаимно-однозначное соответствие.

При соблюдении этих условий будем говорить, что **система кодирования** области G многомерного пространства удовлетворяет **принципу кодирования** и область G кодируется с **дискретностью** δ . Следующие две леммы очевидны.

Лемма 1. Система кодирования области G удовлетворяет принципу кодирования, если $V=NU$, и обратно, где

U - объем области δ ,

V - объем области G ,

N - мощность множества конечных кодов.

Лемма 2. Система кодирования области G , удовлетворяющая принципу кодирования, является **полной** (то есть любой точке соответствует конечный код), **неизбыточной** (то есть каждой точке соответствует единственный конечный код) и **приближенной** (то есть подмножеству точек-векторов, модуль разности которых не превышает некоторой величины, соответствует один конечный код).

Рассмотрим множество n -разрядных кодов вида

$$K = \alpha_{n-1} \dots \alpha_k \dots \alpha_1 \alpha_0, \quad (3)$$

где α_k - цифра, принимающая одно из R_k значений, причем $R_k > 1$ и целое число.

Лемма 3. Если система кодирования удовлетворяет принципу кодирования, то при увеличении разрядности конечных кодов и сохранении дискретности кодирования объем кодируемой области увеличивается также, как мощность множества конечных кодов, и обратно.

Доказательство. Мощность множества конечных кодов

$$N_n = \prod_{k=1}^n R_k. \quad (4)$$

Пусть это множество кодов удовлетворяет принципу кодирования и кодирует область G_n с дискретностью δ . В соответствии с леммой 1 количество областей δ , содержащихся в области G_n , также равно N_n , а область G_n имеет объем

$$V_n = N_n U. \quad (5)$$

Увеличим теперь разрядность кодов на единицу, то-есть добавим разряд α_n , принимающий одно из R_n значений. Очевидно

$$N_{n+1} = R_n N_n. \quad (6)$$

Пусть новое множество кодов также удовлетворяет принципу кодирования и кодирует область G_{n+1} с той же дискретностью δ . Количество областей δ , содержащихся в области G_{n+1} , равно N_{n+1} , то-есть область G_{n+1} имеет объем

$$V_{n+1} = N_{n+1}U. \quad (7)$$

Совмещая три последние формулы, находим, что

$$V_{n+1} = R_{n+1}V_n, \quad (8)$$

то-есть прямая часть леммы доказана.

По условию обратной части леммы справедливы формулы (5), (6), (8). Из них следует (7), откуда в соответствии с леммой 1 получаем доказательство обратной части данной леммы.

Рассмотрим теперь позиционную систему кодирования. В этой системе каждому позиционному коду

$$K(Z) = \alpha_n \dots \alpha_k \dots \alpha_m$$

соответствует точка Z кодируемого многомерного пространства, имеющая разложение следующего вида:

$$Z = \sum_{k=m}^n \alpha_k \rho^k, \quad (9)$$

где ρ - основание кодирования,

k - номер разряда,

α_k - k -ый разряд кода (цифра или количественный эквивалент, сопоставляемый ей в разложении), принимающий одно из R_k значений.

Заметим, что ρ и α_k также являются точками кодируемого многомерного пространства. Позиционный код называется **бесконечным**, если $m = -\infty$, и **конечным**, если m ограничено. Число n называется **длиной** позиционного кода. Если $R_k = R$, то разложение и код называются R -ми. Итак, будем рассматривать величины α , принимающие значение из множества

$$A_R = \{a_0, a_1, a_2, \dots, a_j, \dots, a_{R-1}\}, \quad (10)$$

содержащего R различных величин a_j . В практике позиционного кодирования существенно то, что R ограничено и не превышает нескольких единиц.

Позиционный код точки Z по основанию ρ будем обозначать и записывать также следующим образом

$$\langle Z \rangle_\rho = \alpha_n \dots \alpha_k \dots \alpha_1 \alpha_0, \alpha_{-1} \alpha_{-2} \dots \alpha_m, \quad (11)$$

размещая запятую между нулевым и (-1)-разрядом (индекс - основание не будет указываться, если значение основания ясно из контекста). Вектор (точка) Z , в коде которого $m \geq 0$, будем называть

ρ -целым. Соответственно определяются ρ -дробные (правильные и неправильные) векторы Z . В частности,

$$\langle \rho \rangle_{\rho} = 10. \quad (12)$$

Совокупность $\langle \rho, A_R \rangle$ основания кодирования ρ и множества A_R будем называть **системой позиционного кодирования**. Будем говорить, что точка многомерного евклидова пространства **представима** в данной системе позиционного кодирования, если ей соответствует разложение вида (9) и позиционный код вида (11), в котором разряды принимают значения из множества (10).

Задача заключается в построении таких позиционных систем кодирования, в которых представима любая точка данного пространства и при этом выполняются условия полноты, неизбыточности и приближенности, определенные в лемме 2.

Смысл построения позиционных систем заключается в упрощении выполнения арифметических операций с точками (векторами) многомерного пространства. С другой стороны, существование позиционных кодов, основанных на разложении (9), возможно лишь в том случае, если в данном пространстве определены операции суммирования векторов и умножения вектора на основание ρ (которое также может быть вектором).

В одномерном и двумерном пространствах умножение на основание ρ (умножение на действительное или комплексное число) соответствует увеличению модуля вектора-множимого в $|\rho|$ раз, то-есть

$$\text{если } Z_2 = Z_1 \rho, \text{ то } |Z_2| = |Z_1| \cdot |\rho|. \quad (13)$$

Следует еще раз отметить, что умножению $Z_1 \rho$ соответствует сдвиг кода $\langle Z_1 \rangle$ на один разряд влево *в любом пространстве*. Мы потребуем, чтобы условие (13) выполнялось также *для любого кодируемого пространства* и докажем некоторое условие существования позиционной системы счисления, используя эти два факта.

Теорема 1. Необходимым и достаточным условием того, что любая точка h -мерного евклидова пространства, в котором выполняется условие (13), представима в данной системе позиционного кодирования, является условие

$$|\rho|^h = R. \quad (14)$$

Доказательство. Каждый код $\langle Z_2 \rangle_\rho$ длины $(n+h)$ при $m = -\infty$ может быть получен сдвигом на h разрядов влево некоторого кода $\langle Z_1 \rangle_\rho$ длины n . Но в соответствии с (12) такой сдвиг эквивалентен умножению на основание, то-есть $Z_2 = Z_1 \rho^h$. При этом из соотношения (13) следует, что $|Z_2| = |Z_1| \cdot |\rho|^h$. Следовательно, линейные размеры кодируемой области увеличиваются в $|\rho|^h$ раз (кроме того, кодируемая область, вообще говоря, поворачивается относительно предыдущего положения). Таким образом, объемы областей G_n и G_{n+h} связаны соотношением

$$V_{n+h} = |\rho|^h V_n. \quad (15)$$

Очевидно, ограничение m не изменяет объем кодируемой области. Появляется лишь дискретность кодирования $\delta = G_{m-1}$. Учитывая (14), из (15) получаем

$$V_{n+h} = R V_n. \quad (16)$$

Сравнивая (16) и (8), из леммы 3 находим, что система позиционного кодирования при $m < -\infty$ удовлетворяет принципу кодирования, то есть, вследствие леммы 2, является полной, избыточной и приближенной. Теорема доказана.

1.4. Арифметические системы кодирования

Среди позиционных систем кодирования наибольший интерес представляют такие, к которым применимы простые алгоритмы сложения и умножения. Именно такие системы мы и будем рассматривать в дальнейшем, но предварительно определим их более строго.

Определение 1. Система $\langle \rho, A_R \rangle$ позиционного кодирования называется **арифметической**, если выполнены следующие условия

- число (-1) является ρ -целым,
- сумма и произведение любых пар векторов, принадлежащих множеству A_R , являются ρ -целыми.

Заметим, что условие (13) может выполняться и для неарифметической системы.

Лемма 4. Если в арифметической позиционной системе представимы векторы Z_1 и Z_2 , то в этой системе представимы и векторы $-Z_1$, $-Z_2$, $Z_1 + Z_2$, $Z_1 \cdot Z_2$.

Справедливость леммы вытекает из того, что, как будет показано ниже, для арифметических позиционных систем существуют алгоритмы арифметических операций.

Определение 2. Позиционная система $\langle \rho, A_R \rangle$ называется **нормальной**, если $A_R = B_R$, где $B_R = \{0, 1, 2, \dots, R-1\}$.

Лемма 5. Нормальная система счисления, в которой

$$R = \sum_{k=1}^n \alpha_k \rho^k, \quad (17)$$

$$-R = \sum_{k=1}^w \beta_k \rho^k, \quad (18)$$

то есть коды чисел R и $-R$ являются ρ -целыми и имеют нулевое значение нулевого разряда, является арифметической.

Доказательство. Любое число из множества B_R находится в пределах $0 \leq a_j \leq (R-1)$. Следовательно, для чисел из этого множества выполняются соотношения $-a_j = a_k - R$ и

$a_j + a_k = a_m + R$, если $a_j + a_k \geq R$. Учитывая условия леммы, заключаем, что числа $(-a_j)$ и $(a_j + a_k)$ являются ρ -целыми. Очевидно, произведение $a_j a_k$ можно представить суммой чисел из множества B_R . По индукции в силу существования алгоритма сложения заключаем, что такая сумма также является ρ -целой. Таким образом, условия определения 1 выполняются. Следовательно, рассматриваемая система является арифметической.

Лемма 6. Нормальная система счисления, в которой число R имеет разложение вида (17) и

$$R = \sum_{k=1}^m \alpha_k, \quad (19)$$

является арифметической.

Доказательство. Как следует из (17) и (19), в лемме рассматриваются системы, в которых

$$R = \sum_{k=1}^n \alpha_k \rho^k = \sum_{k=1}^n \alpha_k.$$

Рассмотрим следующий алгоритм:

α_3	α_2	α_1	0				переносы
	α_3	α_2	α_1	0			переносы
		α_3	α_2	α_1	0		переносы
			α_3	α_2	α_1	$0 = \langle R \rangle_\rho$	слагаемое 1
				β_2	β_1	$0 = \langle X \rangle_\rho$	слагаемое 2
0	0	0	0	0	0	0	сумма

Здесь код числа R складывается с кодом некоторого числа X , разряды которого образуются таким образом, чтобы

$$\alpha_1 + \beta_1 = R \text{ и } \alpha_1 + \alpha_2 + \beta_2 = R.$$

При этом и вследствие (19) сложение цифр каждого столбца образует число R , которое формирует перенос и нулевой разряд суммы. В результате образуются бесконечные переносы и нулевая сумма. Следовательно, $X = -R$. Очевидно, такой алгоритм образования кода числа $-R$ осуществим при любом R , соответствующем разложению (17) или, что одно и то же, при любом коде числа R вида

$$\langle R \rangle_\rho = \alpha_m \dots \alpha_2 \alpha_1 0.$$

Результатом этого алгоритма является код числа $-R$ вида

$$\langle -R \rangle_{\rho} = \beta_w \dots \beta_2 \beta_1 0.$$

Этот код соответствует разложению (18). Тем самым лемма доказана.

Заметим, что разложения (17) и (18) можно рассматривать как систему двух степенных уравнений относительно неизвестного ρ .

Решая ее, можно, вообще говоря, определить некоторую систему кодирования. Однако такой прием далеко не всегда приводит к положительному результату потому, что данная система либо не разрешима аналитически, либо не совместима, либо дает в качестве решения результат, не удовлетворяющий условию теоремы 1, либо дает в качестве решения действительное число.

Леммы 4, 5, 6 будут использованы далее при поиске нормальных позиционных систем кодирования.

1.5. Коды действительных чисел

Для действительных чисел размерность кодируемого пространства $h=1$. Следовательно, для позиционных кодов действительных чисел необходимо соблюдать условие $|\rho| = R$.

Широко известны и распространены позиционные коды действительных чисел, в которых $\rho = R$ и разряды принимают значения из множества B_R . Сюда относятся обычные десятичные ($R=10$) и двоичные ($R=2$) коды. Однако такими кодами нельзя изобразить отрицательные действительные числа, для представления которых приходится применять искусственные приемы, в частности, использовать обратные и дополнительные коды, что вызывает ряд неудобств.

В тоже время существуют два способа построения позиционных кодов, пригодных для изображения действительных - положительных и отрицательных чисел. Первый из них заключается в том, что разрядам придают положительные и отрицательные значения из множества

$$D_R = \{-r_1, -r_1 + 1, \dots, -1, 0, 1, \dots, r_2 - 1, r_2\}$$

причем $R = r_1 + r_2 + 1$, $r_1 \neq 0$, $r_2 \neq 0$, а основание, по-прежнему, оставляют равным R (при $r_1 = 0$ множество D_R превращается в множество B_R). Другой способ основан на применении отрицательного основания $\rho = -R$. Величины разрядов при этом могут принимать значения либо из множества B_R , либо из множества D_R . Итак, известные результаты, относящиеся к позиционному кодированию действительных чисел, формулируются следующим образом.

Теорема 2. Любое действительное число представимо в системах

$$\langle R, B_R \rangle, \langle R, D_R \rangle, \langle -R, B_R \rangle, \langle -R, D_R \rangle.$$

Итак, существует четыре системы кодирования действительных чисел:

система $\langle R, B_R \rangle$, например $\langle 5, \{0, 1, 2, 3, 4\} \rangle$;

система $\langle R, D_R \rangle$, например $\langle 5, \{-2, -1, 0, 1, 2\} \rangle$;

система $\langle -R, B_R \rangle$, например $\langle -5, \{0, 1, 2, 3, 4\} \rangle$;

система $\langle -R, D_R \rangle$, например $\langle -5, \{-2, -1, 0, 1, 2\} \rangle$.

Приведем примеры пятиричных кодов некоторых чисел в рассмотренных системах, обозначая величины -1 и -2 цифрами $\bar{1}$ и $\bar{2}$:

1. $K(16) = +31$, $K(-13) = -23$,
2. $K(16) = 1\bar{2}1$, $K(-13) = \bar{1}22$,
3. $K(16) = 121$, $K(-13) = 32$,
4. $K(16) = 121$, $K(-13) = \bar{1}\bar{2}2$.

Здесь следует обратить внимание на то, что коды в первой из этих систем сопровождаются знаками '+' и '-', которые отсутствуют во всех остальных системах, поскольку в них знак числа вместе с модулем определяются значениями разрядов кода.

Важно отметить, что среди указанных систем существуют лишь две системы двоичного кодирования, а именно системы с цифрами $\{0, 1\}$ и основаниями '2' и '-2'.

1.6. Коды комплексных чисел

Докажем вначале несколько теорем существования нормальных арифметических систем кодирования с комплексным основанием, обозначая через j мнимую единицу.

Теорема 3. Любое комплексное число представимо в нормальной системе кодирования по комплексному основанию ρ и эта система является арифметической, если

$$|\rho| = \sqrt{R} \quad (20)$$

и выполняются условия (17), (19).

Доказательство. Для комплексных чисел размерность кодируемого пространства $h=2$ и при любом ρ выполняется условие (13). Отсюда и из (20) следует, что выполняются условия теоремы 1. Следовательно, любое комплексное число представимо в данной системе кодирования. Далее, условия (17) и (19) являются условиями леммы 6. Следовательно, данная система является арифметической.

Теорема 3 дает возможность свести доказательство теорем о представимости любого комплексного числа в нормальной системе кодирования и арифметичности этой системы к доказательству того, что выполняется условие (19) и ρ является комплексным корнем уравнения (17). Именно этим методом доказательства мы и воспользуемся в дальнейшем.

Теорема 4. Любое комплексное число представимо в нормальной системе кодирования по комплексному основанию

$$\langle \rho = \sqrt{2}e^{\pm j\pi/2}; B_2 \rangle \text{ или } \langle \rho = -1 \pm j; \{0, 1\} \rangle$$

и эта система является арифметической.

Доказательство. Предположим, что $\langle 2 \rangle_\rho = 1100$, Это условие равносильно уравнению $\rho^3 + \rho^2 = 2$. Его решение совпадает с условием данной теоремы. Следовательно, выполняется условие (17). Очевидно, что условие (19) также выполняется, поскольку $R=2$. В силу теоремы 3 данная теорема доказана. Для иллюстрации запишем коды некоторых характерных чисел в системе с основанием $\rho = (j-1)$, обозначив через $\bar{\rho}$ число, сопряженное числу ρ : $K(2) = 1100$, $K(-2) = 11100$, $K(-1) = 11101$, $K(j) = 11$, $K(-j) = 111$, $K(\bar{\rho}) = 110$.

Теорема 5. Любое комплексное число представимо в нормальной системе кодирования по комплексному основанию ρ и эта система является арифметической, если

$\rho = \sqrt{R}e^{j\varphi}$, $\varphi = \pm \arccos(-\beta / 2\sqrt{R})$, $\beta < (R, 2\sqrt{R})_{\min}$
и β - целое положительное число.

Доказательство. Предположим, что $\langle R \rangle_\rho = 1\alpha_2\alpha_1 0$, где $1 + \alpha_2 + \alpha_1 = R$, $\alpha_2 = \beta - 1$. Это условие равносильно уравнению

$$\rho^3 + (\beta - 1)\rho^2 + (R - \beta)\rho = R.$$

Его решение дает значение, приведенное в условиях теоремы. В силу теоремы 3 данная теорема доказана.

Для иллюстрации запишем коды некоторых характерных чисел в этой системе, обозначив через $\bar{\rho}$ число, сопряженное числу ρ :

$$K(R) = 1 (\beta - 1) (R - \beta) 0,$$

$$K(-R) = 1 \beta 0,$$

$$K(-1) = 1 \beta (R - 1),$$

$$K(\bar{\rho}) = 1 (\beta - 1) (R - \beta),$$

$$K(-\bar{\rho}) = 1 \beta,$$

$$K(-\rho) = 1 \beta (R - 1) 0,$$

$$K(\rho - \bar{\rho}) = 2 \beta,$$

$$K(\rho + \bar{\rho}) = 1 \beta (R - \beta).$$

В связи с тем, что β может принимать несколько значений при постоянном R , существует несколько типов позиционных кодов в системах рассматриваемого вида. В качестве примера в табл. 1 приведены возможные коды числа R при различных R и β .

Таблица 1. Коды числа R

$R \setminus \beta$	1	2	3	4	5
2	1010				
3	1020	1110			
4	1030	1120	1210		
5	1040	1130	1220	1310	
6	1050	1140	1230	1320	
7	1060	1150	1240	1330	1420
8	1070	1160	1250	1340	1430
9	1080	1170	1260	1350	1440

Для иллюстрации запишем коды некоторых характерных чисел в системе с основанием $\rho = \frac{1}{2}(-1 + j\sqrt{7})$, обозначив через $\bar{\rho}$ число, сопряженное числу ρ : $K(2)=1010$, $K(-2)=110$, $K(-1)=111$, $K(\bar{\rho})=101$, $K(-\rho)=1110$, $K(-\bar{\rho})=11$, $K(j\sqrt{7})=10101$, $K(-j\sqrt{7})=1110011$.

Из систем теоремы 5 можно выделить группы с фиксированным значением аргумента основания, например

$$\varphi = \pm 2\pi/3, \text{ если } \beta = \sqrt{R}, \text{ то-есть при } R=4, 9, 16, 25, \dots;$$

$$\varphi = \pm 3\pi/4, \text{ если } \beta = \sqrt{2R}, \text{ то-есть при } R=8, 18, 32, 50, \dots;$$

$$\varphi = \pm 5\pi/6, \text{ если } \beta = \sqrt{3R}, \text{ то-есть при } R=12, 27, 48, 75, \dots$$

Рассмотрим теперь позиционную систему более общего вида.

Теорема 6. Любое комплексное число представимо в системе кодирования $\langle \rho = 2e^{j\pi/3}, A_4 \rangle$, $A_4 = \{0, 1, e^{2j\pi/3}, e^{-2j\pi/3}\}$ и эта система является арифметической.

Доказательство. Заметим, что $(-2)^k = l_k \rho^k$, где

$$l_k = \{1, e^{2j\pi/3}, e^{-2j\pi/3}\}$$

соответственно при $k = (3m, 3m+1, 3m+2)$, где m - целое. Очевидно, $l_k \in A_4$. Следовательно, любая степень числа '-2' представима в указанной системе кодирования одним разрядом. В силу теоремы 2

любое действительное число X представимо в виде разложения по основанию -2 . Но каждый разряд такого разложения, представляющий степень числа -2 или 0, может быть заменен разрядом разложения в указанной системе кодирования, то есть любое действительное число представимо в этой системе кодирования.

Любое комплексное число Z может быть представлено как $Z = u_1 + u_2 e^{2j\pi/3} + u_3 e^{-2j\pi/3}$, где u_1, u_2, u_3 - некоторые действительные числа. В этой сумме все составляющие представимы в указанной системе счисления поскольку сомножители действительных чисел u_1, u_2, u_3 принадлежат множеству A_4 .

Если эта система является арифметической, то в ней представима и данная сумма, то есть любое комплексное число. Остается показать, что указанная система является арифметической. Для этого составим таблицы попарного умножения, суммирования и таблицу инвертирования (умножения на -1) цифр из множества A_4 - см. таблицы 2, 3, 4. Для удобства записи эти цифры обозначены символами 0, 1, c, d. Как видно из этих таблиц, в рассматриваемой системе кодирования выполнены все условия определения 1. Следовательно, эта система является арифметической, что и требовалось показать.

Таблица 2. Одноразрядное умножение

*	0	1	c	d
0	0	0	0	0
1	0	1	c	d
c	0	c	d	1
d	0	d	1	c

Таблица 3. Одноразрядное сложение

+	0	1	c	d
0	0	1	c	d
1	1	dc0	1d	dc
c	c	1d	d10	c1
d	d	dc	c1	c10

Таблица 4. Инвертирование разряда

x	0	1	c	d
-x	0	c1	dc	1d

Заметим, что в этой системе очень простой вид имеют коды комплексных чисел вида e^{jk60° , где k - целое число - см. табл. 4а. Кроме того, для этой системы в табл. 4в представлены коды чисел 2^k и $(-2)^k$, где k - целое число.

Таблица 4а. Коды чисел e^{jk60° .

φ	0	60	120	180	240	300
код	00	1d	0c	c1	0d	dc

Таблица 4в. Коды чисел 2^k и $(-2)^k$.

k	$(-2)^k$	2^k
-4	0.000d	0.000d
-3	0.001	0.0c1
-2	0.0c	0.0c
-1	0.d	1.d
0	1	1
1	c0	dc0
2	d00	d00
3	1000	c1000
4	c000	c000

Далее мы только более строго изложим результаты, полученные в разделе 2.

Теорема 7. Любое комплексное число Z представимо в позиционной системе счисления $\langle \rho = -R, A_R^2 \rangle$, где множество A_R^h состоит из комплексных чисел $r_m = \alpha_m^1 + j\alpha_m^2$, а числа $\alpha_m \in B_R$.

В частности, существует система $\langle -2, \{0, 1, j, 1+j\} \rangle$.

Теорема 8. Любое комплексное число Z представимо в нормальной позиционной системе $\langle \pm j\sqrt{R}, B_R \rangle$.

Например, существует система $\langle \pm j\sqrt{2}, \{0,1\} \rangle$. Для иллюстрации запишем коды некоторых характерных чисел в системе $\rho = j\sqrt{2}$: $K(2)=10100$, $K(-2)=100$, $K(-1)=101$, $K(j\sqrt{2})=10$, $K(-j\sqrt{2})=1010$.

Очевидно, для систем из теорем 7 и 8 выполняется условие (14), Доказательство этих теорем основано на рассуждениях раздела 2.

Приведем для иллюстрации и сравнения двоичные коды чисел во всех указанных системах кодирования, включая системы кодирования по действительному (положительному и отрицательному) и комплексному основаниям - см. табл. 5.

Таблица 5. Двоичные системы кодирования.

Выделенная система счисления	ρ	$\langle 2 \rangle$	$\langle -2 \rangle$	$\langle -1 \rangle$	Теорема	Рис.
Система 1	ρ_2	10100	100	101	Формула (2)	1
Система 2	$j\sqrt{2}$	10100	100	101	Теорема 8	2
Система 3	$-1 + j$	1100	11100	11101	Теорема 4	3
Система 4	$\frac{1}{2}(-1 + j\sqrt{7})$	1010	110	111	Теорема 5	4
	-2	110	10	11	Теорема 2	
	2	10			Теорема 2	

В дальнейшем мы более подробно остановимся на четырех двоичных системах счисления комплексных чисел – см. столбец «Выделенная система счисления» в табл. 5. На рисунках изображены первые 4 значения базовой функции для выделенных систем счисления.

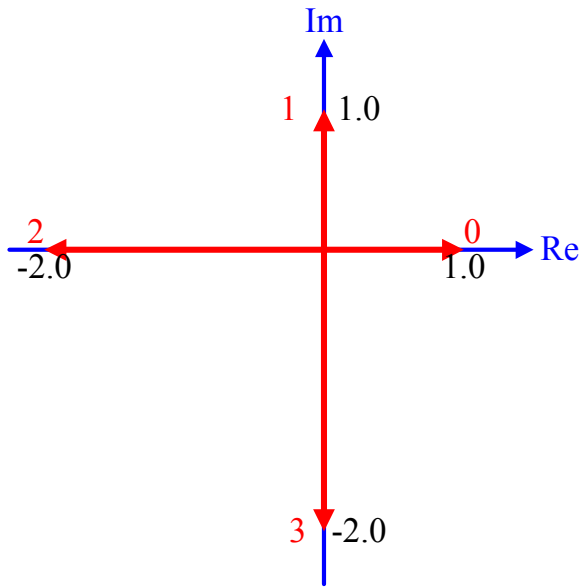


Рис. 1

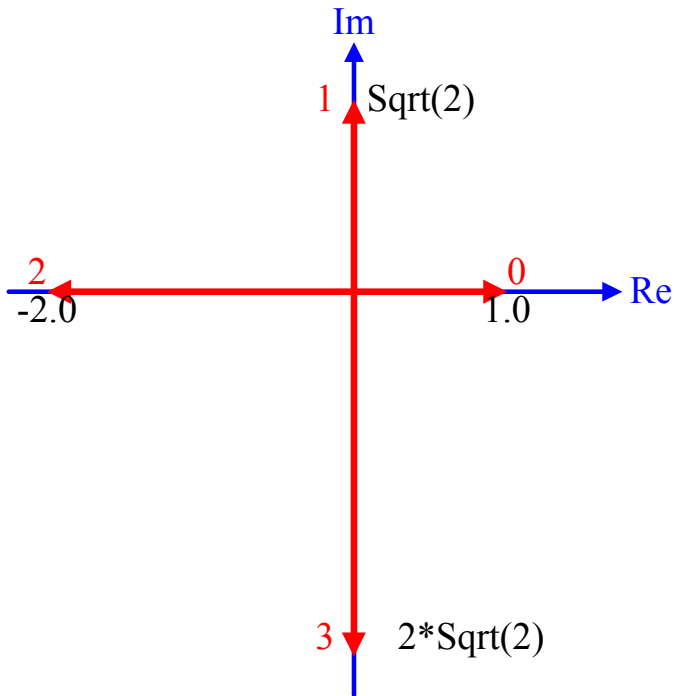


Рис. 2.

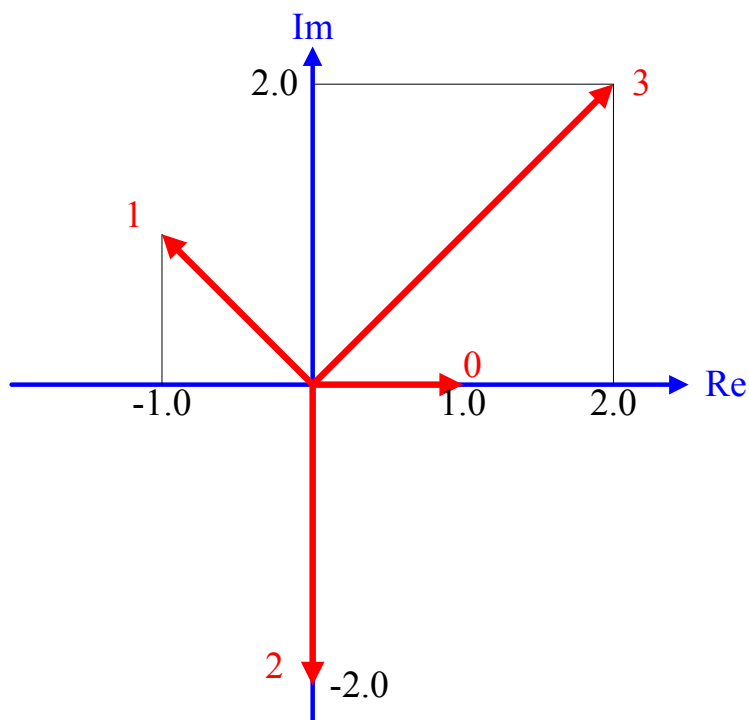


Рис. 3.

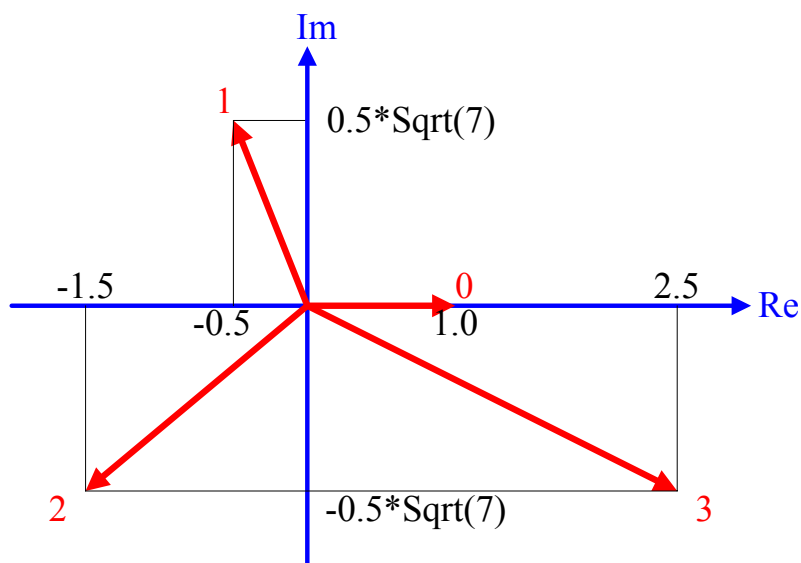


Рис. 4.

Приведем еще табл. 6 четверичных кодов чисел '4' и '-4' во всех рассмотренных выше системах кодирования (в этой таблице '-1' обозначена 'цифрой' 'h').

Таблица 6. Четверичные системы кодирования.

ρ	A_4	$\langle 4 \rangle$	$\langle -4 \rangle$	Теорема
4	{0,1,2,3}	10		2
4	{-1,0,1,2}	10	h0	2
4	{-2,-1,0,1}	10	h0	2
-4	{0,1,2,3}	130	10	2
-4	{-1,0,1,2}	h0	10	2
-4	{-2,-1,0,1}	h0	10	2
$2e^{2j\pi/3}$	{0,1,2,3}	1120	120	5
$2e^{j\pi/3}$	{0,1,c,d}	d00	1d00	6
-2	{0,1,j,1+j}	100	1100	7
ρ_4	{0,1,2,3}	10300	100	8
$\pm 2j$	{0,1,2,3}	10300	100	8

1.7. Коды многомерных векторов

1.7.1. Двоичные коды векторов- способ 1.

Изложенный в разделе 2 метод построения кодов комплексных чисел может быть обобщен и использован для кодирования многомерных векторов. Для этого рассмотрим множество действительных чисел $\{X_i\}$, каждое из которых задано двоичным разложением по основанию $\rho = -2$, то есть

$$X_i = \sum_{(m)} \alpha_m^i \rho^m \quad (i=1, 2, \dots, n).$$

Каждому такому разложению соответствует код

$$K(X_i) = \dots \alpha_m^i \dots$$

Рассмотрим теперь n -мерный вектор

$$Z = E_1 X_1 + E_2 X_2 + \dots + E_n X_n, \quad (21)$$

где $\{E_i\}$ - база n -мерного векторного пространства. Множество кодов $\{K(X_i)\}$ можно при этом трактовать как единый код вектора Z по основанию '-2'. Каждый m -ый разряд этого кода изображается множеством $\{\alpha_m^i\}$ двоичных разрядов. Обозначив эти множества цифрами σ_m , получаем код вектора

$$K(Z) = \dots \sigma_m \dots,$$

соответствующий разложению (21), где вектор

$$r_m = E_1 \alpha_m^1 + E_2 \alpha_m^2 + \dots + E_i \alpha_m^i + \dots + E_n \alpha_m^n \quad (22)$$

изображается цифрой σ_m .

В частности, при $n=2$ образуются коды комплексных чисел по основанию ρ_2 , рассмотренные выше – см. формулу (2). При $n=3$ образуются коды трехмерных векторов, в которых разряды принимают одно из восьми значений:

$$r_m \in \{0, i, j, k, i+j, i+k, j+k, i+j+k\}, \quad (23)$$

где i, j, k - орты прямоугольных координатных осей. Аналогично предыдущему для кодирования трехмерных векторов может быть введена векторная функция действительного целого аргумента m :

$$\mathcal{G}_2 = \left\{ \begin{array}{ll} i(-2)^m & \text{if } m = 3k \\ j(-2)^{m-1} & \text{if } m = 3k + 1 \\ k(-2)^{m-2} & \text{if } m = 3k + 2 \end{array} \right\}, \quad (24)$$

При этом рассматриваемый код трехмерного вектора по основанию \mathcal{G}_2 с векторными значениями разрядов (23) может рассматриваться как код трехмерного вектора по основанию \mathcal{G}_2 с двоичными разрядами. Этому коду соответствует разложение вектора в виде $Z = \sum_m (\alpha_m \mathcal{G}_2)$.

Аналогично, для кодирования n -мерных векторов может быть введена векторная функция действительного целого аргумента m :

$$\mathcal{G}_2^n = \left\{ \begin{array}{ll} i(-2)^m & \text{if } m = nk \\ j(-2)^{m-1} & \text{if } m = nk + 1 \\ \dots & \\ k(-2)^{m-n+1} & \text{if } m = nk + n - 1 \end{array} \right\}, \quad (25)$$

Очевидно, $\rho_2 = \mathcal{G}_2^2$, $\mathcal{G}_2 = \mathcal{G}_2^3$.

1.7.2. Двоичные коды векторов- способ 2.

Построим теперь, как ранее для комплексных чисел, последовательность чередующихся двоичных разрядов α_m^i :

$$\dots \alpha_{m+1}^2 \alpha_{m+1}^1 \alpha_m^n \alpha_m^{n-1} \dots \alpha_m^2 \alpha_m^1 \alpha_{m-1}^n \alpha_{m-1}^{n-1} \dots$$

В других обозначениях эта последовательность является двоичным кодом

$$K(Z) = \dots \alpha_k \dots$$

некоторого вектора Z . При этом основание кодирования также является вектором

$$\rho = E_2^{\sqrt[n]{2}}, \quad (26)$$

где E_2 - второй орт базы $\{E_i\}$ n -мерного векторного пространства. Закодированный вектор Z определяется в данном случае по формуле

$$Z = X_1 + \rho X_2 + \dots + \rho^{i-1} X_i + \dots + \rho^{n-1} X_n. \quad (27)$$

1.7.3. Многозначные коды векторов- способ 2.

Совершенно аналогично строятся позиционные коды векторов (в том числе комплексных чисел и многомерных векторов) на основе объединения позиционных кодов чисел-проекций векторов по основанию $\rho = -R$, где R - целое число. В этом случае, например, вместо функции ρ_2 в качестве основания кодирования комплексных чисел должна рассматриваться функция

$$\rho_R = \left\{ \begin{array}{ll} (-R)^{m/2} & \text{if } m - \text{even} \\ j(-R)^{m-1/2} & \text{if } m - \text{odd} \end{array} \right\}, \quad (28)$$

вместо функции \mathcal{G}_2 в качестве основания кодирования трехмерных векторов должна рассматриваться функция

$$\mathcal{G}_R = \left\{ \begin{array}{ll} i(-R)^{m/3} & \text{if } m = 3k \\ j(-R)^{m-1/3} & \text{if } m = 3k + 1 \\ k(-R)^{m-2/3} & \text{if } m = 3k + 2 \end{array} \right\}, \quad (29)$$

и, вообще, вместо функции \mathcal{G}_2^n для кодирования n -мерных векторов должна рассматриваться функция

$$\mathcal{G}_R^n = \left\{ \begin{array}{ll} i(-R)^m & \text{if } m = nk \\ j(-R)^{m-1} & \text{if } m = nk + 1 \\ \dots & \\ k(-R)^{m-n+1} & \text{if } m = nk + n - 1 \end{array} \right\}. \quad (30)$$

Итак, справедливы следующие теоремы.

Теорема 7а. Если в n -мерном евклидовом пространстве с базой $\{E_i\}$ определена алгебра, то любая точка Z этого пространства представима в позиционной системе счисления $\langle \rho = -R, A_{R^n} \rangle$, где множество A состоит из векторов (22), а числа $\alpha_m \in B_R$.

В частности, для комплексных чисел существует система $\langle \rho = -R, A_{R^2} \rangle$, например, четверичная система $\langle -2, \{0, 1, j, 1+j\} \rangle$, а для трехмерных векторов с ортами $\mathbf{i}, \mathbf{j}, \mathbf{k}$ – восьмиричная система, где каждый разряд принимает значения (23).

Теорема 8а. Если в n -мерном евклидовом пространстве с базой $\{E_i\}$ определена алгебра, то любая точка Z представима в нормальной позиционной системе

$$\langle \rho = \pm E_2 \sqrt[n]{R}, B_R \rangle. \quad (31)$$

В частности, при $R=2$ имеем двоичную систему кодирования векторов по основанию (26). Для комплексных чисел существует

система $\langle \pm j\sqrt{R}, B_R \rangle$, например, двоичная система

$\langle \pm j\sqrt{2}, \{0, 1\} \rangle$, а для трехмерных векторов с ортами $\mathbf{i}, \mathbf{j}, \mathbf{k}$ –

двоичная система $\langle \pm j\sqrt[3]{2}, \{0, 1\} \rangle$. В последней системе имеем:

$$\langle i \rangle = 1, \langle -i \rangle = 1001, \langle 2i \rangle = 1001000, \langle -2i \rangle = 1000;$$

$$\langle j \rangle = 10, \langle -j \rangle = 10010, \langle 2j \rangle = 10010000, \langle -2j \rangle = 10000;$$

$$\langle k \rangle = 100, \langle -k \rangle = 100100, \langle 2k \rangle = 100100000, \langle -2k \rangle = 100000.$$

Для трехмерных векторов с ортами $\mathbf{i}, \mathbf{j}, \mathbf{k}$ существует также четверичная система $\langle \pm j\sqrt[3]{4}, \{0, 1, 2, 3\} \rangle$, где

$$\langle 4i \rangle = 1003000 \text{ и } \langle -4i \rangle = 1000.$$

Очевидно, для систем из теорем 7а и 8а выполняется условие (14).

Глава 2. Точность кодирования

2.1. Область представимых чисел

Рассмотрим числа Z , имеющие в данной системе $\langle \rho; D_1 = \{0, 1, \dots, R-1\} \rangle$ бесконечные разложения вида:

$$Z = \sum_{K=-\infty}^M \alpha_k \rho^k. \quad (1)$$

Эти числа могут быть представлены в виде суммы

$$Z = \sum_{j=0}^{a-1} Z^{(j)}, \quad (2)$$

$$\text{где } Z^{(j)} = \sum_{m=-\infty}^{\left[\frac{M+j}{a} \right]} \alpha_{am-j} \rho^{am-j} \text{ или}$$

$$Z^{(j)} = \rho^{-j} \sum_{m=-\infty}^{\left[\frac{M+j}{a} \right]} \alpha_{am-j} \rho^{am}. \quad (3)$$

(здесь $[\lambda]$ - целая часть от λ).

Пусть a - наименьшее целое число ($\neq 0$), при котором $\arg(\rho^a) = 0$. Тогда, очевидно, $\arg(\rho^{am}) = 0$ и $\arg(Z^{(j)}) = \arg(\rho^{-j})$
или

$$\arg(Z^{(j)}) = -j\varphi, \quad (4)$$

где $\varphi = \arg(\rho)$. В частности, при $j=0$ и $\arg(Z^{(j)}) = 0$, т.е. число $Z^{(0)}$ является действительным положительным. Определим область изменения числа $Z^{(0)}$ при $M=0$. Согласно (3) имеем:

$$Z^{(0)} = \sum_{m=-\infty}^0 \alpha_{am} \rho^{am}. \quad (5)$$

Очевидно, $Z_{\min}^{(0)} = 0$ и $Z_{\max}^{(0)} = \sum_{m=-\infty}^0 (R-1) \rho^{am}$ или $Z_{\max}^{(0)} = (R-1)L$, где

$$L = \frac{\rho^a}{\rho^a - 1}. \quad (6)$$

Таким образом, при $M=0$

$$0 \leq Z^{(0)} \leq (R-1)L. \quad (7)$$

Далее, при $M=0$ имеем: $Z^{(j)} = \rho^{-j} \sum_{m=-\infty}^0 \alpha_{am-j} \rho^{am}$. Область

изменения числа $\sum_{m=-\infty}^0 \alpha_{am-j} \rho^{am}$ совпадает с областью изменения

числа $Z^{(0)}$ - см. (5). Следовательно,

$$0 \leq |Z^{(j)}| \leq |\rho|^{-j} (R-1)L. \quad (8)$$

Формулы (8) и (4) определяют область изменения числа $Z^{(j)}$. В геометрической интерпретации эта область является отрезком

$OA = |\rho|^{-j} (R-1)L$ прямой, расположенной под углом $(-j\varphi)$ на комплексной плоскости - см. рис. 1. Иначе говоря, областью изменения числа $Z^{(j)}$ при $M=0$ является радиус-вектор

$$r_j = (R-1)L \rho^{-j}. \quad (9)$$

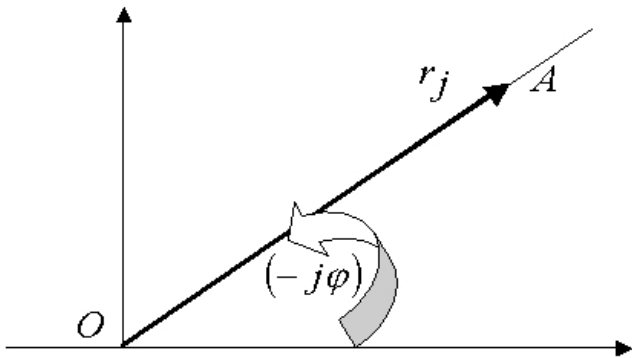


Рис. 1.

Заметим, что

$$r_{j+1} = r_j / \rho. \tag{10}$$

В табл. 1 приведены величины φ , a , L для различных оснований ρ , необходимые для определения векторов r_j . Векторы r_j используются, в свою очередь, для построения области изменения числа Z при $M=0$, которую будем обозначать символом G_0 . В простейшем случае при $\rho=R$, $Z=Z^{(0)}$ область G_0 есть вектор $r_0=R$ (см. (9) и табл. 1.), т.е. отрезок $[0,R]$ действительной оси - см. рис. 2.

Таблица 1.

ρ	R	$-R$	$\sqrt{R} \cdot e^{i\varphi}$			
φ	0	π	$\pm \frac{1}{2}\pi$	$\pm \frac{2}{3}\pi$	$\pm \frac{3}{4}\pi$	$\pm \frac{5}{6}\pi$
a	1	2	4	3	8	12
L	$\frac{R}{R-1}$	$\frac{R^2}{R^2-1}$	$\frac{R^2}{R^2-1}$	$\frac{R\sqrt{R}}{R\sqrt{R}-1}$	$\frac{R^4}{R^4-1}$	$\frac{R^6}{R^6-1}$

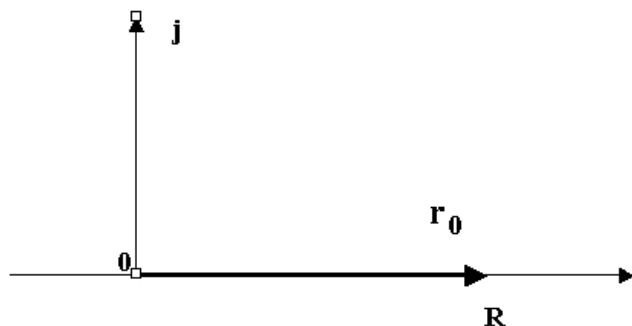


Рис. 2.

При $\rho = -R$ область G_0 для числа $Z = Z^{(0)} + Z^{(1)}$ есть совокупность векторов $r_0 = \frac{R^2}{R+1}$ (см. (9) и табл. 1) и $r_1 = -\frac{R}{R+1}$ (см. (10)), т.е. отрезок $\left[-\frac{R}{R+1}; \frac{R^2}{R+1}\right]$ действительной оси - см. рис. 3.

Для определения области G_0 при $\rho = (\sqrt{R}) \cdot e^{\pm\pi/2}$ необходимо определить предварительно области изменения действительной и мнимой частей числа Z .



Рис. 3.

Очевидно, $\operatorname{Re} Z = Z^{(0)} + Z^{(2)}$ и $\operatorname{Im} Z = Z^{(1)} + Z^{(3)}$. Поэтому область изменения числа $\operatorname{Re} Z$ есть совокупность векторов $r_0 = \frac{R^2}{R+1}$ и $r_2 = -\frac{R}{R+1}$, а область изменения числа $\operatorname{Im} Z$ есть совокупность

векторов $r_1 = \frac{\mp iR\sqrt{R}}{R+1}$ и $r_3 = \frac{\mp i\sqrt{R}}{R+1}$ - здесь векторы

r_0, r_1, r_2, r_3 определены, по-прежнему, при помощи формул (9), (10) и табл. (1). Таким образом, числа $\text{Re}Z$ и $\text{Im}Z$ определены соответственно на отрезке AB действительной оси и на отрезке CD мнимой оси - см. рис. 4. Очевидно, число Z может принимать лишь те значения, для которых $\text{Re}Z$ и $\text{Im}Z$ не выходят из пределов отрезков AB и CD . Следовательно, область G_0 в данном случае представляет собой прямоугольник $EKLM$, стороны которого проходят через точки A, B, C, D перпендикулярно осям.

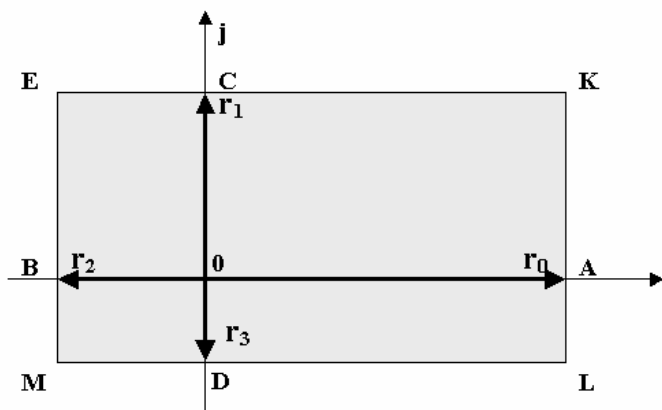


Рис. 4.

Для построения области G_0 при $\rho = (\sqrt{R}) \cdot e^{\pm \frac{2}{3}\pi i}$, также, как и в предыдущем случае, предварительно вычисляются и строятся векторы r_0, r_1 и r_2 - см. рис. 5. Число Z всегда может быть представлено проекциями на оси OA, OC, OB . При этом оно может принимать лишь те значения, при которых указанные проекции не превышают величины отрезков OA, OC, OB . Поэтому область G_0 представляет собой сумму трех параллелограммов, построенных на отрезках OA, OC, OB .

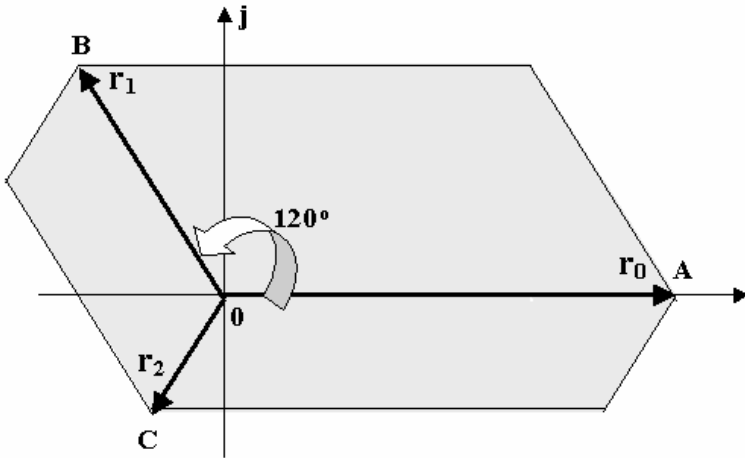


Рис. 5.

Более сложные построения приходится производить для определения области G_0 при основаниях, где j принимает 8 и 12 значений - см. табл. 1. В качестве примера произведем эти построения для $\rho = i - 1$ $\left(\rho = \sqrt{2} e^{i\varphi}; R = 2; \varphi = \frac{3}{4}\pi; a = 8; L = \frac{16}{15} \right)$.

По формулам (9) и (10) находим: $r_0 = \frac{16}{15}$; $r_{j+1} = \frac{r_j}{\sqrt{2}} e^{-i\varphi}$.

Используя полученные формулы, строим векторы r_j на комплексной плоскости – см. рис. 6. Заметим, что модуль вектора r_{j+1} может быть определен как сторона квадрата, диагональю которого является модуль вектора r_j .

Эти векторы определяют две пары взаимно-перпендикулярных осей координат: (15) \perp (37) и (26) \perp (48). В каждой из полученных осей координат построим, аналогично рис. 4, области, определяемые тетрадами векторов: в системе осей (15) и (37) - область G_0^I (прямоугольник ABCD) изменения числа $Z^{(0)} + Z^{(2)} + Z^{(4)} + Z^{(6)}$; в системе осей (26) и (48) - область G_0^{II} (прямоугольник EKLМ) изменения числа $Z^{(1)} + Z^{(3)} + Z^{(5)} + Z^{(7)}$. Очевидно, область G_0 определяется как совокупность областей G_0^I и G_0^{II} .

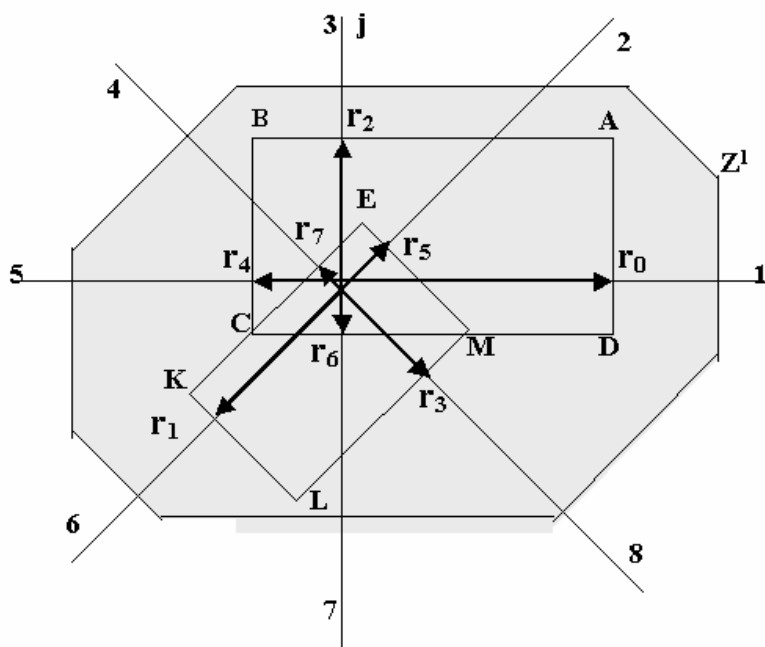


Рис. 6.

Для определения границ области G_0 необходимо точку O , принадлежащую системе координат (26) и (48), поместить на границу области G_0^I и перемещать по этой границе без поворота осей (26) и (48). Площадь, закрываемая прямоугольником $EKLM$ при перемещении, в сумме с площадью $ABCD$ и будет составлять искомую область G_0 . Таким образом, область G_0 принимает вид фигуры, заштрихованной на рис. 6.

Аналогично строится область G_0 при других основаниях, где $a=(8, 12)$, за исключением того, что при $a=12$ область G_0 определяется как совокупность трех прямоугольных областей.

Итак, область G_0 может быть построена для любых ρ и Z при $M=0$. Нетрудно заметить, что при $M \neq 0$ область G_M изменения числа Z может быть получена подобным преобразованием и поворотом области G_0 , причем коэффициент подобия равен

$|\rho|^M$, а угол поворота равен $\arg \rho^M$. Построенная таким образом

область G_M является областью определения чисел Z , представимых бесконечными кодами вида:

$$K(Z) = \alpha_M \dots \alpha_K \dots, \quad (11)$$

где M фиксировано. В частности, в эту область входят и числа, имеющие конечные коды, а также коды, у которых ограниченное число старших разрядов равно нулю.

Область G_M позволяет определить область изменения представимого бесконечным кодом числа Z с фиксированным аргументом. Для этого достаточно из начала координат провести луч под углом $\arg Z$. Отрезок этого луча, заключенный в области G_M , и является искомой областью изменения. В частности, *область изменения действительной правильной ρ -ой дроби есть отрезок действительной оси, заключенный в области G_{-1} .*

2.2. Область представимых модулей

Граница области G_M изображает функцию максимального модуля, представимого кодом (11) числа, имеющего данный аргумент. Эта функция интересна тем, что по ней нетрудно найти число Z' с максимально возможным модулем. Точка, изображающая это число, имеет наибольший радиус-вектор - см., например, рис. 6. Модуль числа Z' при данном ρ является функцией только от M . Поэтому обозначим:

$$|Z'| = \psi(M)$$

Очевидно,

$$\psi(M) = |\rho|^M \cdot \psi(0). \quad (12)$$

В частности, максимальный модуль правильной ρ -ой дроби

$$\psi(-1) = \frac{1}{|\rho|} \cdot \psi(0) \quad (13)$$

Пример 1.

При $\rho = R$ из рис. 2 находим: $\psi(0) = R$; $\psi(-1) = 1$.

При $\rho = -R$ из рис. 3 находим: $\psi(0) = \frac{R^2}{R+1}$; $\psi(-1) = \frac{R}{R+1}$.

При $\rho = \pm i\sqrt{R}$ из рис. 4 находим:

$$\psi(0) = \sqrt{|r_0|^2 + |r_1|^2} = \frac{R\sqrt{R(R+1)}}{R+1}; \quad \psi(-1) = \frac{R\sqrt{R+1}}{R+1}.$$

При $\rho = i - 1$ из рис. 6 находим: $\psi(0) \approx \frac{72}{50} \cdot r_0 = 1,53$; $\psi(-1) \approx 1,1$.

Рассмотрим код, в котором старший разряд $\alpha_M = 1$:

$$K(Z) = 1\alpha_{M-1} \dots \alpha_k \dots$$

Будем называть такой код *приведенным*. Пусть номер старшего разряда приведенного бесконечного кода $M=1$; область Q_1 чисел, представленных таким кодом, образуется из области G_0 смещением

начала координат из точки 0 в точку 0_1 такую, что $\overline{0_1 0} = \rho$ - см. рис. 7. Область G_0 , построенная в осях (11) и (22), является областью Q_1 в осях (33) и (44). Область Q_M чисел, представленных приведенным кодом, в котором $M \neq 1$, образуется аналогично области G_M подобным преобразованием и поворотом области Q_1 . При этом коэффициент подобия равен $|\rho|^{M-1}$, а угол поворота равен $\arg \rho^{M-1}$.

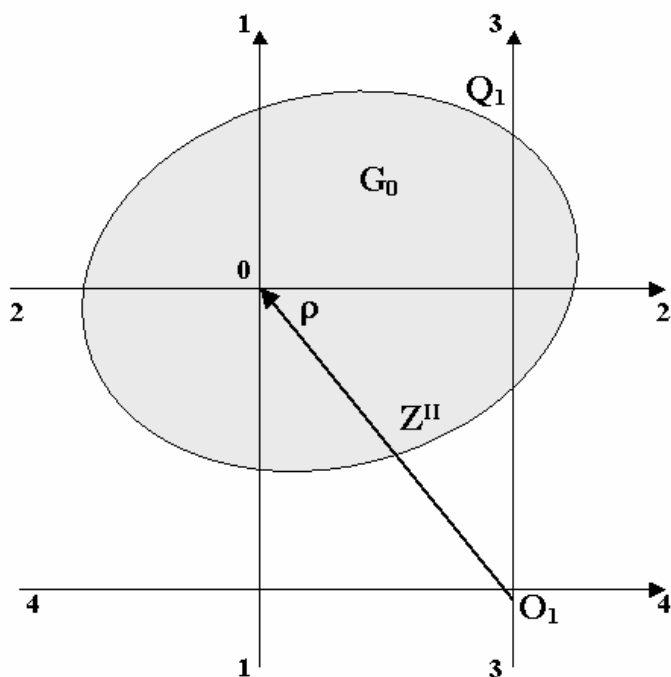


Рис. 7.

В области Q_M нетрудно найти точку, изображающую число Z'' с минимально возможным модулем. Эта точка имеет наименьший радиус-вектор – см. например, рис. 7. Модуль числа Z'' при данном ρ является функцией только от M . Поэтому обозначим:

$$|Z''| = \phi(M).$$

Очевидно,

$$\phi(M) = |\rho|^{M-1} \cdot \phi(1). \quad (14)$$

В частности, минимальный модуль правильной ρ -ой дроби

$$\phi(-1) = \frac{1}{|\rho|^2} \cdot \phi(1) \quad (15)$$

Пример 2.

Производя построения, аналогичные рис. 7, находим:

$$\text{при } \rho = R: \phi(1) = R; \phi(-1) = \frac{1}{R}$$

$$\text{при } \rho = -R: \phi(1) = \frac{R}{R+1}; \phi(-1) = \frac{1}{R(R+1)}$$

$$\text{при } \rho = \pm i\sqrt{R}: \phi(1) = \frac{\sqrt{R}}{R+1}; \phi(-1) = \frac{1}{\sqrt{R}(R+1)}$$

$$\text{при } \rho = i-1: \phi(1) \approx 0,28; \phi(-1) \approx 0,14$$

Рассмотрим код, в котором старший разряд $\alpha_M \neq 0$:

$$K(Z_M) = \alpha_M \dots \alpha_K \dots$$

Будем называть такой код *нормализованным*. Заметим, что частным случаем нормализованного кода является приведенный код, а для двоичных кодов эти понятия совпадают. Модуль числа Z_M представленного нормализованным кодом, заключен в пределах:

$$\phi(M) \leq |Z_M| \leq \psi(M) \quad (16)$$

Последняя формула показывает область изменения модуля числа, представленного кодом с фиксированным номером старшего значащего разряда. Будем обозначать его как НСЗР или символом Ω (в формулах). Из формулы (16) следует, что между НСЗР и модулем $|Z_M|$ в общем случае не существует однозначной зависимости. Область изменения модуля $|Z_M|$ изображена на рис. 8 заштрихованной полосой, ограниченной функциями $\phi(M)$ и $\psi(M)$. Эти функции являются показательными – см. рис. (12) и (14).

Область изменения модуля при конкретном значении НСЗР = M_1 , изображается здесь вертикальным отрезком ab .

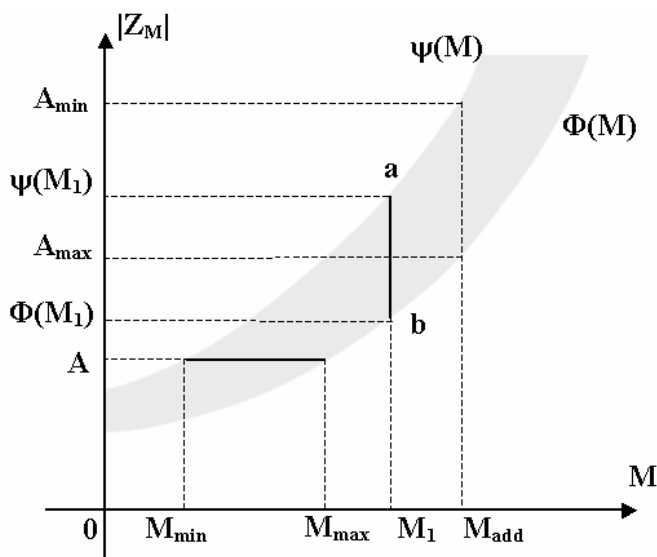


Рис. 8.

Вид функций $\psi(M)$ и $\phi(M)$ для двоичных кодов определяется в следующем примере.

Пример 3.

Используя результаты, полученные в примерах 1 и 2, и формулы (12) и (14) для двоичных кодов находим:

$$\text{при } \rho = 2: \quad 2^M \leq |Z_M| \leq 2 \cdot 2^M;$$

$$\text{при } \rho = -2: \quad \frac{1}{3} \cdot 2^M \leq |Z_M| \leq \frac{4}{3} \cdot 2^M;$$

$$\text{при } \rho = i\sqrt{2}: \quad \frac{1}{3} \cdot 2^{\frac{M}{2}} \leq |Z_M| \leq \frac{2\sqrt{6}}{3} \cdot 2^{\frac{M}{2}};$$

$$\text{при } \rho = i-1: \quad 0,2 \cdot 2^{\frac{M}{2}} \leq |Z_M| \leq 1,53 \cdot 2^{\frac{M}{2}}.$$

Числа, имеющие равный модуль, в общем случае могут быть представлены кодами с разными НСЗР. Найдем пределы изменения кодов чисел с данным модулем A - см. рис. 8. Обозначим:

M_{\min} - минимальный НСЗР кода числа с данным модулем A ;

M_{\max} - максимальный НСЗР кода числа с данным модулем A .

Имеем:

$$\psi(M_{\min}) \leq A \leq \phi(M_{\max}). \quad (17)$$

Точное равенство в общем случае не выполняется в связи с тем, что величина M и, следовательно, функции $\psi(M)$ и $\phi(M)$ принимают дискретные значения. Из формулы (17) нетрудно определить M_{\min} и M_{\max} как предельные значения аргумента M в функциях $\psi(M)$ и $\phi(M)$, при которых неравенства еще выполняются.

Пример 4.

Определим пределы изменения НСЗР (Ω) двоичных кодов чисел с модулем $A=2^C$. Пользуясь результатами примера 3, находим:

- при $\rho = 2$: $2 \cdot 2^{M_{\min}} \leq 2^C$, т.е. $M_{\min} \leq C-1$,
 $2^{M_{\max}} \geq 2^C$, т.е. $M_{\max} \geq C$; таким образом, $C-1 \leq \Omega \leq C$;
- при $\rho = -2$: $\frac{4}{3} \cdot 2^{M_{\min}} \leq 2^C$, т.е. $M_{\min} \leq C$, $\frac{1}{3} \cdot 2^{M_{\max}} \geq 2^C$,
 т.е. $M_{\max} \geq C+2$; таким образом, $C \leq \Omega \leq C+2$;
- при $\rho = i\sqrt{2}$: $\frac{2\sqrt{6}}{3} \cdot 2^{\frac{1}{2}M_{\min}} \leq 2^C$, т.е. $M_{\min} \leq 2C-1$,
 $\frac{1}{3} \cdot 2^{\frac{1}{2}M_{\max}} \geq 2^C$, т.е. $M_{\max} \geq 2C+4$, таким образом,
 $2C-1 \leq \Omega \leq 2C+4$;
- при $\rho = i-1$: $1,53 \cdot 2^{\frac{1}{2}M_{\min}} \leq 2^C$, т.е. $M_{\min} \leq 2C-1$,
 $0,2 \cdot 2^{\frac{1}{2}M_{\max}} \geq 2^C$, т.е. $M_{\max} \geq 2C+4$; таким образом,
 $2C-1 \leq \Omega \leq 2C+4$.

Полагая $A = \psi(M_{\min}) = \phi(M_{\max})$, нетрудно найти

$$M_{\max} - M_{\min} = 1 + \lg_{|\rho|} \frac{\psi(0)}{\phi(1)}.$$

Из этой формулы следует, что разность $M_{\max} - M_{\min}$ не зависит от величины модуля A . В частности, это означает, что горизонтальные отрезки, заключенные в заштрихованной области рис. 1, имеют постоянную длину. Расчет по этой формуле дает результаты, приведенные в примере 4.

Аналогично можно определить максимальный и минимальный НСЗР кода числа, модуль которого заключен в некоторых пределах. Приведенные формулы дают также возможность определить максимальный модуль числа, при котором НСЗР не превышает заданной величины.

Пусть допустимый максимальный НСЗР равен M_{\lim} - см. рис. 8. *Допустимый максимальный модуль* $A_{\lim \max}$ числа, код которого имеет $\Omega \leq M_{\lim}$, определяется из неравенства:

$$A_{\lim \max} \leq \phi(M_{\lim}). \quad (18)$$

Например, допустимый максимальный модуль правильной ρ -ой дроби равен $\phi(-1)$. Однако это не означает, что правильная ρ -ая дробь не может иметь модуль $\geq \phi(-1)$: допустимый максимальный модуль *гарантирует* соблюдение условия $\Omega \leq M_{\lim}$, но *не ограничивает* величину возможного модуля, при котором это условие соблюдается.

Величину $A_{g.\max}$ удобно применять при выборе масштабных коэффициентов. В этом случае выбор может оказаться намного проще, чем при использовании области G_M .

Наконец, *допустимый минимальный модуль* $A_{g.\max}$ числа, код которого имеет $\Omega \geq M_{\lim}$, определяется из неравенства (см. рис. 8):

$$A_{\lim \min} \geq \psi(M_{\lim}) \quad (19)$$

В частности, допустимый минимальный модуль правильной ρ -ой дроби равен $\psi(-1)$. Здесь также, как и в предыдущем случае, допустимый минимальный модуль *гарантирует* соблюдение условия $\Omega \geq M_{\text{lim}}$, но *не ограничивает* величину возможного модуля, при котором это условие соблюдается.

Из (18) и (19) следует, что соблюдение условия $M' \leq \Omega \leq M''$ для кода некоторого числа Z гарантируется в том случае, если

$$\psi(M') \leq |Z| \leq \phi(M''). \quad (20)$$

2.3. Погрешности конечных кодов

В общем случае для точного представления числа Z необходимо использовать бесконечный код

$$K(Z) = \alpha_M \dots \alpha_{N+1} \alpha_N \dots \quad (21)$$

Однако практически всегда достаточно иметь приближенное значение \tilde{Z} числа Z , отличающееся от него на величину погрешности Δ :

$$\tilde{Z} = Z - \Delta. \quad (22)$$

Пусть \tilde{Z} имеет конечный код

$$K(\tilde{Z}) = \alpha_M \dots \alpha_{N+1}; \quad (23)$$

тогда

$$K(\Delta) = \alpha_N \dots \quad (24)$$

По определению, *абсолютная погрешность* представления числа Z приближенным значением \tilde{Z} равна модулю погрешности Δ : $|\Delta| = |Z - \tilde{Z}|$. Максимальная абсолютная погрешность $\varepsilon = |\Delta|_{\max}$. Следовательно,

$$\varepsilon = \psi(N). \quad (25)$$

Область изменения числа, представленного конечным кодом (23), отличается от области изменения числа, представленного бесконечным кодом (21) с тем же номером старшего разряда, на область изменения числа Δ , представленного кодом (24). Заметим, что область изменения числа, представленного конечным кодом, следует отличать от диапазона представимых чисел, т.е. от количества различных конечных кодов с данным числом разрядов. Понятия области и диапазона совпадают для целых кодов по основанию $\rho = R$ с нулевым номером младшего разряда.

Относительная погрешность представления числа Z конечным кодом (23) по определению равна отношению $|\Delta| : |Z_M|$, где Z_M - число, представленное нормализованным кодом (21). Это отношение принимает максимальное значение при $|\Delta| = |\Delta|_{\max}$ и $|Z_M| = |Z_M|_{\min}$. Следовательно, максимальная относительная погрешность

$$\delta = \frac{\psi(N)}{\phi(M)}. \quad (26)$$

По этой формуле подсчитывается максимальная относительная погрешность представления числа Z_M нормализованным конечным кодом (23). Для решения обратной задачи, т.е. для определения *длины* (числа разрядов) конечного кода, обеспечивающей необходимую максимальную относительную погрешность, необходимо из формулы (26) выразить длину кода через δ . Совмещая формул (12), (14) и (26), получаем:

$$\delta = \frac{|\rho|^N \cdot \psi(0)}{|\rho|^{M-1} \cdot \phi(1)}. \quad (27)$$

Длина кода

$$l = M - N + 1. \quad (28)$$

Подставляя (28) в (27), находим:

$$\delta = \gamma \cdot |\rho|^{-l} \quad (29)$$

или

$$l = \lg_{|\rho|} \frac{\gamma}{\delta}, \quad (30)$$

где

$$\gamma = \frac{|\rho|^2 \cdot \psi(0)}{\phi(1)}. \quad (31)$$

Пример 5.

Определим коэффициенты γ для различных оснований, пользуясь результатами примеров 1 и 2.

- при $\rho = R$: $\gamma = R^2$; в частности, при $\rho = 2$: $\gamma = 4$.
- при $\rho = -R$: $\gamma = R^3$; в частности, при $\rho = -2$: $\gamma = 8$.
- при $\rho = i\sqrt{R}$: $\gamma = R^2 \sqrt{R+1}$; в частности, при $\rho = i\sqrt{2}$: $\gamma = 6,9$.
- при $\rho = i-1$: $\gamma = 11$.

При $|\rho| = R$ имеем:

$$l = \lg_R \gamma - \lg_R \delta. \quad (32)$$

При $|\rho| = \sqrt{R}$ получаем:

$$l = 2[\lg_R \gamma - \lg_R \delta]. \quad (33)$$

Полученные формулы позволяют вычислить длину кода по известной величине δ . Сравнение формул (32) и (33) показывает, что при фиксированном δ длина кода по комплексному основанию вдвое больше длины кода по действительному основанию. Это утверждение не совсем точно, если учесть различие в величинах $\lg_R \gamma$. Однако это различие не превышает единицы (см. пример 5) и ей можно пренебречь. Этого результата и следовало ожидать, поскольку код комплексного числа несет информацию одновременно о действительной и мнимой частях этого числа.

Пример 6.

Пусть $\delta = 2^{-b}$, где b - положительное число. Определим длину двоичных кодов по данному δ для различных оснований.

$$\text{При } \rho = 2: \quad l_1 = b + 2.$$

$$\text{При } \rho = -2: \quad l_2 = b + 3.$$

$$\text{При } \rho = i\sqrt{2}: \quad l_3 = 2b + 6.$$

$$\text{При } \rho = i - 1: \quad l_4 = 2b + 7.$$

Следует заметить, что действительная длина двоичного кода по основанию $\rho = 2$ за счет добавления знакового разряда равна $l_1^I = l_1 + 1$. Таким образом, $l_1^I = l_2$ и $l_3 = 2l_2$. Следовательно, применение кодов комплексных чисел удваивает разрядность регистров.

2.4. Коды с плавающей точкой

Формула (29) выражает максимальную относительную погрешность числа, представленного нормализованным кодом длины l . Очевидно, если старшие разряды $\alpha_M, \alpha_{M-1}, \alpha_{M-2}, \dots$ принимают нулевые значения, то длина кода уменьшается и максимальная относительная погрешность возрастает. Для того, чтобы сохранять длину кода, применяют коды с плавающей точкой. Этот прием можно распространить и на коды комплексных чисел.

Любое комплексное число Z может быть представлено в следующей форме:

$$Z = T \cdot \rho^\pi, \quad (34)$$

где

π - целое действительное число, *экспонента* (порядок) числа Z ;

T - комплексное число, правильная ρ -ая дробь; *мантисса* числа Z .

Итак,

$$K_\rho(T) = 0, \alpha_{-1} \alpha_{-2} \dots \alpha_k \dots \alpha_{-n}. \quad (35)$$

Пусть

$$K_{\rho_1}(\pi) = \beta_{Q-1} \dots \beta_q \dots \beta_0. \quad (36)$$

это Q -разрядный код числа π по действительному основанию $\rho_1 = \pm R$, представляющий R^Q различных значений числа π , заключенного в пределах: $\pi_{\min} \leq \pi \leq \pi_{\max}$, причем

$$\pi_{\max} - \pi_{\min} + 1 = R^Q. \quad (37)$$

Кодом с плавающей точкой числа Z называется выражение

$$\tilde{K}(Z) = \beta_{Q-1} \dots \beta_q \dots \beta_0; \alpha_{-1} \alpha_{-2} \dots \alpha_k \dots \alpha_{-n}. \quad (38)$$

Код $\tilde{K}(Z)$ называется *нормализованным*, если $\alpha_{-1} \neq 0$. Максимальная относительная погрешность нормализованного кода с плавающей запятой определяется согласно (29):

$$\delta = \gamma \cdot |\rho|^{-n}. \quad (39)$$

Учитывая (18), (19) и (34), находим, что допустимый максимальный модуль числа Z , представленного нормализованным кодом $\tilde{K}(Z)$,

$$|Z|_{\lim \max} = |\rho|^{\pi_{\max}} \cdot \phi(-1), \quad (40)$$

а допустимый минимальный модуль этого же числа

$$|Z|_{\lim \min} = |\rho|^{\pi_{\min}} \cdot \psi(-1). \quad (41)$$

Отношение

$$\lambda = \frac{|Z|_{\lim \max}}{|Z|_{\lim \min}} \quad (42)$$

характеризует область изменения модуля числа Z , представленного нормализованным кодом $\tilde{K}(Z)$ с максимальной относительной погрешностью δ . Из (13), (15), (31), (37), (40), (41), (42) находим:

$$\lambda = \frac{1}{\gamma} \cdot |\rho|^{R^Q}. \quad (43)$$

Определим для сравнения коэффициент λ^I для обычного кода, имеющего ту же разрядность, что и код $\tilde{K}(Z)$. Пусть

$$K(Z^I) = \alpha_{Q-1} \dots \alpha_q \dots \alpha_0, \alpha_{-1} \alpha_{-2} \dots \alpha_{-n}. \quad (44)$$

Для того, чтобы максимальная относительная погрешность этого кода определялась формулой (39), необходимо, чтобы его НСЗР находился в пределах: $-1 \leq \text{НСЗР} \leq Q-1$. Тогда согласно (20)

находим $\psi(-1) \leq |Z'| \leq \phi(Q-1)$, откуда получаем $\lambda' = \frac{\phi(Q-1)}{\psi(-1)}$ или, согласно (13), (14), (31),

$$\lambda' = \frac{1}{\gamma} \cdot |\rho|^{Q+1}. \quad (45)$$

Очевдно, коэффициент λ' не зависит от положения точки в коде (44).

Отношение коэффициентов λ и λ' характеризует эффективность применения кодов с плавающей точкой вместо

кодов с фиксированной точкой при заданной максимальной относительной погрешности:

$$\lambda : \lambda' = |\rho|^{R^Q - Q - 1}. \quad (46)$$

Пример 7

Определим эффективность применения плавающей точки для двоичных 40-разрядных кодов при $\rho = i\sqrt{2}$ и $\rho_1 = -2$. Здесь $\gamma = 6,9$ - см. пример 5.

1. Пусть $\delta = 2^{-15}$. Из примера 6 находим $n = 36$. Далее имеем:

$$Q = 4; \quad \lambda = 0,15 \cdot 2^{2^3}; \quad \lambda' = 0,15 \cdot 2^{2^{2,5}}. \quad \text{Следовательно, отношение } \lambda : \lambda' = 2^{5,5}.$$

2. Пусть $\delta = 2^{-14}$. Тогда $n = 34$; $Q = 6$; $\lambda = 0,15 \cdot 2^{2^5}$; $\lambda' = 0,15 \cdot 2^{3,5}$ и $\lambda : \lambda' = 2^{28,5}$.

3. Пусть $\delta = 2^{-13}$. Тогда $n = 32$; $Q = 8$; $\lambda = 0,15 \cdot 2^{2^7}$; $\lambda' = 0,15 \cdot 2^{4,5}$ и $\lambda : \lambda' = 2^{123,5}$.

4. Пусть $\delta = 2^{-12}$. Тогда $n = 30$; $Q = 10$; $\lambda = 0,15 \cdot 2^{2^9}$; $\lambda' = 0,15 \cdot 2^{5,5}$ и $\lambda : \lambda' = 2^{506,5}$.

Таким образом, эффективность применения плавающей точки резко возрастает с уменьшением допустимой относительной погрешности.

Пример 8. Точность кодирования по основанию $\rho = j\sqrt{2}$.

Рассмотрим пределы изменения нормализованного числа с 10-разрядной экспонентой. Пределы изменения экспоненты и мантиссы таковы:

- экспонента (степень числа «-2») – от (-682) до (341)
- величина положительной экспоненты – $\left(4^{-341}\right) \div \left(4^{170}\right)$,
- величина отрицательной экспоненты – $\left(-2 \cdot 4^{-340}\right) \div \left(-2 \cdot 4^{170}\right)$,
- действительная часть мантиссы – от $\left(-\frac{2}{3}\right) \approx (-0.6667)$ до $\left(\frac{1}{3}\right) \approx (0.3333)$,
- мнимая часть мантиссы – от $\left(-\frac{2\sqrt{2}}{3}\right) \approx (-0.9428)$ до $\left(\frac{\sqrt{2}}{3}\right) \approx (0.4714)$.

Пример 9. Точность кодирования по основанию ρ_2 .

Рассмотрим пределы изменения мантиссы комплексного числа при кодирования по основанию ρ_2 . Код нормализованной мантиссы X имеет вид $0.xu***\dots\dots$, где по крайней мере один из разрядов x, u имеет значение «1». Величина соответствующей части мантиссы (x относится к Im , u относится к Re) находится в пределах $-\frac{2}{3} < \bullet < -\frac{1}{6}$. Величина другой (а, следовательно,

любой) части мантиссы находится в пределах $-\frac{2}{3} < \bullet < \frac{1}{3}$.

Пределы изменения мантиссы комплексного числа таковы:

- действительная или мнимая нормализованная часть мантиссы
– от $\left(-\frac{2}{3}\right) \approx (-0.6667)$ до $\left(-\frac{1}{6}\right) \approx (-0.16667)$,
- действительная или мнимая ненормализованная часть
мантиссы – от $\left(-\frac{2}{3}\right) \approx (-0.6667)$ до $\left(\frac{1}{3}\right) \approx (0.3333)$.

Глава 3. Поразрядные арифметические операции

3.1. Поразрядные операции

Пусть Z_1 и Z_2 - некоторые векторы,

$$Z = Z_1 \Delta Z_2 \quad (1)$$

- результат операции Δ с ними, а позиционные коды этих векторов в арифметической системе $\langle \rho, A_R \rangle$ имеют вид $K(Z_1) = \dots \alpha_k \dots$, $K(Z_2) = \dots \beta_k \dots$, $K(Z) = \dots \sigma_k \dots$

Рассмотрим класс поразрядных операций, то есть операций сложения, вычитания, умножения на постоянный коэффициент и, в том числе, инвертирования, то есть умножения на -1 . Поразрядная Δ операция производится последовательно над каждой парой чисел α_k и β_k (k -ых разрядов исходных кодов) с учетом переноса из младших разрядов по формуле

$$S_k = Q_k + P_k, \quad (2)$$

где

$Q_k = \alpha_k \Delta \beta_k$ - результат операции Δ над k -ми разрядами,

P_k - перенос из младших разрядов в k -ый разряд,

S_k - разрядный результат.

Разрядный результат всегда можно представить в виде

$$S_k = \sigma_k + \rho P_{k+1}, \quad (3)$$

где

σ_k - k -ый разряд результирующего кода,

P_{k+1} - перенос из k -го разряда в $(k+1)$ -ый разряд.

Очевидно, для того, чтобы не возникало переноса из старших разрядов в младшие, значение P_{k+1} должно быть ρ -целым при любом k . В частности, P_k должно быть ρ -целым. Отсюда следует, что значения S_k и Q_k также должны быть ρ -целыми. Для

соблюдения всех этих условий необходимо и достаточно, чтобы лишь значение Q_k было ρ -целым.

Таким образом, операция с позиционными кодами является поразрядной, если выполняются два условия:

- $(\alpha_k \Delta \beta_k) \rho^k = \alpha_k \rho^k \Delta \beta_k \rho^k$,
- Q_k - ρ -целое.

Из определения 1.1 следует, что в арифметической системе операции инвертирования, сложения и умножения на число из множества A_R являются поразрядными операциями. Следовательно, операции алгебраического сложения и умножения на ρ -целый множитель в арифметической системе являются поразрядными операциями.

Метод выполнения поразрядной операции и сложность ее реализации зависит, главным образом, от метода и сложности вычисления значений переноса из младших разрядов в старшие. Например, при большой разрядности ρ -целого множителя задача вычисления переносов практически неразрешима. При алгебраическом сложении сложность вычисления переносов зависит, в основном, от величины R и числа разрядов в кодах чисел ' R ' и ' $-R$ '.

Существует два способа определения значений переноса и поэтому поразрядная операция может выполняться по двум различным алгоритмам.

3.2. Первый алгоритм поразрядных операций

Пусть $\langle S_k \rangle = \gamma_m \dots \gamma_j \dots \gamma_1 \gamma_0$, где γ_j - цифры из множества A_R . Тогда, согласно (3),

$$\sigma_k = \gamma_0, \quad (4)$$

$$\langle P_{k+1} \rangle = \mu_{m-1} \dots \mu_j \dots \mu_0, \quad (5)$$

где $\mu_{j-1} = \gamma_j$. С использованием полученных соотношений составляем последовательность элементарных операций для вычислений в k -ом разряде:

Алгоритм 1. Известны α_k, β_k, P_k .

1. Определяется $Q_k = \alpha_k \Delta \beta_k$.
2. Определяется S_k по (2).
3. Определяется код $\langle S_k \rangle$.
4. Определяется σ_k по (4).
5. Определяется код $\langle P_{k+1} \rangle$ вида (5).
6. Определяется P_{k+1} по коду $\langle P_{k+1} \rangle$.
7. Выполняются операции 1-6 для $(k+1)$ -го разряда.

Рис. 1 иллюстрирует схему вычислений в k -разряде, а рис. 2 иллюстрирует схему распространения переносов по алгоритму 1.

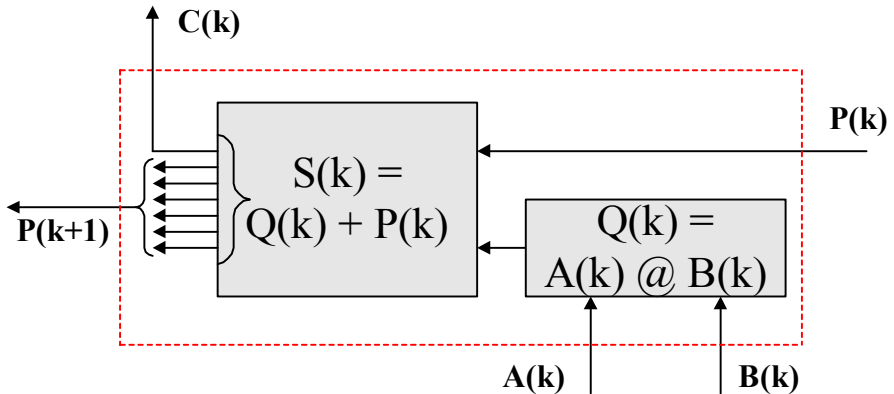


Рис. 1. Схема вычислений в k -разряде (к алгоритму 1)

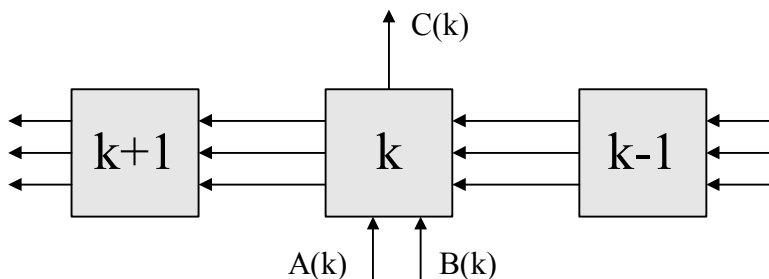


Рис. 2. Схема распространения переносов (к алгоритму 1)

Полученная форма записи алгоритма не дает наглядного представления о его сложности и особенностях и неудобна для практического применения. Поэтому она используется только для получения другой формы записи алгоритма - построения таблицы, описывающей поразрядную операцию в одном разряде. Эта таблица является законченной только в том случае, если удовлетворяет двум следующим условиям:

- если перенос P_{k+1} принимает некоторое значение, то и перенос P_k принимает такое же значение;
- в таблице присутствует любая возможная комбинация значений α_k , β_k , P_k .

Эти условия в дальнейшем будем называть **условиями полноты** таблицы поразрядных операций. Рассмотрим некоторые примеры синтеза таблиц поразрядных операций по алгоритму 1 и вычислений с помощью полученных таблиц.

Пример 1. Инвертирование кодов в системе $\langle \rho = -1 \pm j; \{0, 1\} \rangle$ (см. теорему 1.4). Имеется код $K(Z) = \dots \alpha_k \dots$. Необходимо определить код $K(-Z) = \dots \sigma_k \dots$

Очевидно, $Q_k = \alpha_k$. Применяя алгоритм 1, строим таблицу 1, удовлетворяющую условиям полноты: переносы P_k и P_{k+1} принимают значения из общего пятиэлементного множества $P = \{0, 1, j, -1, 1+j\}$ и в таблице присутствуют все возможные пары чисел, одно из которых принадлежит множеству $\{0, 1\}$, а другое - множеству P .

Таблица 1. Инвертирование кодов.

α_k	P_k	Q_k	S_k	$\langle P_{k+1} \rangle$	σ_k	P_{k+1}
0	0	0	0	0	0	0
1	0	-1	-1	1110	1	1+j
0	1+j	0	1+j	111	0	-j
1	1+j	-1	j	1	1	1
0	-j	0	-j	11	1	j
1	-j	-1	-1-j	11	0	j
0	j	0	j	1	1	1
1	j	-1	-1+j	1	0	1
0	1	0	1	0	1	0
1	1	-1	0	1110	1	1+j

Применяя таблицу 1, легко выполнить инвертирование любого кода данной системы. Приведем два примера:

переносы	1	j	-j	1+j	
данный код			1	0	1
результат	1	1	0	0	1

переносы	1	1+j	0	0	1	j	-j	1+j	
данный код	1	1	1	0	0	1	0	0	1
результат		1	1	0	1	0	1	0	1

Пример 2. Сложение кодов в системе $\langle \rho = \sqrt{2}e^{j\varphi}, \{0, 1\} \rangle$, где $\cos \varphi = -1/2\sqrt{2}$ (см. теорему 1.5). В этом случае $Q_k = \alpha_k + \beta_k$ принимает три значения: $\{0, 1, 2\}$. Применяя алгоритм 1, строим таблицу 2, описывающую процесс сложения. В ней для сокращения объема

- не записаны комбинации разрядов α_k и β_k , дающие в сумме Q_k ;
- величина S_k не приводится, а записывается только код $\langle S_k \rangle$;
- код $\langle S_k \rangle$ записывается в клетке, находящейся на пересечении столбца Q_k и строки P_k (например, при $Q_k=1$ и $P_k = \rho$ код $\langle S_k=1+\rho \rangle=101$).

Эта таблица удовлетворяет условиям полноты, так как в ней присутствуют все возможные комбинации значений Q_k и P_k , а код $\langle P_k \rangle$ принимает те же значения, что и код $\langle P_{k+1} \rangle$, состоящий из старших разрядов кода $\langle S_k \rangle$ (например, при $\langle S_k \rangle = 1110$ код $\langle P_{k+1} \rangle = 111$, но код $\langle P_k \rangle = 111$ также присутствует в таблице).

Таблица 2. Сложение кодов.

P_k	$\langle P_k \rangle$	код $\langle S_k \rangle$ при		
		$Q_k = 0$	$Q_k = 1$	$Q_k = 2$
0	0	0	1	1010
1	1	1	1010	1011
ρ	10	10	11	11100
$-\bar{\rho}$	11	11	11100	11101
ρ^2	100	100	101	1110
$\bar{\rho}$	101	101	1110	1111
-1	111	111	0	1
$-\rho$	1110	1110	1111	1000

Приведем два примера использования построенной таблицы 2:

переносы	-1	$\bar{\rho}$	0	-1	$\bar{\rho}$	0	
слагаемое 1	1	0	1	1	1	1	1
слагаемое 2		1	1	0	1	1	0
сумма					1	0	1

переносы				1	$-\bar{\rho}$	-1	$-\rho$	ρ	$\bar{\rho}$	
слагаемое 1			1	0	0	0	0	1	0	1
слагаемое 2		1	0	0	0	0	1	1	0	1
сумма	1	1	1	1	1	1	1	0	1	0

Пример 3. Вычитание кодов в системе $\langle \rho = \sqrt{2}e^{j\varphi}, \{0, 1\} \rangle$, где $\cos \varphi = -1/2\sqrt{2}$ (см. теорему 1.5). В этом случае $Q_k = \alpha_k - \beta_k$ принимает значения: $\{0, 1, -1\}$. Применяя алгоритм 1, строим таблицу 3, описывающую процесс вычитания.

Таблица 3. Вычитание кодов.

P_k	$\langle P_k \rangle$	код $\langle S_k \rangle$ при		
		$Q_k=0$	$Q_k=-1$	$Q_k=1$
0	0	0	111	1
1	1	1	0	1010
ρ	10	10	111001	11
$-\bar{\rho}$	11	11	10	11100
$-\rho^2$	11100	11100	11	11101
$\bar{\rho}$	101	101	100	1110
-1	111	111	110	0
$-\rho$	1110	1110	101	1111

Пример 4. Инвертирование кодов в системе $\langle \rho = \sqrt{2}e^{j\varphi}, \{0, 1\} \rangle$, где $\cos \varphi = -1/2\sqrt{2}$ (см. теорему 1.5). В этом случае $Q_k = \alpha_k - \beta_k$ принимает значения: $\{0, -1, -2\}$. Применяя алгоритм 1, строим таблицу 4, описывающую процесс инвертирования. В ней (также, как и в таблице 2) для сокращения объема

- величина S_k не приводится, а записывается только код $\langle S_k \rangle$;
- код $\langle S_k \rangle$ записывается в клетке, находящейся на пересечении столбца Q_k и строки P_k .

Эта таблица удовлетворяет условиям полноты, так как в ней присутствуют все возможные комбинации значений Q_k и P_k , а код $\langle P_k \rangle$ принимает те же значения, что и код $\langle P_{k+1} \rangle$, состоящий из старших разрядов кода $\langle S_k \rangle$.

Таблица 4. Инвертирование кодов.

P_k	$\langle P_k \rangle$	код $\langle S_k \rangle$ при	
		$Q_k=0$	$Q_k=-1$
0	0	0	111
1	1	1	0
$-\bar{\rho}$	11	11	10

Пример 5. Инвертирование кодов в системе $\langle \rho = 2e^{j\pi/3}, A_4 = \{0, 1, c, d\} \rangle$, где $c = e^{2j\pi/3}$, $d = e^{-2j\pi/3}$ (см. теорему 1.6).

Таблица 5. Инвертирование кодов.

α_k	Q_k	(P_{k+1}, σ_k) при P_k , равном						
		0	1	c	d	-1	-c	-d
0	0	0,0	0,1	0,c	0,d	c,1	d,c	1,d
1	-1	c,b	0,0	-d,d	1,c	c,d	0,d	0,c
c	-c	d,c	-c,d	0,0	-1,1	0,d	d,0	0,1
d	-d	1,d	-c,c	-d,1	0,0	0,c	0,1	1,0

По алгоритму 1 с помощью таблиц 1.3 и 1.4 одноразрядного сложения и инвертирования строим таблицу 5, удовлетворяющую условиям полноты. В этой таблице на пересечении строк Q_k и столбцов P_k записывается пара чисел (P_{k+1}, σ_k) . Применяя таблицу 5, выполним инвертирование некоторых кодов данной системы.

переносы	1	-d	c	0	-1	d	
данный код			d	1	d	1	c
результат	1	d	1	1	c	c	c

переносы	c	-1	d	0	d	0	0	c	
данный код	c	0	1	c	d	c	0	0	1
результат		1	c	c	0	c	0	c	1

Пример 6. Алгебраическое сложение кодов в системах с базовыми функциями $f(\rho, m) = \begin{cases} \rho^{m/2} & \text{if } m - \text{even} \\ j \cdot \rho^{(m-1)/2} & \text{if } m - \text{odd} \end{cases}$, $\rho = -2$ и $f(\rho, m) = \rho^m$, $\rho = \pm j\sqrt{2}$. Заметим, что в этих системах $K(2)=10100$, $K(-2)=100$, $K(-1)=101$. Принимая во внимание метод кодирования в этих системах и вид указанных кодов можно заметить, что алгебраическое сложение в этих системах выполняется по правилам алгебраического сложения в системе кодирования действительных чисел по основанию $\rho = -2$, где $K(2)=110$, $K(-2)=10$, $K(-1)=11$.

3.3. Второй алгоритм поразрядных операций

Будем, в отличие от алгоритма 1, исходить из представления S_k в виде

$$S_k = \rho^m \eta_{k+m} + \dots + \rho^j \eta_{k+j} + \dots + \rho \eta_{k+1} + \sigma_k, \quad (6)$$

где

σ_k - k -ый разряд результирующего кода,

η_{k+j} - **частичный перенос** из k -го разряда в $(k+j)$ -ый,

m - максимальное число частичных переносов.

Очевидно, при такой структуре S_k перенос P_k в k -ый разряд складывается из m частичных переносов μ_{kj} , образующихся в m младших разрядах с номерами от $(k-m)$ до $(k-1)$. Итак,

$$P_k = \sum_{j=1}^m \mu_{kj} \quad (7)$$

Для того, чтобы P_k было β -целым, частичные переносы μ_{kj} также должны быть β -целыми. Других ограничений на числа μ_{kj} и η_{k+j} не накладывается, поэтому они в общем случае могут отличаться от чисел множества A_R .

Назовем выражение вида

$$\langle\langle S_k \rangle\rangle = \eta_{k+m} \dots \eta_{k+j} \dots \eta_{k+1} \sigma_k$$

квазикодом разрядного результата, а выражение вида

$$\langle\langle P_k \rangle\rangle = \eta_{k+m} \dots \eta_{k+j} \dots \eta_{k+1}$$

- **квазикодом** переноса из k -го разряда.

Используя полученные соотношения, составляем последовательность элементарных операций для вычислений в k -ом разряде:

Алгоритм 2. Известны α_k , β_k , μ_{kj} .

1. Определяется $Q_k = \alpha_k \Delta \beta_k$.
2. Определяется P_k по (7).

3. Определяется S_k по (2).
4. Определяется квазикод $\ll S_k \gg$.
5. Определяется σ_k из квазикода $\ll S_k \gg$.
6. Определяется квазикод $\ll P_{k+1} \gg$, то-есть частичные переносы из k -го разряда.
7. Выполняются операции 1-6 для $(k+1)$ -го разряда.

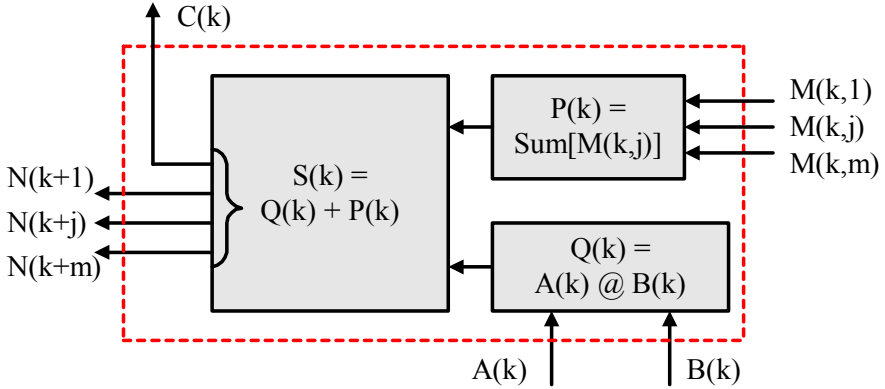


Рис. 4. Схема вычислений в k -разряде (к алгоритму 2)

Рис. 4 иллюстрирует схему вычислений в k -разряде, а рис. 5 иллюстрирует схему распространения переносов по алгоритму 2. На этих рисунках обозначено: $N(k+j) = \eta_{kj}$, $M(k,j) = \mu_{kj}$.

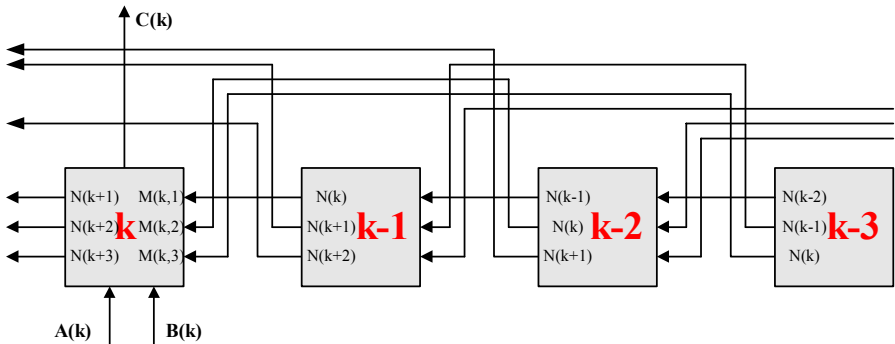


Рис. 5. Схема распространения переносов (к алгоритму 2)

Алгоритм 2 для данной поразрядной операции в данной системе кодирования может иметь несколько вариантов в связи с тем, что существует несколько квазикодов для разрядного результата S_k . В частности, квазикод может совпадать с позиционным кодом, если

значения частичных переносов выбраны из множества A_R . В общем случае различные разряды квазикода принимают значения из множеств, отличающихся от A_R не только значениями элементов, но и мощностью (количеством элементов). Например, часть разрядов квазикодов может иметь всегда нулевое значение (определяться на пустом множестве), часть разрядов - принимать значения из множества A_R , часть разрядов - из множества $A' \neq A_R$.

Последовательность операций алгоритма 2 используется для синтеза таблиц поразрядных операций, удовлетворяющих условиям полноты, которые в данном случае формулируются несколько иначе:

- перенос P_k принимает все возможные значения, соответствующие любой комбинации η_{k+j} ,
- в таблице присутствует любая возможная комбинация чисел α_k, β_k, P_k .

Очевидно, согласно алгоритму 2, имеющему несколько вариантов, может быть построено несколько типов таблиц для данной поразрядной операции в данной системе кодирования. Рассмотрим некоторые примеры синтеза таблиц поразрядных операций согласно алгоритму 2 и вычислений с помощью полученных таблиц.

Пример 7. Инвертирование кодов в системе

$\langle \rho = 2e^{2j\pi/3}, A_4 = \{0,1,2,3\} \rangle$ (см. теорему 1.5). В данном случае $Q_k = -\alpha_k$ принимает четыре значения (0, -1, -2, -3), так как α_k принимает четыре значения (0, 1, 2, 3). Используя алгоритм 2, строим таблицу 7, в которой значения S_k и их квазикоды записываются на пересечении строк Q_k и столбцов P_k в виде пары $(S_k, \langle\langle S_k \rangle\rangle)$. В этой таблице квазикоды значений S_k совпадают с позиционными кодами тех же значений и основаны на их представлении в виде $S_k = \rho^2 \eta_{k+2} + \rho \eta_{k+1} + \sigma_k$.

Таблица 7. Инвертирование кодов

σ_k	Q_k	$(S_k, \langle S_k \rangle)$ при P_k , равном			
		0	1	2	3
0	0	0, 0	1, 1	2, 2	3, 3
1	-1	-1, 123	0, 0	1, 1	2, 2
2	-2	-2, 122	-1, 123	0, 0	1, 1
3	-3	-3, 121	-2, 122	-1, 123	0, 0

Как следует из таблицы 7, частичные переносы принимают значения из множества $\{0, 1, 2\}$. В связи с этим перенос в данный разряд может принимать четыре различных значения, соответствующие всем возможным комбинациям из двух частичных переносов в данный разряд. Таким образом, $P_k \in \{0, 1, 2, 3\}$. Все эти значения P_k , так же как и все значения Q_k , присутствуют в таблице. Следовательно, она удовлетворяет условиям полноты. Ниже приведен пример вычислений с помощью таблицы 7:

переносы	1	2	0			
		1	2	0		
			0	0	0	
				1	2	0
данный код	0	3	1	3	2	2
результат	1	0	3	2	0	2

Пример 8. Сложение кодов в системе $\langle \rho = j-1; \{0, 1\} \rangle$ (см. теорему 1.4). Для описания этой операции построим, согласно алгоритму 2, сокращенную таблицу 8, содержащую только величину S_k и квазикод $\langle\langle S_k \rangle\rangle$, все разряды которого принимают значения 0 или 1. В этом случае квазикод совпадает с позиционным кодом, но два его разряда всегда имеют нулевое значение. Следовательно, в k -ом разряде вырабатывается не более шести частичных переносов, равных 1. Это означает, что и в k -ый разряд может поступить также не более шести частичных переносов, то-есть $0 \leq P_k \leq 6$. Так как $0 \leq Q_k \leq 2$ при сложении данных кодов, то $0 \leq S_k \leq 8$. Все 8 значений S_k присутствуют в таблице 8, следовательно, она удовлетворяет условиям полноты.

Таблица 8. Сложение кодов

S_k	$\langle S_k \rangle$
0	000000000
1	000000001
2	000001100
3	000001101
4	111010000
5	111010001
6	111011100
7	111011101
8	111000000

Для иллюстрации существования нескольких вариантов алгоритма 2 построим еще две таблицы 8а и 8в, описывающие сложение в данной системе. В этих таблицах цифрой *a* обозначена величина ‘-1’.

Таблица 8а. Сложение кодов

S_k	$\langle S_k \rangle$
-1	11101
0	00000
1	00001
2	01100
3	01101
4	a0000
5	a000

Таблица 8в. Сложение кодов

S_k	$\langle S_k \rangle$
-3	10001
-2	0aa00
-1	0aa01
0	00000
1	00001
2	aaa00
3	aaa01

Как следует из таблицы 8а, в k -ый разряд может поступать одновременно три частичных переноса (частичный перенос, всегда равный нулю, не рассматривается):

$$0 \leq \mu_{k1} \leq 1, 0 \leq \mu_{k2} \leq 1, -1 \leq \mu_{k3} \leq 1,$$

следовательно, $-1 \leq P_k \leq 3$, откуда $-1 \leq S_k \leq 5$. Таким образом, таблица 8а является полной. Аналогично доказывается полнота таблицы 8в. Выбор того или иного варианта алгоритма 2 определяется минимальным числом значений, которые принимает перенос P_k . С этой точки зрения таблица 8 уступает последним таблицам 8а и 8в. Рассмотрим пример сложения по таблице 8в:

квазикоды переносов	1	0	0	0						
			а	а	0					
				а	а	0				
					а	а	0			
					а	а	а	0		
						а	а	а	0	
слагаемое 1			1	1	0	1	0	0	1	1
слагаемое 2		1	1	1	0	0	1	0	1	1
сумма	1	1	1	0	1	0	1	1	0	0

Пример 9. Обратное сложение кодов в системе $\langle \rho = j-1; \{0, 1\} \rangle$ (см. теорему 1.4).

Таблица 9. Обратное сложение кодов.

S_k	$\langle\langle S_k \rangle\rangle$
-2	11100
-1	11101
0	00000
1	00001
2	01100
3	01101

Аналогично предыдущему примеру построим сокращенную таблицу 9, содержащую только величины S_k и квазикод $\langle\langle S_k \rangle\rangle$, все разряды которого принимают значения 0 или 1. В этом случае квазикод совпадает с позиционным кодом, но один

его разряд всегда имеет нулевое значение. Следовательно, в k -ом разряде вырабатывается не более трех частичных переносов, равных 1. Это означает, что и в k -ый разряд может поступить не более трех частичных переносов, то-есть $0 \leq P_k \leq 3$. Так как $-2 \leq Q_k \leq 0$ при обратном сложении данных кодов, то $-2 \leq S_k \leq 3$. Все 6 значений S_k присутствуют в таблице 9, следовательно, она удовлетворяет условиям полноты. В этой таблице выделена часть, относящаяся к инвертированию кодов в этой системе.

На основе таблицы 9 построена таблица 9а истинности одноразрядных схем для и обратного сложения кодов в данной системе. В ней также выделена часть, являющаяся таблицей истинности для инвертирования. Рис. 6 иллюстрирует схему распространения переносов по одноразрядным схемам в системе кодирования данного примера.

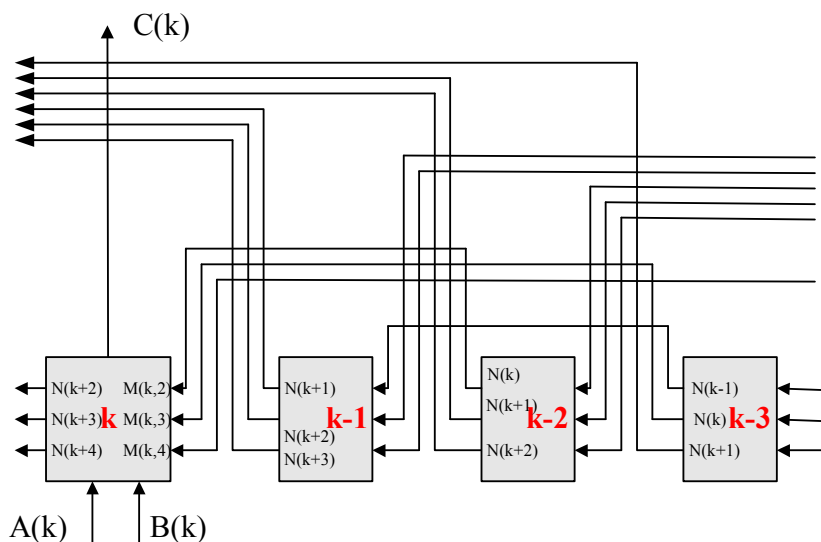


Рис. 6. Схема распространения переносов (к примеру 9)

Таблица 9а истинности.

Input					S_k	<< S_k >>			
β_k	α_k	μ_k				η_{k+j}			σ_k
		4	3	2		4	3	2	
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	1
0	0	0	1	0	1	0	0	0	1
0	0	0	1	1	2	0	1	1	0
0	0	1	0	0	1	0	0	0	1
0	0	1	0	1	2	0	1	1	0
0	0	1	1	0	2	0	1	1	0
0	0	1	1	1	3	0	1	1	1
0	1	0	0	0	-1	1	1	1	1
0	1	0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	0
0	1	0	1	1	1	0	0	0	1
0	1	1	0	0	0	0	0	0	0
0	1	1	0	1	1	0	0	0	1
0	1	1	1	0	1	0	0	0	1
0	1	1	1	1	2	0	1	1	0
1	0	0	0	0	-1	1	1	1	1
1	0	0	0	1	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0
1	0	0	1	1	1	0	0	0	1
1	0	1	0	0	0	0	0	0	0
1	0	1	0	1	1	0	0	0	1
1	0	1	1	0	1	0	0	0	1
1	0	1	1	1	2	0	1	1	0
1	1	0	0	0	-2	1	1	1	0
1	1	0	0	1	-1	1	1	1	1
1	1	0	1	0	-1	1	1	1	1
1	1	0	1	1	0	0	0	0	0
1	1	1	0	0	-1	1	1	1	1
1	1	1	0	1	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	1

Пример 10. Обратное сложение кодов в системе $\langle \rho = \sqrt{2}e^{j\varphi}, \{0, 1\} \rangle$, где (см. теорему 1.5).

Таблица 10. Обратное сложение кодов.

S_k	$\langle\langle S_k \rangle\rangle$
-2	0110
-1	0111
0	0000
1	0001
2	1010
3	1011

Аналогично предыдущему примеру построим сокращенную таблицу 10, содержащую только величины S_k и квазикод $\langle\langle S_k \rangle\rangle$, все разряды которого принимают значения 0 или 1. В этом случае квазикод совпадает с позиционным кодом. В k -ом разряде вырабатывается не более трех частичных переносов, равных 1. Это означает, что и в k -ый разряд может поступить также не более трех частичных переносов, то-есть $0 \leq P_k \leq 3$. Так как $-2 \leq Q_k \leq 0$ при обратном сложении данных кодов, то $-2 \leq S_k \leq 3$. Все 6 значений S_k присутствуют в таблице 10, следовательно, она удовлетворяет условиям полноты. В этой таблице выделена часть, относящаяся к инвертированию кодов в этой системе.

На основе таблицы 10 построена таблица 10а истинности одноразрядных схем для обратного сложения кодов в данной системе. В ней также выделена часть, являющаяся таблицей истинности для инвертирования.

Таблица 10а истинности.

Input					S_k	<< S_k >>			
β_k	α_k	μ_k				α_k			μ_k
		3	2	1		3	2	1	
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	1
0	0	0	1	0	1	0	0	0	1
0	0	0	1	1	2	1	0	1	0
0	0	1	0	0	1	0	0	0	1
0	0	1	0	1	2	1	0	1	0
0	0	1	1	0	2	1	0	1	0
0	0	1	1	1	3	1	0	1	1
0	1	0	0	0	-1	0	1	1	1
0	1	0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	0
0	1	0	1	1	1	0	0	0	1
0	1	1	0	0	0	0	0	0	0
0	1	1	0	1	1	0	0	0	1
0	1	1	1	0	1	0	0	0	1
0	1	1	1	1	2	1	0	1	0
1	0	0	0	0	-1	0	1	1	1
1	0	0	0	1	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0
1	0	0	1	1	1	0	0	0	1
1	0	1	0	0	0	0	0	0	0
1	0	1	0	1	1	0	0	0	1
1	0	1	1	0	1	0	0	0	1
1	0	1	1	1	2	1	0	1	0
1	1	0	0	0	-2	0	1	1	0
1	1	0	0	1	-1	0	1	1	1
1	1	0	1	0	-1	0	1	1	1
1	1	0	1	1	0	0	0	0	0
1	1	1	0	0	-1	0	1	1	1
1	1	1	0	1	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	1

Пример 11. Сложение кодов в системе $\langle \rho = -R, A_{R^3} \rangle$ кодирования трехмерных векторов с базой $\mathbf{i}, \mathbf{j}, \mathbf{k}$, где $A_{R^3} = \{0, \mathbf{k}, \mathbf{j}, \mathbf{j} + \mathbf{k}, \mathbf{i}, \mathbf{i} + \mathbf{k}, \mathbf{i} + \mathbf{j}, \mathbf{i} + \mathbf{j} + \mathbf{k}\}$ - см. теорему 1.7. Для обозначения элементов этого множества будем использовать восемь следующих 'цифр': $\mathbf{z} = 0, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}, \mathbf{f}, \mathbf{g}, \mathbf{h}$. Прежде всего заметим, что в этой системе

- $\langle -\mathbf{z} \rangle = \mathbf{zz}$ - инвертирование,
- $\langle \rho \mathbf{z} \rangle = \mathbf{z0}$ - умножение на основание,
- $\langle 2\mathbf{z} \rangle = \mathbf{zz0}$ - удвоение.

Для описания сложения в этой системе построим по алгоритму 2 таблицу 11, содержащую коды слагаемых и суммы.

Таблица 11. Одноразрядное сложение.

+	0	b	c	d	e	f	g	h
0	0	b	c	d	e	f	g	h
b		bb0	d	bbc	f	bbe	h	bbg
c			cc0	ccb	g	h	cce	ccf
d				dd0	h	bbg	ccf	dde
e					ee0	eb	eec	eed
f						ff0	eed	ffc
g							gg0	ggb
h								hh0

Пользуясь этой таблицей, найдем сумму следующих кодов:

квазикоды		b	b			
переносов			e	e		
слагаемое 1				d	f	h
слагаемое 2			c	b	g	0
сумма		b	h	g	d	h

3.4. Поразрядные операции при отрицательном основании

Система кодирования действительных чисел по отрицательному основанию имеет ряд достоинств по сравнению с традиционной системой кодирования по положительному основанию. Эти достоинства являются следствием того, что положительные и отрицательные числа при отрицательном основании представляются единообразно, и заключаются в следующем:

- не требуется выполнять преобразования из прямого кода в обратный (или дополнительный) и обратно,
- исключаются операции со знаковыми разрядами,
- упрощаются правила определения переполнения при алгебраическом сложении,
- упрощаются алгоритмы выполнения операций с кодами переменной длины.

Вместе с тем алгоритмы арифметических операций при отрицательном основании столь же просты, как и при положительном основании. Покажем это, для чего рассмотрим так называемую операцию **обратного сложения**, выполняемую по формуле $a = -b - c$. В этой операции

$$S_k = -\alpha_k - \beta_k + P_k, \quad S_k = \sigma_k - R \cdot P_{k+1}.$$

Следовательно, $P_k \in \{0, 1\}$, то есть оба алгоритма поразрядных операций сливаются в единственный простейший алгоритм операции с двоичным переносом.

Очевидно, этот алгоритм по сложности эквивалентен алгоритму обычного сложения кодов действительных чисел по положительному основанию. В связи с этим обратное сложение следует использовать как базовую операцию алгебраического сложения при отрицательном основании, а обычное сложение, как более сложную операцию, сводить к трем операциям обратного сложения:

$$a = b + c = -(-b - 0) - (-c - 0).$$

Практически операции инвертирования (являющиеся частным случаем обратного сложения) могут выполняться параллельно с основной операцией, что существенно уменьшает время сложения при любых знаках слагаемых и знаках, стоящих перед слагаемыми в формуле алгебраического сложения.

Как указывалось, два способа кодирования комплексных чисел и векторов состоят в построении некоторых композиций из кодов действительных чисел по отрицательному основанию - **композиционных кодов**. При выполнении поразрядных операций такие коды векторов удобно рассматривать как состоящие из // независимых частей - кодов чисел-проекций векторов и выполнять указанные операции с каждой из этих частей параллельно и независимо по известным алгоритмам. Все сказанное выше в данном разделе при этом сохраняет силу, что является определенным достоинством композиционных кодов векторов и комплексных чисел.

Кроме того, эти коды позволяют выполнять разноименные поразрядные операции с каждой проекцией вектора. Например, сложение и действительной и мнимой составляющих комплексных чисел может производиться независимо. В частности, так может быть выполнено инвертирование мнимой составляющей комплексного числа, в результате чего будет получено сопряженное ему число.

Итак, в композиционных системах алгебраическое сложение проекций вектора производится независимо по правилам операций в системе $\langle -R, B_R \rangle$ и только при умножении и делении явно учитывается 'векторность' основания или 'векторность' разрядов кода.

Поразрядные операции с кодами действительных чисел по основанию (-2) подробно описаны в [45]. Поэтому здесь эти операции не рассматриваются.

Глава 4. Алгоритмы кодирования и декодирования комплексных чисел

Кодируемое комплексное число представляется в виде $Z = X_\alpha + jX_\beta$, где X_α , X_β - действительная и мнимая части комплексного числ, являющиеся действительными (положительными или отрицательными) числами. Для этих чисел существуют разложения и позиционные коды по основанию $\rho = -2$. В перечисленных в табл. 1.5 системах кодирования мнимая единица может быть представлена в следующем виде: $j = \mu \cdot \omega$, где μ - действительное число, ω - комплексное число, имеющее малоразрядный код в данной системе. В табл. 1 приведены эти числа и коды для всех вышеуказанных систем кодирования.

Таблица 1.

Система кодирования		μ	ω	$K(\omega)$
№	Базовая функция и основание			
1	$f(\rho, m) = \begin{cases} \rho^{m/2} & \text{if } m - \text{even} \\ j \cdot \rho^{m-1/2} & \text{if } m - \text{odd} \end{cases},$ $\rho = -2$	1	j	10
2.1	$f(\rho, m) = \rho^m, \rho = j\sqrt{2}$	$1/\sqrt{2}$	$j\sqrt{2}$	10
2.2	$f(\rho, m) = \rho^m, \rho = -j\sqrt{2}$	$-1/\sqrt{2}$	$-j\sqrt{2}$	10
3.1	$f(\rho, m) = \rho^m, \rho = (-1 + j)$	1	j	11
3.2	$f(\rho, m) = \rho^m, \rho = (-1 - j)$	-1	$-j$	11
4	$f(\rho, m) = \rho^m, \rho = \frac{1}{2}(-1 + j\sqrt{7})$	$1/\sqrt{7}$	$j\sqrt{7}$	10101

Поэтому кодируемое комплексное число представляется в виде

$$Z = X_\alpha + \omega \cdot \bar{X}_\beta, \text{ where } \bar{X}_\beta = \mu \cdot X_\beta.$$

При этом алгоритм кодирования в общем случае имеет следующий вид:

Алгоритм 1.

1. Вычисление $\bar{X}_\beta = \mu \cdot X_\beta$ в традиционной системе двоичного кодирования.
2. Представление действительных (положительных или отрицательных) чисел X_α , \bar{X}_β в системе кодирования по основанию $\rho = -2$ в виде $X = \sum_{(m)} \alpha_m (-2)^m$, $\alpha_m = \{0,1\}$ (этот алгоритм известен и здесь не описывается).
3. Кодирование действительных чисел X_α , \bar{X}_β в данную систему кодирования комплексных чисел, путем вычисления по предыдущей формуле, где коды чисел $(-2)^m$ определены заранее (например, путем последовательного умножения на (-2)).
4. Вычисление по формуле $Z = X_\alpha + \omega \cdot \bar{X}_\beta$ в данной системе кодирования комплексных чисел, где код $K(\omega)$ известен – см. предыдущую таблицу.

Алгоритм декодирования в общем случае имеет следующий вид:

Алгоритм 2.

1. Выделение комплексных кодов действительной и мнимой частей X_α , X_β из комплексного числа $Z = X_\alpha + jX_\beta$.
2. Декодирование действительных (положительных или отрицательных) чисел X_α , X_β в систему кодирования по основанию $\rho = -2$, а именно, вычисление разрядов $\alpha_m = \{0,1\}$ для разложения вида $X = \sum_{(m)} \alpha_m (-2)^m$. Это вычисление заключается в последовательном вычитании чисел $(-2)^m$ из декодируемого числа. При этом, если очередное вычитание уменьшает модуль остатка, то $\alpha_m = 1$. В противном случае $\alpha_m = 0$, степень m уменьшается на 1 и т.д.

3. Декодирование действительных (положительных или отрицательных) чисел X_α , X_β из системы кодирования по основанию $\rho = -2$ в традиционную систему двоичного кодирования.

Для некоторых систем эти общие алгоритмы существенно упрощаются. Рассмотрим их.

Алгоритм 3. Кодирование в системе 1.

1. Представление действительных (положительных или отрицательных) чисел X_α , X_β в системе кодирования по основанию $\rho = -2$.
2. Формирование кода

$$K(Z) = \dots \beta_m \alpha_m \dots \beta_1 \alpha_1 \beta_0 \alpha_0, \beta_{-1} \alpha_{-1} \beta_{-2} \alpha_{-2} \dots \quad (1)$$

комплексного числа $Z = X_\alpha + jX_\beta$, который в дальнейшем представляется в виде

$$K(Z) = \dots \gamma_m \dots, \quad (2)$$

$$\text{где } \begin{cases} \gamma_{2m} = \alpha_m \text{ if } m - \text{even} \\ \gamma_{2m+1} = \beta_m \text{ if } m - \text{odd} \end{cases}$$

Алгоритм 4. Декодирование в системе 1.

1. Выделение из кода (2) комплексного числа $Z = X_\alpha + jX_\beta$ четных и нечетных разрядов по правилу

$$\begin{cases} \alpha_{m/2} = \gamma_m \text{ if } m - \text{even} \\ \beta_{(m-1)/2} = \gamma_m \text{ if } m - \text{odd} \end{cases}$$
2. Формирование из разрядов α_m и β_m по основанию $\rho = -2$ для чисел X_α , X_β соответственно.
3. Декодирование действительных (положительных или отрицательных) чисел X_α , X_β из системы кодирования по основанию $\rho = -2$ в традиционную систему двоичного кодирования.

Алгоритм 5. Кодирование в системе 2 при $\rho = j\sqrt{2}$.

1. Вычисление $\bar{X}_\beta = \mu \cdot X_\beta$ при $\mu = 1/\sqrt{2}$. Это вычисление выполняется в традиционной системе двоичного кодирования.
2. Представление действительных (положительных или отрицательных) чисел X_α , \bar{X}_β в системе кодирования по основанию $\rho = -2$.
3. Формирование кода (1) комплексного числа $Z = X_\alpha + jX_\beta$, который в дальнейшем представляется в виде кода (2).

Алгоритм 6. Декодирование в системе 2 при $\rho = j\sqrt{2}$.

1. Выделение из кода (2) комплексного числа $Z = X_\alpha + jX_\beta$ четных и нечетных разрядов по правилу

$$\left\{ \begin{array}{l} \alpha_{m/2} = \gamma_m \text{ if } m - \text{even} \\ \beta_{(m-1)/2} = \gamma_m \text{ if } m - \text{odd} \end{array} \right\}$$
2. Формирование из разрядов α_m и β_m кодов по основанию $\rho = -2$ для чисел X_α , \bar{X}_β соответственно, где $\bar{X}_\beta = \mu \cdot X_\beta$ при $\mu = 1/\sqrt{2}$.
2. Декодирование действительных (положительных или отрицательных) чисел X_α , \bar{X}_β из системы кодирования по основанию $\rho = -2$ в традиционную систему двоичного кодирования.
3. Вычисление $X_\beta = \bar{X}_\beta \sqrt{2}$. Это вычисление выполняется в традиционной системе двоичного кодирования.

Упомянутые выше алгоритмы кодирования и декодирования кодов по основанию (-2) и соответствующие устройства подробно описаны в [45] и поэтому здесь не рассматриваются.

Глава 5. Умножение

5.1. Специальная алгебра в векторном пространстве

5.1.1. Алгебра в трехмерном векторном пространстве

Рассмотрим некоторую алгебру в трехмерном векторном пространстве. Пусть $\mathbf{i}, \mathbf{j}, \mathbf{k}$ – база трехмерного векторного пространства. Определим для этой базы табл. 1 умножения ортов. Согласно этой таблице умножение ортов описывается следующим образом: если $U = U_1 * U_2$, где

$$U_m = x_m i + y_m j + z_m k$$

для любых индексов m , то

$$x = x_1 x_2 - y_1 z_2 - z_1 y_2,$$

$$y = x_1 y_2 + y_1 x_2 - z_1 z_2,$$

$$z = x_1 z_2 + y_1 y_2 + z_1 x_2.$$

Таблица 1. Умножение трехмерных векторов

*	\mathbf{i}	\mathbf{j}	\mathbf{k}
\mathbf{i}	i	j	k
\mathbf{j}	j	k	$-i$
\mathbf{k}	k	$-i$	$-j$

Это умножение не имеет ничего общего с векторным или скалярным умножением и, в отличие от них, обозначается далее символом $*$. Нетрудно убедиться, что умножение, определенное табл. 1 для ортов $\mathbf{i}, \mathbf{j}, \mathbf{k}$ будет ассоциативным и коммутативным. Следовательно, умножение, определенное для любых векторов рассматриваемого векторного пространства, также будет ассоциативным и коммутативным и дистрибутивным относительно сложения. Кроме того, выполняется условие

$$(bU_1) * U_2 = U_1 * (bU_2) = b(U_1 * U_2),$$

где b - действительное число. Следовательно, табл. 1 определяет в трехмерном векторном пространстве алгебру без деления над полем действительных чисел.

Сложение в рассматриваемой алгебре соответствует обычному сложению векторов, а умножение - повороту вектора-множимого с одновременным изменением его длины. В общем случае параметры поворота - положение оси поворота, угол поворота, масштабный множитель зависят от координат обоих сомножителей. Таким образом, геометрическая интерпретация умножения в этой алгебре довольно сложна, однако несколькими операциями умножения и сложения можно описать такие преобразования векторов, которые имеют простой геометрический смысл.

5.1.2. Покомпонентное умножение

Рассмотрим некоторые операции в этой алгебре, предварительно определив операцию покомпонентного умножения. Этот термин будет использоваться для наименования операции умножения вектора U_1 на упорядоченную тройку векторов U_2, U_3, U_4 . Покомпонентное умножение состоит в вычислении вектора по формуле

$$U = x_1 i * U_2 + y_1 j * U_3 + z_1 k * U_4$$

или

$$x = x_1 x_2 - y_1 z_3 - z_1 y_4,$$

$$y = x_1 y_2 - y_1 x_3 - z_1 z_4,$$

$$z = x_1 z_2 - y_1 y_3 - z_1 x_4.$$

Для обозначения этой операции будем употреблять также запись следующего вида:

$$U = U_1 * [U_2, U_3, U_4].$$

В частности, $U_1 * [U_2, U_2, U_2] = U_1 * U_2$.

5.1.3. Векторное произведение

$$U_1 \times U_2 = U_1 * [(-jz_2 + ky_2), (-jx_2 - kz_2), (jy_2 - kx_2)].$$

5.1.4. Скалярное произведение

$$i(U_1 \bullet U_2) = U_1 * [(ix_2), (-ky_2), (-jz_2)]$$

- здесь для удобства вычислений действительное число $U_1 \bullet U_2$ отождествляется с вектором $i(U_1 \bullet U_2)$.

5.1.5. Поворот вектора

Поворот вектора, когда он перемещается по поверхности некоторого конуса, также может быть описан покомпонентным умножением на тройку векторов, полученных из параметров поворота. Здесь уместно отметить аналогию с алгеброй комплексных чисел, где умножение эквивалентно повороту плоского вектора.

Рассмотрим прямую с ортом

$$r_o = i\cos\alpha + j\cos\beta + k\cos\gamma,$$

проходящую через начало координат. Пусть вокруг этой прямой по окружности некоторого радиуса вращается точка. Ее радиус-вектор переходит из положения U_1 в положение U . Если, кроме того, вращение производится против часовой стрелки (при наблюдении с конца вектора r_o) и угол поворота $0 \leq \varphi \leq \pi$, то можно показать, что

$$U = U_1 \cos\varphi + (r_o \times U_1) \sin\varphi - r_o (r_o \bullet U_1) (1 - \cos\varphi)$$

или

$$U = U_1 * [U_2, U_3, U_4].$$

где

$$U_2 = \left\{ \begin{array}{l} i(\cos\varphi \cdot \sin^2\alpha + \cos^2\alpha) + \\ j(\cos\gamma \cdot \sin\varphi + \cos\alpha \cdot \cos\beta \cdot (1 - \cos\varphi)) + \\ k(-\cos\beta \cdot \sin\varphi + \cos\alpha \cdot \cos\gamma \cdot (1 - \cos\varphi)) \end{array} \right\}$$

$$U_3 = \left\{ \begin{array}{l} i(\cos\varphi \cdot \sin^2\beta + \cos^2\beta) + \\ j(\cos\alpha \cdot \sin\varphi + \cos\beta \cdot \cos\lambda \cdot (1 - \cos\varphi)) + \\ k(\cos\lambda \cdot \sin\varphi - \cos\alpha \cdot \cos\beta \cdot (1 - \cos\varphi)) \end{array} \right\}$$

$$U_4 = \left\{ \begin{array}{l} i(\cos\varphi \cdot \sin^2\gamma + \cos^2\gamma) + \\ j(-\cos\beta \cdot \sin\varphi - \cos\alpha \cdot \cos\lambda \cdot (1 - \cos\varphi)) + \\ k(\cos\alpha \cdot \cos\varphi - \cos\beta \cdot \cos\lambda \cdot (1 - \cos\varphi)) \end{array} \right\}$$

5.1.6. Центроаффинное преобразование

Центроаффинное преобразование эквивалентно покомпонентному умножению на тройку векторов, построенных из элементов матрицы центроаффинного преобразования:

$$U_2 = U_1^T \cdot \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = U_1 * \begin{bmatrix} (ia_{11} - ja_{12} - ka_{13}), \\ (ia_{21} - ja_{22} - ka_{23}), \\ (ia_{31} - ja_{32} - ka_{33}) \end{bmatrix}.$$

5.1.7. Многомерное пространство

Обобщим теперь полученные результаты на n -мерное пространство. Выберем в нем базу E_1, E_2, \dots, E_n , элементы которой удовлетворяют условию

$$E_a E_b = -E_c \text{ при } d > n+1,$$

$$E_a E_b = E_c \text{ при } d < n+2,$$

$$\text{где } d = (a+b-1), c = d \bmod n. \quad (1)$$

Можно показать, что умножение, определенное для элементов этой базы, является ассоциативным, коммутативным и дистрибутивным относительно сложения, а также удовлетворяет условию

$$(bU_1) * U_2 = U_1 * (bU_2) = b(U_1 * U_2),$$

где b - действительное число. Следовательно, множество n -мерных векторов составляет алгебру. В частности, при $n=2$ и базе этого пространства $\{1, j\}$ получаем алгебру комплексных чисел - см. также табл. 2 умножения комплексных чисел; при $n=3$ получаем алгебру с табл. 1, описанную выше; при $n=4$ последней формуле соответствует табл. 3 умножения четырех ортов и так далее.

Таблица 2. Умножение комплексных чисел.

*	1	j
1	1	j
j	j	-1

Таблица 3. Умножение четырехмерных векторов

*	i	j	k	m
i	i	j	k	m
j	j	k	m	-i
k	k	m	-i	-j
m	m	-i	-j	-k

5.2. Умножение многомерных векторов

5.2.1. Метод умножения комплексных чисел и многомерных векторов

Необходимо найти произведение комплексных чисел или векторов $C=A*B$, где множитель и множимое имеют соответственно разложения

$$A = \sum_h \alpha_h f(\rho, h), \quad B = \sum_k \beta_k f(\rho, k).$$

Код произведения определяется как $C = \sum_h [B \alpha_h f(\rho, h)]$.

Поскольку $\alpha_h = \{0,1\}$, то умножение состоит только из операций умножения на базовую функцию $f(\rho, h)$ и суммирования.

Пример 1. Умножение в системе теоремы 1.6. Умножение двух разрядов в этой системе описывается табл. 1.2. При этом даже не возникают переносы. Рассмотрим, например, поворот плоского вектора - комплексного числа на 60° . Такой поворот соответствует умножению на код **1d** - см. табл. 1.4а. Умножение на код **d** эквивалентно преобразованию всех разрядов множимого в соответствии со столбцом 'd' табл. 1.2. Умножение на код **10**, как всегда, равносильно сдвигу влево. Сложение результатов этих двух действий дает код повернутого на 60° вектора.

Покомпонентное умножение векторов также состоит из циклов 'сдвиг-сложение'. Однако, в отличие от простого умножения действительных и комплексных чисел, в каждом цикле покомпонентного умножения значение множимого, с которым производится сложение, зависит от номера разряда множителя. В частности, если выполняется покомпонентное умножение

$$Z = Y * (X, V, W),$$

то множимое M определяется следующим образом:

$$M = X, \text{ если } m = 3k,$$

$$M = V, \text{ если } m = 3k+1,$$

$$M = W, \text{ если } m = 3k+2,$$

где m - номер разряда множителя Y , k - целое число.

Рассмотрим умножение на базовую функцию для двух важных случаев.

5.2.2. Умножение векторов по основанию (1.31).

В этом случае умножение множимого на основание равносильно сдвигу на h разрядов. Таким образом, умножение кодов в этой системе сводится к последовательно выполняемым операциям сдвига и сложения.

5.2.3. Умножение векторов по основанию (1.30).

Разрядность кода множителя $N = n \cdot m$, где n – размерность вектора. Код множителя можно разбить на m групп, а в каждой t -группе рассматривать первый (младший) разряд, второй разряд, ... i -разряд, ... n -разряд. Группы будем нумеровать также, как разряды кода, справа налево от 0 до $(m-1)$. При этом умножение множимого на базовую функцию (если соответствующий множителя равен 1) состоит из двух действий (которые совмещаются во времени):

1. Сдвиг множимого на $h = n \cdot t$ разрядов, если рассматривается t -группа разрядов множителя.
2. Умножение множимого B на орт в зависимости от номера i разряда в группе:

$$B_i = E_i B, \quad i = \overline{1, n} \quad (2)$$

- см. также (1.21). Например, при $n=2$, имеем:

$$B_1 = B, \quad B_2 = jB;$$

при $n=3$, имеем:

$$B_1 = iB, \quad B_2 = jB, \quad B_3 = kB;$$

при $n=4$, имеем:

$$B_1 = iB, \quad B_2 = jB, \quad B_3 = kB, \quad B_4 = mB$$

и т.д. Заметим, что преобразованные множимые B_i могут быть заготовлены перед умножением.

Вычисление по формуле (2) выполняется в соответствии с таблицей умножения вектора или формулой (1). Пусть в соответствии с (1.21)

$$B = E_1 b_1 + E_2 b_2 + \dots + E_n b_n,$$

В частности, для комплексных чисел используется табл. 2. Имеем:

$$B_1 = B, \quad B_2 = j(b_1 + jb_2) = -b_2 + jb_1.$$

Для трехмерных векторов используется табл. 1. Имеем:

$$B_1 = B,$$

$$B_2 = j(b_1 + jb_2 + kb_3) = -b_3 + jb_1 + kb_2,$$

$$B_{32} = k(b_1 + jb_2 + kb_3) = -b_2 - jb_3 + kb_1.$$

Для четырехмерных векторов используется табл. 3. Имеем:

$$B_1 = B,$$

$$B_2 = j(b_1 + jb_2 + kb_3 + mb_4) = -b_4 + jb_1 + kb_2 + mb_3,$$

$$B_3 = k(b_1 + jb_2 + kb_3 + mb_4) = -b_3 - jb_4 + kb_1 + mb_2,$$

$$B_4 = m(b_1 + jb_2 + kb_3 + mb_4) = -b_2 - jb_3 - kb_4 + mb_1.$$

Отсюда следует, что умножение кода вектора на орт состоит из инвертирования некоторых компонент и перестановки компонент кода вектора.

5.2.4. Последовательное и матричное умножение векторов.

Для комплексных чисел и многомерных векторов можно (аналогично умножению действительных чисел) предложить последовательные и матричные алгоритмы. Рассмотрим некоторые примеры.

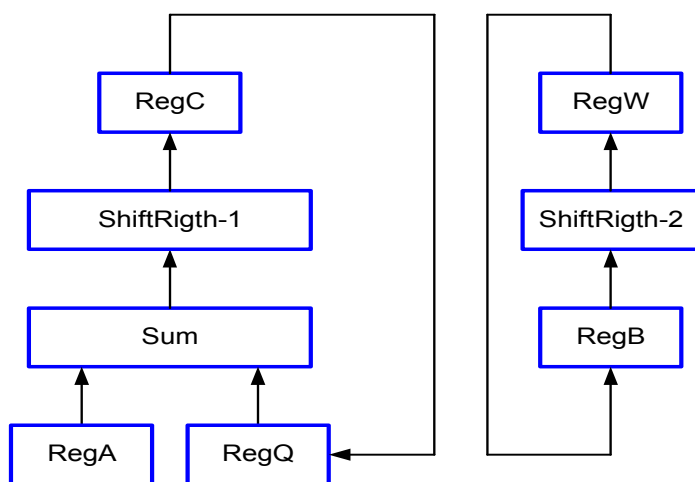


Рис. 1.

На рис. 1 приведена схема последовательного умножения, которая применима и для умножения комплексных чисел $C=A*B$.

В начале умножения множитель A записывается в регистр $RegA$, множимое B - в регистр $RegB$, а в регистр $RegQ$ частичного произведения записывается «0». На каждом шаге

- анализируется m младших разрядов $RegB$; комплексное значение этих разрядов обозначим через M ;
- в сумматоре Sum вычисляется комплексный код числа $S = M * RegA + RegQ$;
- комплексный код S с выхода сумматора Sum через сдвигатель $ShiftRigth-1$ записывается в $RegC$ со сдвигом на m разрядов вправо;
- код из $RegC$ пересылается в $RegQ$;
- комплексный код из регистра $RegB$ через сдвигатель $ShiftRigth-2$ записывается в $RegW$ со сдвигом на m разрядов вправо;
- код из $RegW$ пересылается в $RegB$.

Количество m анализируемых на каждом шаге разрядов определяет быстродействие умножения и сложность сумматора. Например, в системах 1 и 2 при $m = 2$ число $M = \{1, 0, -1, -2\}$ и сумматор должен выполнять сложение, вычитание и вычитание удвоенного числа.

Для умножения комплексных кодов $C = A * B$, кроме рассмотренной выше последовательной схемы, может быть применена матричная схема умножения – см. рис. 2. Такая схема содержит регистры $RegA$, $RegB$, $RegC$ и матричный множитель MM , который состоит из множества сумматоров $Add(K)$. Первые входы всех сумматоров подключены ко выходу регистра $RegA$. К управляющему входу каждого сумматора $Add(K)$ подключен выход K -разряда регистра $RegB$. Выход каждого сумматора (кроме $Add(N)$) соединен со входом следующего сумматора со сдвигом на 1 разряд. Выход сумматора $Add(N)$ соединен со входом регистра $RegC$.

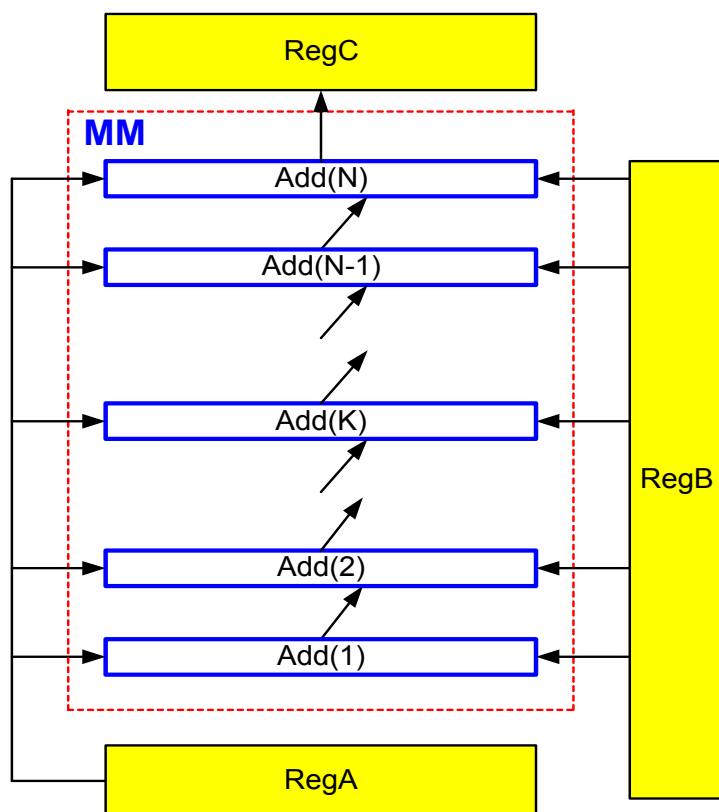


Рис. 2.

В начале умножения множитель A записывается в регистр RegA, а множимое B - в регистр RegB. Если K -разряд регистра RegB равен «1», то сумматор Add(K) складывает комплексный код A со сдвинутым на 1 разряд кодом выхода предыдущего сумматора. Если K -разряд регистра RegB равен «0», то сумматор Add(K) передает дальше сдвинутый на 1 разряд код выхода предыдущего сумматора. Таким образом на входе сумматора Add(N) образуется комплексный код результата, который записывается в регистр RegC.

Подробно различные варианты умножителей рассмотрены в [45].

Далее мы рассмотрим простейшие алгоритмы последовательного умножения векторов. По аналогии с ним можно построить алгоритмы матричного умножения векторов.

5.2.5. Умножение целых кодов векторов по основанию (1.31).

Рассмотрим систему кодирования n -мерных векторов по основанию (1.31). Будем анализировать разряды множителя, начиная со старшего, и сдвигать множимое вправо. В соответствии с этим алгоритм умножения имеет вид:

1. Вначале частичное произведение равно 0, а множимое B расположено так, что его младший 0 -разряд совмещен со старшим $(N-1)$ -разрядом α_{N-1} множителя A . Номер текущего разряда множителя $t=N-1$, т.е. $\alpha_t = \alpha_{N-1}$.
2. Сложение частичного произведения со множимым B , если $\alpha_t = 1$.
3. Сдвиг множимого B на 1 разряд вправо и уменьшение текущего номера $t := t - 1$.
4. Прекращение вычисления, если $t < 0$, или переход к п. 2.

5.2.6. Умножение целых кодов векторов по основанию (1.30).

Рассмотрим систему кодирования n -мерных векторов по основанию (1.30). Разрядность кода множителя $N = n \cdot m$. В этом случае алгоритм умножения имеет вид:

1. Подготавливаются n вариантов множимого B по формуле (2).
2. Будем рассматривать группы по n разрядов множителя. Вначале частичное произведение равно 0, а множимые B_i расположены так, что их младшие 0 -разряды совмещены с разрядом множителя A , имеющего номер $N - n = n \cdot (m - 1)$. Номер текущей группы разрядов множителя $t=m$.
3. Рассматривается t -группа разрядов множителя A . В ней
 - 3.1. Рассматривается первый (младший) разряд $\alpha_{n(t-1)}$.
Выполняется сложение частичного произведения со множимым B_1 , если $\alpha_{n(t-1)} = 1$.
 - 3.2. Рассматривается второй разряд $\alpha_{n(t-1)+1}$. Выполняется сложение частичного произведения со множимым B_2 , если $\alpha_{n(t-1)+1} = 1$.

...

- 3.и. Рассматривается i -разряд $\alpha_{n(t-1)+i}$. Выполняется сложение частичного произведения со множимым B_i , если $\alpha_{n(t-1)+i} = 1$.
- ...
- 3.п. Рассматривается n -разряд α_{nt} . Выполняется сложение частичного произведения со множимым B_n , если $\alpha_{nt} = 1$.
4. Сдвиг множимого на n разряд вправо (напомним, что здесь n – размерность вектора) и уменьшение текущего номера $t := t - 1$.
5. Прекращение вычисления, если $t < 0$, или переход к п. 3.

5.2.7. Покомпонентное умножение многомерных векторов.

В отличие от простого умножения, в каждом цикле покомпонентного умножения значение множимого, с которым производится сложение, зависит от номера m разряда множителя (т.е. от того, к какой компоненте вектора множителя принадлежит этот разряд). Пусть

m - номер разряда множителя A ,
 k - целое число.

Если выполняется покомпонентное умножение *комплексного числа*

$$C = A * (X, Y),$$

то множимое B определяется следующим образом:

$$B = X, \text{ если } m = 3k, \\ B = Y, \text{ если } m = 3k+1.$$

Если выполняется покомпонентное умножение *трехмерного вектора*

$$C = A * (X, Y, V),$$

то множимое B определяется следующим образом:

$$B = X, \text{ если } m = 3k, \\ B = Y, \text{ если } m = 3k+1, \\ B = V, \text{ если } m = 3k+2.$$

Если выполняется покомпонентное умножение *четырёхмерного вектора*

$$C = A * (X, Y, V, W),$$

то множимое B определяется следующим образом:

$$B = X, \text{ если } m = 3k,$$

$$B = Y, \text{ если } m = 3k+1,$$

$$B = V, \text{ если } m = 3k+2,$$

$$B = W, \text{ если } m = 3k+3.$$

Как показано выше, покомпонентное умножение на заранее определенные тройки векторов эквивалентно скалярному и векторному умножению, умножению на число, центроаффинному преобразованию и т.д.

5.3. Скалярное и векторное умножения

При выполнении операций *скалярного и векторного* умножения, подчиняющихся каким-либо законам, которые должны выполняться в кольце, реализация умножения усложняется. Выше было показано, что эти операции могут быть заменены покомпонентным умножением. Однако при этом необходимо предварительно формировать сомножители. Поэтому далее рассматриваются другие способы скалярного и векторного умножений трехмерных векторов в системе кодирования трехмерных векторов по основанию (1.30). В этой системе код вектора имеет вид $K(Z) = \dots \sigma_m \dots$, а его разряды принимают значения $r_m \in \{0, i, j, k, i+j, i+k, j+k, i+j+k\}$. Для их обозначения будем использовать 8 следующих «цифр»:

$$\sigma_m = a, b, c, d, e, f, g, h.$$

Ниже разряды векторов-сомножителей V и W представляются векторами v_m и w_m с тремя компонентами - действительными числами, принимающими значения 0 или 1:

$$v_m = \{\alpha', \beta', \gamma'\}, w_m = \{\alpha'', \beta'', \gamma''\}.$$

5.3.1. Скалярное произведение

По формуле для скалярного произведения

$$v \bullet w = \alpha' \alpha'' + \beta' \beta'' + \gamma' \gamma'' \quad (3)$$

строится табл. 4, в которой для произведений - чисел указаны коды по основанию $\rho = -2$.

Таблица 4. Одноразрядное скалярное умножение.

•	a	b	c	d	e	f	g	h
a	0							
b	0	1						
c	0	0	1					
d	0	1	1	110				
e	0	0	0	0	1			
f	0	1	0	1	1	110		
g	0	0	1	1	1	1	110	
h	0	1	1	110	1	110	110	111

Скалярное произведение $Z = V \bullet W$ может быть вычислено по формуле

$$Z = \sum_k (V \bullet w_k) \rho^k.$$

Следовательно, скалярное умножение многоразрядных кодов векторов состоит из циклов 'сдвиг - скалярное умножение на k -ый разряд множителя - сложение'. В результате образуется код числа Z по основанию $\rho = -2$.

5.3.2. Векторное произведение

Формула векторного произведения имеет вид: $Z = V \times W$, причем $z_m = \{\alpha, \beta, \gamma\}$, где

$$\begin{aligned}\alpha &= \beta' \gamma'' - \gamma' \beta'', \\ \beta &= \gamma' \alpha'' - \alpha' \gamma'', \\ \gamma &= \alpha' \beta'' - \beta' \alpha''.\end{aligned}\tag{4}$$

Координаты вектора \mathbf{Z} , вычисленные по формуле (4), могут принимать значения -1, 0, 1. Следовательно, вектор \mathbf{Z} всегда можно представить как разность двух векторов, каждый из которых имеет код по основанию (1.30). Используя, далее, правила алгебраического сложения этих векторов, можно построить табл. 5, описывающую векторное умножение. В отличие от предыдущей таблицы эта таблица должна быть заполнена полностью, так как является несимметричной относительно диагонали (векторное произведение некоммутативно).

Таблица 5. Одноразрядное векторное умножение.

\times	a	b	c	d	e	f	g	h
a	0	0	0	0	0	0	0	0
b	0	0	e	e	cc	cc	cg	cg
c	0	ee	0	ee	b	ef	b	ef
d	0	ee	e	0	cd	gh	ch	cd
e	0	cc	bb	bd	0	c	bb	bd
f	0	cc	bf	bh	cc	0	dh	bf
g	0	eg	bb	fh	b	eh	0	eg
h	0	eg	bf	bd	cd	ef	cg	0

Векторное произведение $Z=V \times W$ может быть вычислено по формуле $Z = \sum_k (V \times w_k) \rho^k$. Следовательно, векторное

умножение многоразрядных кодов векторов состоит из циклов 'сдвиг - векторное умножение на k -ый разряд множителя - сложение'. В результате образуется код вектора Z по основанию (1.30).

5.3.3. Переносы при скалярном умножении.

При выполнении операций сдвига и алгебраического сложения код вектора удобно рассматривать как состоящий из трех независимых частей - кодов чисел - проекций вектора и выполнять указанные операции с каждой из этих частей независимо. Однако при векторном и скалярном умножении одного кода вектора на разряд другого кода дело усложняется тем, имеется перекрестное воздействие разрядов неоднородных частей друг на друга. Рассмотрим этот вопрос подробнее сначала для скалярного умножения.

Скалярное умножение описывается формулой (3), но в случае многоразрядного кода необходимо еще учитывать перенос p из младшего разряда. При этом разрядный результат должен вычисляться по формуле

$$S = \alpha' \alpha'' + \beta' \beta'' + \gamma' \gamma'' + p. \quad (5)$$

Величина S должна быть представлена в виде

$$S = \sigma + P \rho, \quad (6)$$

где $\sigma = (0, 1)$ - значение данного разряда результата,
 P - значение переноса в старший разряд.

Нетрудно показать, что перенос P из данного разряда (а, следовательно, и перенос p в данный разряд) может принимать одно из следующих значений: $P = (-1, 0, 1, 2)$. При этом $S = (-1, 0, 1, 2, 3, 4, 5)$ и скалярное умножение описывается табл. 6.

Таблица 6. Переносы при скалярном умножении.

S	-1	0	1	2	3	4	5
σ	1	0	1	0	1	0	1
P	1	0	0	-1	-1	2	2

Распространение переносов можно организовать иначе, а именно, так, чтобы перенос в данный разряд поступал из двух предыдущих (p и q) и передавался из данного разряда в два последующих (P и Q). В этом случае разрядный результат должен вычисляться по формуле

$$S = \alpha' \alpha'' + \beta' \beta'' + \gamma' \gamma'' + p + q. \quad (7)$$

и представляться в виде

$$S = \sigma + P\rho + Q\rho^2. \quad (8)$$

В этом случае переносы могут принимать лишь два значения (0,1) и $S = (0, 1, 2, 3, 4, 5)$. При этом скалярное умножение описывается табл. 7.

Таблица 7. Переносы при скалярном умножении.

S	0	1	2	3	4	5
σ	0	1	0	1	0	1
P	0	0	1	1	0	0
Q	0	0	1	1	1	1

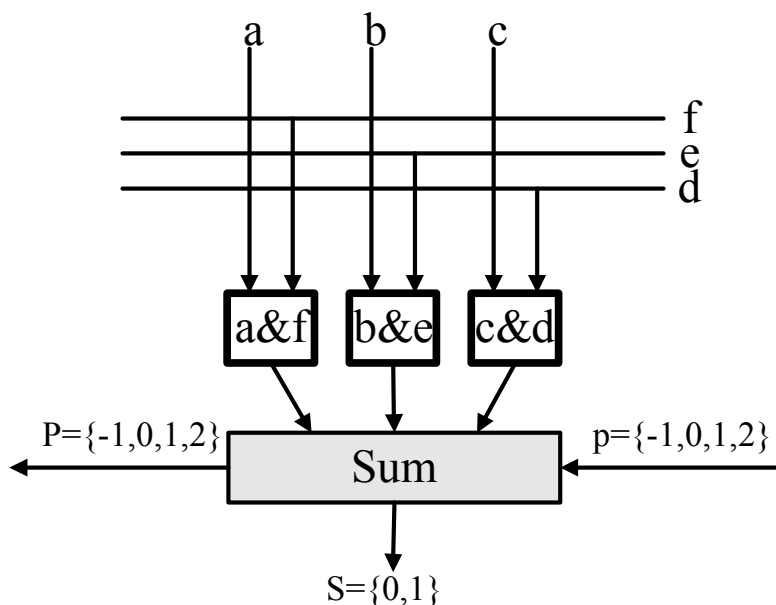


Рис. 3. Сумматор в блоке скалярного умножения

На рис. 3 приведена схема сумматора в блоке скалярного умножения на один разряд множителя. На этой схеме

a, b, c – разряды кода множимого,

d, e, f – разряд кода множителя,

p – входной перенос,

P – выходной перенос,

Sum – одноразрядный сумматор.

5.3.4. Переносы при векторном умножении.

Векторное умножение описывается формулами (4). С учетом переносов она приобретает следующий вид:

$$\begin{aligned}\rho P_{\alpha} + \alpha &= p_{\alpha} + \beta' \gamma'' - \gamma' \beta'', \\ \rho P_{\beta} + \beta &= p_{\beta} + \gamma' \alpha'' - \alpha' \gamma'', \\ \rho P_{\gamma} + \gamma &= p_{\gamma} + \alpha' \beta'' - \beta' \alpha''.\end{aligned}\tag{9}$$

где p, P – значения переносов в данный и в старший разряды. В формуле (9) переносы принимают только два значения (0,1). Первая (в частности) из этих формул описывается табл. 8.

Таблица 8. Переносы при векторном умножении.

$\beta' \gamma''$	$\gamma' \beta''$	p_{α}	α	P_{α}
0	0	0	0	0
0	1	0	1	1
1	0	0	1	0
1	1	0	0	0
0	0	1	1	0
0	1	1	0	0
1	0	1	0	-1
1	1	1	1	0
0	0	-1	1	1
0	1	-1	0	1
1	0	-1	0	0
1	1	-1	1	1

На рис. 4 приведена схема сумматора в блоке векторного умножения на один разряд множителя. На этой схеме

a, b, c – разряды кода множимого,

d, e, f – разряд кода множителя,

pG, pH, pM – входные переносы,

PG, PH, PM – выходные переносы,

SumG, SumH, SumM – одноразрядные сумматоры.

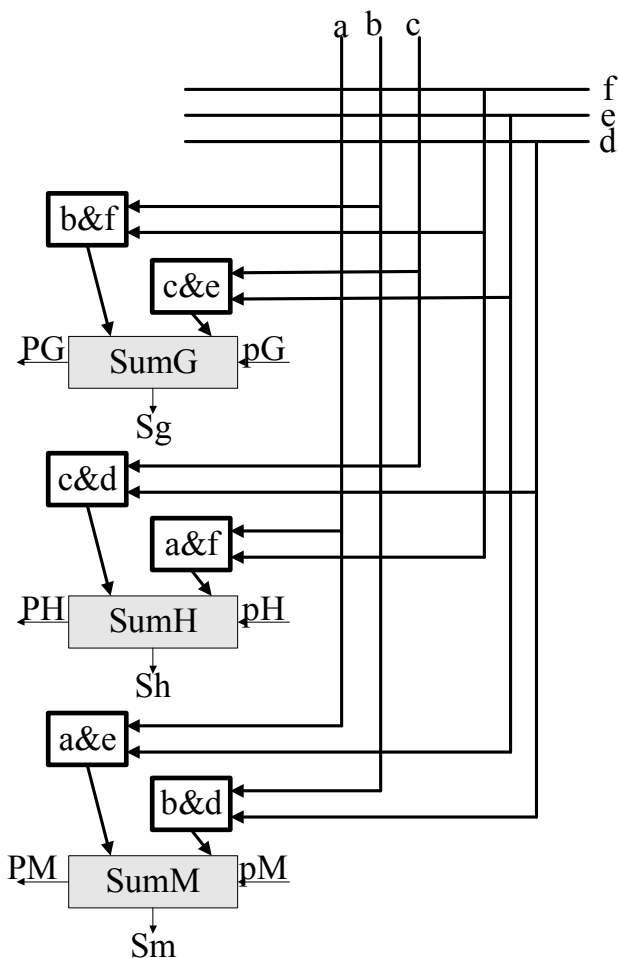


Рис. 4. Сумматор в блоке векторного умножения

Глава 6. Метод «цифра за цифрой»

6.1. Введение

В связи с существованием кодов комплексных чисел возникают, естественно, новые возможности для вычисления функций комплексного аргумента и решения уравнений с комплексными неизвестными.

В данной главе описывается метод решения этих задач, называемый методом «цифра за цифрой». Этот метод известен в применении к действительным числам [48, 49]. Ниже показывается, что он может быть обобщен на коды комплексных чисел. Затем разрабатываются алгоритмы вычисления различных функций и решения некоторого класса трансцендентных уравнений.

Пусть Z – некоторое комплексное число. Рассмотрим последовательность комплексных чисел Z_h ($h=1, \dots, m$):

$$Z_1, Z_2, Z_3, \dots, Z_h, Z_{h+1}, \dots, Z_m,$$

последовательность комплексных кодов $K(Z - Z_h)$:

$$K(Z - Z_1), K(Z - Z_2), \dots, K(Z - Z_h), \dots, K(Z - Z_m)$$

Предположим, что код комплексного числа имеет некоторую характеристику, называемую далее (условно) **размером кода**. Будем обозначать эту характеристику кода комплексного числа Z символом $NK(Z)$. В качестве размера кода может использоваться номер старшего значащего разряда или модуль кодируемого числа. Рассмотрим последовательность размеров $NK(Z - Z_h)$:

$$NK(Z - Z_1), NK(Z - Z_2), \dots, NK(Z - Z_h), \dots, NK(Z - Z_m)$$

Последовательность чисел Z_h будем называть образующей последовательностью $\{Z_h\}$, если при любом $h=1, \dots, m$ выполняется условие:

$$NK(Z - Z_{h+1}) \leq NK(Z - Z_h) \quad (1)$$

Метод сравнения размера кодов рассматривается далее.

Потребуем, чтобы числа Z_h удовлетворяли условию рекуррентности:

$$Z_{h+1} = \Phi[Z_h, \varepsilon_{h+1}, \rho^{h+1}] \quad (2)$$

где $\varepsilon_h = 0, 1, 2, \dots$; ρ - основание системы кодирования.

Выражением ρ^h мы далее будем пользоваться для обозначения базовой функции $f(\rho, h)$, которая не всегда равна величине ρ^h - см. выше.

Рассматривая (1) и (2) совместно находим:

$$NK[Z - \Phi(Z_h, \varepsilon_{h+1}, \rho^{h+1})] \leq NK[Z - Z_h]. \quad (3)$$

Для кодов действительных чисел по действительному основанию неравенство (3) эквивалентно неравенству с модулями

$$|Z - \Phi(Z_h, \varepsilon_{h+1}, \rho^{h+1})| \leq |Z - Z_h|,$$

т. к. в этом случае число с большим номером старшего значащего разряда имеет больший модуль и наоборот.

Вернемся снова к образующей последовательности $\{Z_h\}$.

Каждый h - член (h -ая образующая Z_h) этой последовательности представляет число Z с некоторой точностью, а именно с точностью до максимального модуля кода, имеющего размер $NK(Z - Z_h)$. Следовательно, представление числа Z с допустимой максимальной абсолютной погрешностью Δ равносильно вычислению такого m - члена Z_m образующей последовательности, при котором $NK(Z - Z_m)$ имеет величину, равную максимальному размеру числа с модулем, не превышающем Δ .

Обозначим:

$$NK(Z - Z_m) = H(m). \quad (4)$$

6.2. Декомпозиция

6.2.1. Вступление

Перейдем к описанию алгоритма вычисления Z_m , для чего предварительно рассмотрим процесс перехода от Z_h к Z_{h+1} . Итак известна h -ая образующая Z_h . Для определения $(h+1)$ -ой образующей Z_{h+1} нужно найти ту максимальную величину a_{h+1} , при которой Z_{h+1} , вычисленная по формуле (1.2), еще удовлетворяет условию (1.1). Очевидно этому условию при $a_{h+1} = 0$ удовлетворяет $Z_{h+1}^{(0)} = Z_h$. При $a_{h+1} = 1$ проверке на удовлетворение условия (1.1) подлежит величина $\bar{Z}_{h+1}^{(1)} = \Phi[Z_h, 1, \rho^{h+1}]$. Проверка заключается в сравнении размеров кодов чисел $(Z - Z_h)$ и $(Z - \bar{Z}_{h+1}^{(1)})$. При сравнении возможны три варианта:

- а) $NK[Z - \bar{Z}_{h+1}^{(1)}] < NK[Z - Z_h]$; это неравенство свидетельствует о том, что $a_{h+1} = 1$ и $Z_{h+1} = Z_{h+1}^{(1)}$;
- б) $NK[Z - \bar{Z}_{h+1}^{(1)}] > NK[Z - Z_h]$; это неравенство свидетельствует о том, что $a_{h+1} = 0$ и $Z_{h+1} = Z_h$;
- в) $NK[Z - \bar{Z}_{h+1}^{(1)}] = NK[Z - Z_h]$; это равенство свидетельствует о том, что $a_{h+1} \geq 1$ и следует проверить на соответствие условию (1.3) величины $\bar{Z}_{h+1}^{(2)} = \Phi[Z_h, 2, \rho^{h+1}]$ или $\bar{Z}_{h+1}^{(2)} = \Phi[Z_{h+1}^{(1)}, 1, \rho^{h+1}]$. Эта проверка осуществляется аналогично сравнением кодов чисел $(Z - Z_{h+1}^{(1)})$ и $(Z - Z_{h+1}^{(2)})$. Результатом проверки является либо установление значения величин $a_{h+1} = 1$ or $a_{h+1} = 2$ и $Z_{h+1} = \bar{Z}_{h+1}^{(1)}$ or $Z_{h+1} = \bar{Z}_{h+1}^{(2)}$,

либо переход к опросу кодов чисел $\left(Z - \bar{Z}_{h+1}^{(2)}\right)$ и $\left(Z - \bar{Z}_{h+1}^{(3)}\right)$, где $\bar{Z}_{h+1}^{(3)} = \Phi\left[Z_{h+1}^{(2)}, 1, \rho^{h+1}\right]$.

Таким образом, в результате последовательного применения проверки к кодам чисел $\left(Z - \bar{Z}_{h+1}^{(\mu-1)}\right)$ и $\left(Z - \bar{Z}_{h+1}^{(\mu)}\right)$, где $\mu = 0, 1, 2, \dots, a_{h+1}$, определяется величина $Z_{h+1} = \bar{Z}_{h+1}^{(a_{h+1})}$ по известной величине Z_h . Очевидно, что процесс вычисления $(h+1)$ -ой образующей по известной h -ой образующей можно начинать от первой образующей Z_1 и заканчивать m -ой образующей Z_m .

6.2.2. Алгоритм декомпозиции

В дальнейшем вычисление образующей последовательности будем называть декомпозицией, т.е. представлением комплексного числа в виде суммы или произведения некоторых других известных чисел – элементов разложения.

Алгоритм, описывающий h -ый цикл вычисления Z_m , имеет следующий вид.

Известна предыдущая образующая $\bar{Z}_h^{(\mu)} = Z_{prev} \ (\mu = 0, 1, 2, \dots)$,

$H(m)$, h и разность $E_{prev} = Z - Z_{prev}$.

1. Определяется ожидаемая следующая образующая

$\bar{Z}_h^{(\mu+1)} = Z_{next}$ по формуле

$$Z_{next} = \Phi\left[Z_{prev}, 1, \rho^{h+1}\right] \quad (1)$$

2. Вычисляется разность $E_{next} = Z - Z_{next}$.

3. Производится сравнение размеров $H_{prev} = NK(E_{prev})$ и $H_{next} = NK(E_{next})$. Далее

по результатам сравнения H_{prev} и H_{next} :

4. Определяется номер следующего цикла

$$h' = \left\{ \begin{array}{ll} h & \text{if } H_{next} = H_{prev} \\ h+1 & \text{if } H_{next} \neq H_{prev} \end{array} \right\}.$$

5. Определяется значение предыдущей образующей для следующего цикла

$$Z'_{prev} = \begin{cases} Z_{prev} & \text{if } H_{next} > H_{prev} \\ Z_{next} & \text{if } H_{next} \leq H_{prev} \end{cases}.$$

6. Определяется разность $E'_{prev} = Z - Z'_{prev}$ для следующего цикла.

7. Проверяется выполнение условия $H_{next} = H(m)$; удовлетворение этого равенства свидетельствует о том, что вычислена m -ая образующая Z_m , т.е. вычисление закончено; при неудовлетворении – выполняется h' - цикла вычислений.

Пункты 4, 5, 6 выполняются согласно табл. 1 (без столбца W'_{prev}).

При этом, если $NK(E_{next}) \leq NK(E_{prev})$, то соответствующий элемент включается в состав разложения, т.е. число a_h увеличивается на 1; кроме того, значение величины E обновляется ($E_{prev} = E_{next}$) и выполняется переход к следующему шагу с прежним значением h . В противном случае выполняется переход к следующему шагу с уменьшенным на 1 значением h и значением $a_h = 0$.

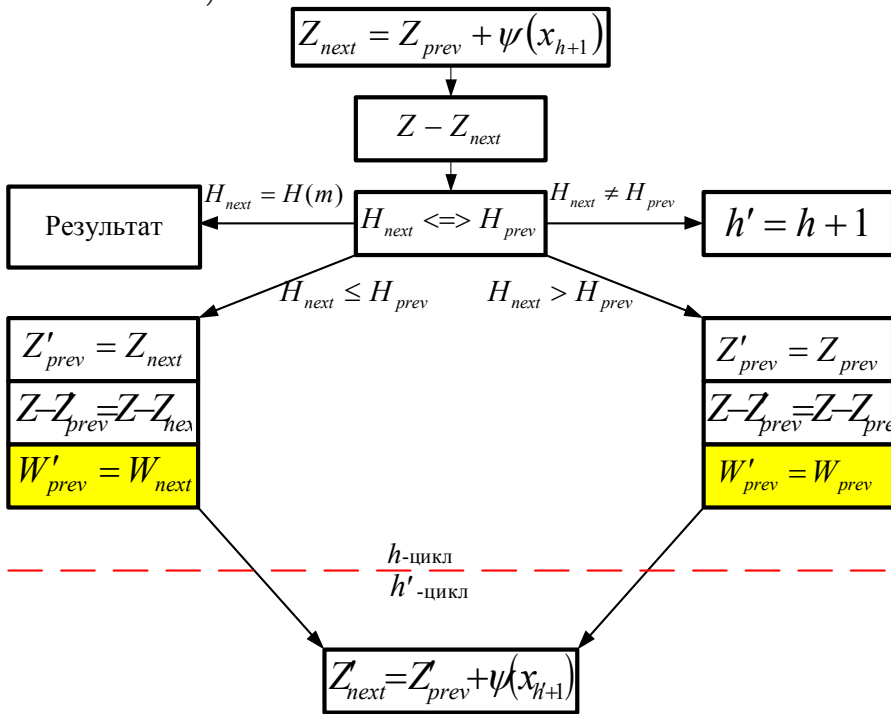
Таблица 1.

Результат проверки	h'	Z'_{prev}	$Z - Z'_{prev}$	W'_{prev}
$H_{next} < H_{prev}$	$h+1$	Z_{next}	$Z - Z_{next}$	W_{next}
$H_{next} = H_{prev}$	h	Z_{next}	$Z - Z_{next}$	W_{next}
$H_{next} > H_{prev}$	$h+1$	Z_{prev}	$Z - Z_{prev}$	W_{prev}

Последовательность чисел a_h является результатом декомпозиции. Каждый потенциальный элемент разложения может отсутствовать в

конкретном разложении или присутствовать в нем, повторяясь a_h раз, т.е. несколько циклов могут иметь один и тот же номер (при $a_{h+1} > 1$); поэтому число циклов Ω может превосходить число образующих m .

Таким образом, по алгоритму декомпозиции вычисляется образующая последовательность $\{Z_h\}$. Этот алгоритм содержит лишь элементарные машинные операции сложения, вычитания и сравнения размеров и поэтому легко реализуется в арифметическом устройстве. Более наглядно последовательность элементарных операций алгоритма декомпозиции изображена на рис.1 (без выделенного блока).



6.2.3. Варианты декомпозиций

В табл. 2 и табл. 3 перечислены виды декомпозиций, которые используются далее, и формулы для их вычисления.

Таблица 2. Каталог декомпозиций.

#	Обозначение	Назначение
D1	DecompDivis	Декомпозиция для вычисления обратной величины
D1a	DecompDivis1a	Декомпозиция для деления
D2	DecompLogar	Декомпозиция на сумму логарифмов
D3	DecompBinom	Декомпозиция на произведение биномов
D32	DecompBinomInv	Декомпозиция на инверсное произведение биномов
D4	DecompBinom2	Декомпозиция на произведение квадратов биномов
D42	DecompBinom2inv	Декомпозиция на инверсное произведение квадратов биномов
D1R	DecompDivisReal	Декомпозиция для деления действительных чисел (h - четное)
D2R	DecompLogarReal	Декомпозиция на сумму логарифмов действительных чисел (h - четное)
D3R	DecompBinomReal	Декомпозиция на произведение действительных биномов (h - четное)
D32R	DecompBinomInvReal	Декомпозиция на инверсное произведение действительных биномов (h - четное)
D4R	DecompBinom2real	Декомпозиция на произведение квадратов действительных биномов (h - четное)
D42R	DecompBinom2invReal	Декомпозиция на инверсное произведение квадратов действительных биномов (h - четное)

Таблица 3. Формулы декомпозиций.

#	Декомпозиция	Zo	Enext =	Eo
D1	$\frac{1}{Z} = \sum_h a_h \rho^{-h}$	0	$E_{\text{prev}} - Z\rho^{-h}$	1
D1a	$\frac{Y}{Z} = \sum_h a_h \rho^{-h}$	0	$E_{\text{prev}} - Z\rho^{-h}$	Y
D2	$Z = \sum_h a_h \ln(1 + \rho^{-h})$	0	$E_{\text{prev}} - \ln(1 + \rho^{-h})$	Z
D3	$Z = \prod (1 + \rho^{-h})^{a_h}$	1	$E_{\text{prev}} - (Z - E_{\text{prev}})\rho^{-h}$	Z-1
D32	$\frac{1}{Z} = \prod (1 + \rho^{-h})^{a_h}$	Z	$E_{\text{prev}} - (1 - E_{\text{prev}})\rho^{-h}$	1-Z
D4	$Z = \prod (1 + \rho^{-h})^{2 \cdot a_h}$	1	$Z - (Z - E_{\text{prev}})(1 + \rho^{-h})^2$	Z-1
D42	$\frac{1}{Z} = \prod (1 + \rho^{-h})^{2 \cdot a_h}$	Z	$1 - (1 - E_{\text{prev}})(1 + \rho^{-h})^2$	1-Z
D1R	$Z = Y \cdot \sum_h a_h \rho^{-h}$	0	$E_{\text{prev}} - Y \cdot \rho^{-h}$	Z
D2R	$Z = \sum_h a_h \ln(1 + \rho^{-h})$	0	$E_{\text{prev}} - \ln(1 + \rho^{-h})$	Z
D3R	$Z = \prod (1 + \rho^{-h})^{a_h}$	1	$E_{\text{prev}} - (Z - E_{\text{prev}})\rho^{-h}$	Z-1
D32R	$1 = Z \cdot \prod (1 + \rho^{-h})^{a_h}$	Z	$E_{\text{prev}} - (1 - E_{\text{prev}})\rho^{-h}$	1-Z
D4R	$Z = \prod (1 + \rho^{-h})^{2 \cdot a_h}$	1	$Z - (Z - E_{\text{prev}})(1 + \rho^{-h})^2$	Z-1
D42R	$1 = Z \cdot \prod (1 + \rho^{-h})^{2 \cdot a_h}$	Z	$1 - (1 - E_{\text{prev}})(1 + \rho^{-h})^2$	1-Z

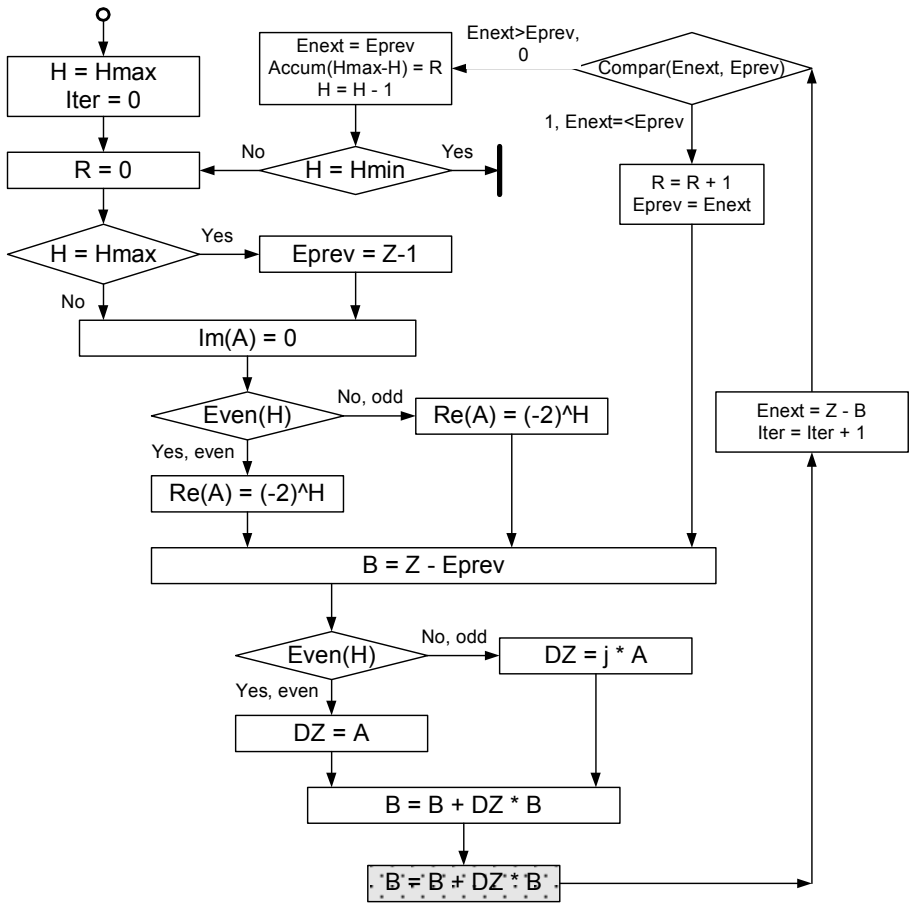


Рис. 2

Подробные алгоритмы для декомпозиций приведены далее. В этих алгоритмах приняты следующие обозначения:

Z — данное комплексное число,

E_{prev} — предыдущее значение результата,

E_{next} — следующее значение результата,

$Compar$ — функция сравнения комплексных чисел по модулю, которая выполняется компаратором комплексных чисел и возвращает значение «1», если $E_{next} \leq E_{prev}$ и «0», если $E_{next} > E_{prev}$,

R — текущее значение разряда коллектора,

$Accum$ — функция записи числа R в данный разряд коллектора,

H – текущий номер разряда, $H_{\min} \leq H \leq H_{\max}$

Even – функция проверки четности действительного целого положительного числа H (которая эквивалентна проверке значения младшего разряда кода числа),

Logar – функция выбора константы $\ln(1 + \rho^{-h})$,

DZ – приращение,

Iter – счетчик итераций,

A, B – промежуточные результаты (комплексные числа).

На рис. 2 приведена схема алгоритмов декомпозиций DecompBinom и DecompBinom2 . Выделенный на этом рисунке блок относится только к последней из этих декомпозиций. На рис. 3 приведена схема алгоритмов декомпозиций DecompBinomReal и DecompBinom2Real . Выделенный на этом рисунке блок относится только к последней из этих декомпозиций.

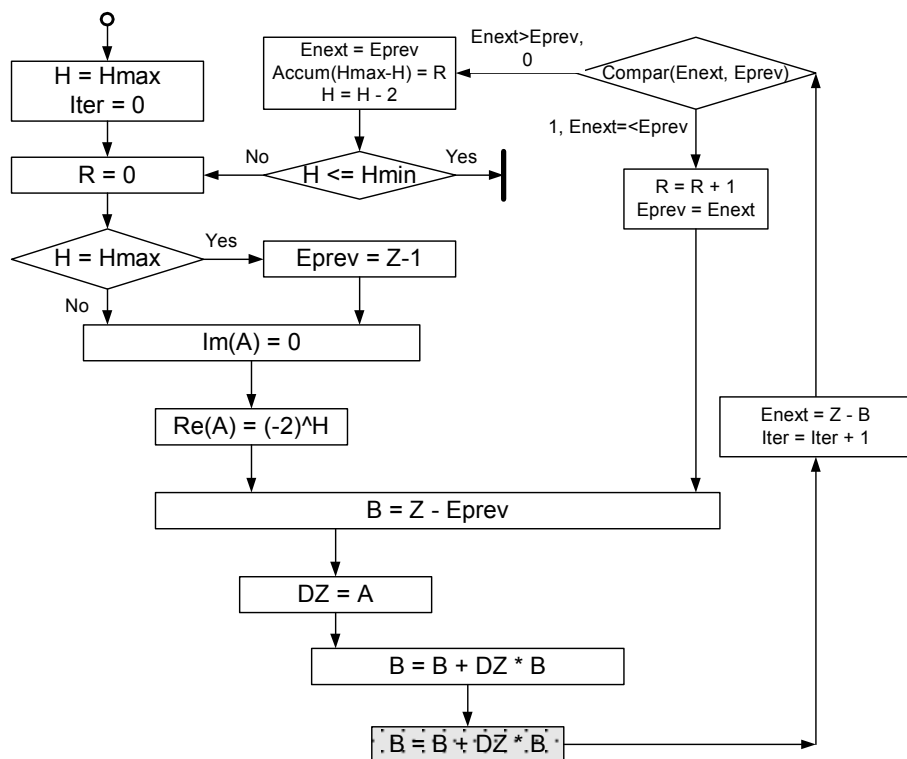


Рис. 3

На рис. 4 приведена схема алгоритмов декомпозиций `DecompBinomLogar` и `DecompBinomLogarReal`. Выделенные на этом рисунке блоки «H-1» и «H-2» относятся соответственно только к декомпозиции `DecompBinomLogar` и `DecompBinomLogarReal`.

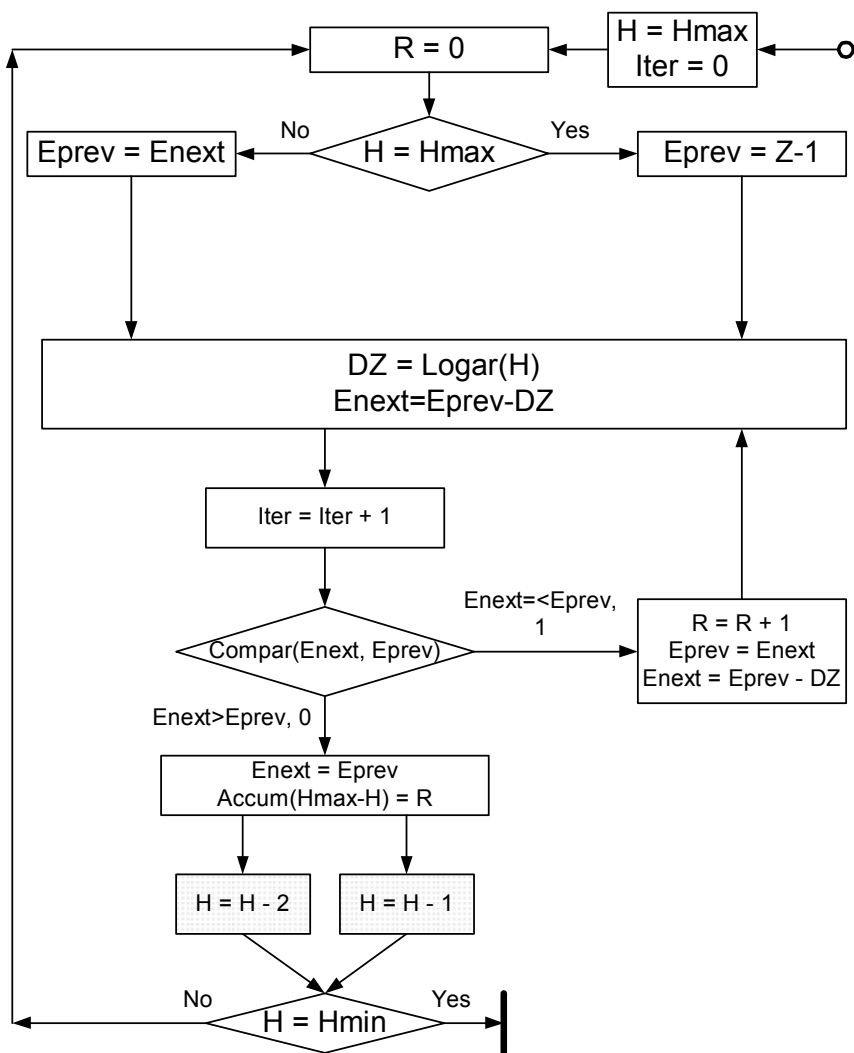


Рис. 4

6.3. Композиция

Как указывалось, результатом декомпозиции является последовательность чисел a_h . Каждый потенциальный элемент разложения может отсутствовать в конкретном разложении или присутствовать в нем, повторяясь a_h раз. Очевидно, по этой последовательности можно восстановить число, которое подвергалось декомпозиции. Такое вычисление будем называть *композицией* (compositing). Она, очевидно, является операцией, обратной декомпозиции, и заключается в вычислении комплексного числа как суммы или произведения некоторых других известных чисел - элементов разложения. В подобном контексте такое представление числа также будем называть *композицией* (composition). Важно отметить, что в процессе композиции элементы разложения могут преобразовываться или заменяться другими элементами. В этом состоит смысл композиции и в дальнейшем мы воспользуемся этим приемом.

В табл. 4 представлены варианты композиций. Композиция представляет собой итерационный процесс, где на каждом шаге известна величина a_h и выполняется изменение предыдущего значения искомого комплексного числа по формулам, приведенным в табл. 1. Каждый потенциальный элемент разложения может отсутствовать в конкретном разложении или присутствовать в нем, повторяясь a_h раз.

Подробные алгоритмы для композиций приводятся ниже. В этих алгоритмах приняты следующие обозначения (кроме обозначений, принятых в алгоритмах декомпозиции):

A — данное комплексное число; аргумент некоторых композиций (присутствует в композициях с окончанием 'A'; в других композициях $A=I$),

Unit — функция выбора константы «1» в h -разряде» или сдвига комплексного числа A на h -разрядов (при $A \triangleright 1$),

W_{prev} — предыдущее значение результата,

W_{next} — следующее значение результата,

DW — приращение,

k — счетчик значения разряда коллектора,

B — промежуточная переменная (комплексное число).

Таблица 4. Каталог композиций.

№	Обозначение	Композиция	Формула
C1	CompDivis	Композиция для деления	$W = \sum a_h \rho^{-h}$
C1A	CompDivisA	Композиция для деления	$W = \sum a_h A \rho^{-h}$
C2	CompBinom	Композиция биномов	$W = \prod (1 + \rho^{-h})^{a_h}$
C2A	CompBinomA	Композиция биномов	$W = A \cdot \prod (1 + \rho^{-h})^{a_h}$
C3	CompLogar	Композиция для логарифми-ия	$W = \sum a_h \ln(1 + \rho^{-h})$
C4	CompLogarAngle	Композиция для аргумента комплексного числа	$W = -j \cdot \text{Im} \sum a_h [\ln(1 + \rho^{-h})]$
C5	CompLogarModul	Композиция для логарифма модуля	$W = \text{Re} \sum a_h [\ln(1 + \rho^{-h})]$
C7	CompBinomConjug	Композиция для сопряженного корня квадратного	$W = \prod (1 + \tilde{\rho}^{-h})^{a_h}$
C8	CompBinomModul	Композиция для модуля	$W = \prod [(1 + \rho^{-h})(1 + \tilde{\rho}^{-h})]^{a_h}$
C8A	CompBinomModul	Композиция для модуля	$W = A \cdot \prod [(1 + \rho^{-h})(1 + \tilde{\rho}^{-h})]^{a_h}$
C9	CompBinom2	Композиция биномов в квадрате	$W = \prod (1 + \rho^{-h})^{2a_h}$
C9A	CompBinom2A	Композиция биномов в квадрате	$W = A \cdot \prod (1 + \rho^{-h})^{2a_h}$
C4 and C9A		Аргумент и модуль (одновременно)	$\arg(Z) = -2j \cdot \text{Im}(\sum \ln(1 + \rho^{-h}))$ $ Z = \prod (1 + \rho^{-h})(1 + \tilde{\rho}^{-h})$

На рис. 5 приведена схема алгоритма композиции CompBinom.

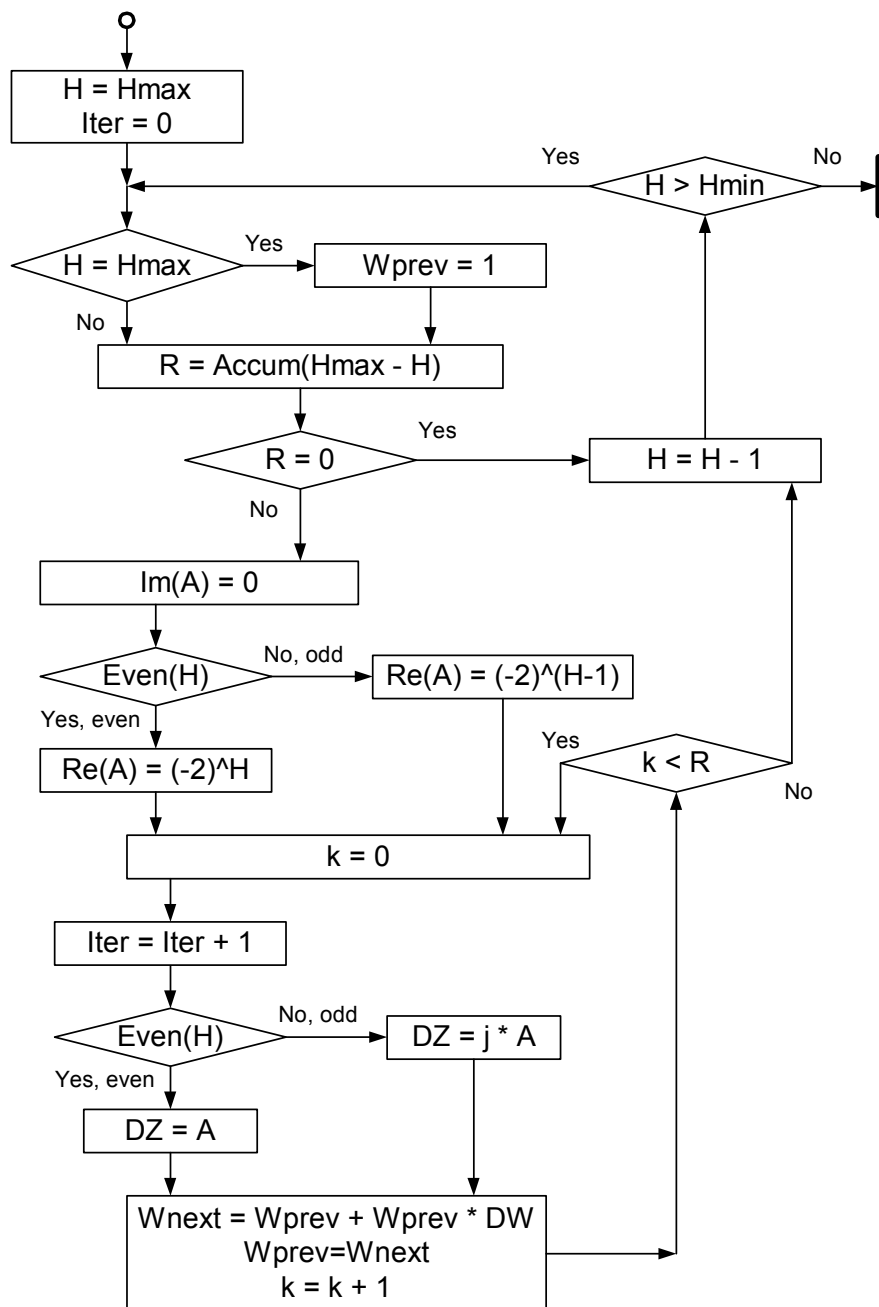


Рис. 5.

На рис. 6 приведена схема алгоритма композиции CompLogar.

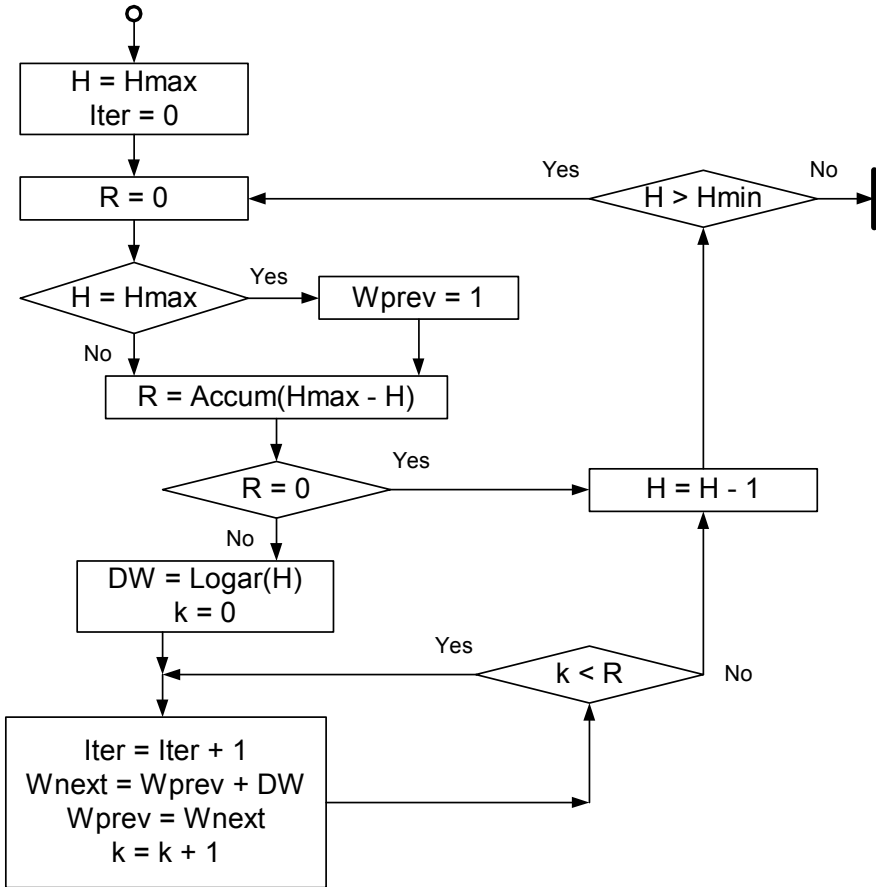


Рис. 6.

6.4. Двухшаговые операции

6.4.1. Введение

Рассмотрим теперь некоторую функцию $W = f(Z)$. Если для числа Z известна образующая последовательность $\{Z_h\}$, то может быть вычислена образующая последовательность $\{W_h\}$ числа W , ибо $W_h = f(Z_h)$. В частности, предыдущая образующая $W_{prev} = f(Z_{prev})$, а следующая образующая $W_{next} = f(Z_{next})$ или, учитывая формулу (1.2),

$$W_{next} = f\{\Phi[Z_{prev}, 1, \rho^{h+1}]\} \quad (1)$$

Часто возможно представить выражение (1) в следующем виде:

$$W_{next} = \Psi[W_{prev}, 1, \rho^{h+1}] \quad (2)$$

Таким образом, вычисление m -ой образующей W_m , представляющей функцию $W = f(Z)$ с допустимой максимальной абсолютной погрешностью Δ' , сводится к алгоритму декомпозиции и вычислениям по формуле (2). Число образующих m определяется, по-прежнему, из условия (1.4), где Z_m - m -ая образующая аргумента, имеющая абсолютную погрешность Δ . Необходимая величина погрешности Δ определяется величиной погрешности Δ' и видом функции $W = f(Z)$.

6.4.2. Алгоритм вычисления функции

Итак, вычисление значения функции с допустимой погрешностью может быть произведено по алгоритму, который в основном совпадает с алгоритмом декомпозиции и описывает h -ый цикл вычисления. Отличие заключается только в том, что вначале известна еще величина W_{prev} , и п.5 алгоритма имеет следующий вид:

5. Определяется значение предыдущих образующих Z'_{prev} и

W'_{prev} для следующего цикла, где

$$Z'_{prev} = \begin{cases} Z_{prev} & \text{if } H_{next} > H_{prev} \\ Z_{next} & \text{if } H_{next} \leq H_{prev} \end{cases} \text{ и}$$

$$W'_{prev} = \begin{cases} W_{prev} & \text{if } H_{next} > H_{prev} \\ W_{next} & \text{if } H_{next} \leq H_{prev} \end{cases}. \text{ При этом } W_{next}$$

вычисляется (если это необходимо) по формуле (2).

Пункты 4, 5, 6 выполняются согласно табл. 2 (со столбцом W'_{prev}).

Более наглядно последовательность элементарных операций этого алгоритма изображена на рис. 2 (с выделенным блоком).

Целесообразность вычисления функции по этому алгоритму полностью определяется сложностью вычислений по формулам (1) и (2), т.е. видом функции $W = f(Z)$. В том случае, если вычисления по формулам (1) и (2) сводятся к таким операциям, которые просто реализуются на цифровых схемах, применение метода «цифра за цифрой» для вычисления функции $W = f(Z)$ имеет смысл. Этот метод может быть реализован как программным, так и аппаратным способом. В первом случае набор операций процессора должен содержать операцию сравнения размеров кодов, после которой вырабатываются сигналы условного перехода по тому или иному направлению в программе вычислений. Аппаратный способ может быть использован, если вычисления по формулам (1) и (2) производятся лишь при помощи так называемых «коротких» или элементарных машинных операций: сдвига, сложения, вычитания.

6.4.3. Об аппаратной реализации

Аппаратная реализация указанного алгоритма состоит в разработке соответствующего конечного автомата. Как правило, целесообразно выполнять этот алгоритм в два этапа:

- 1) декомпозиция аргумента Z , т.е. вычисление последовательности чисел a_h ,
- 2) композиция, т.е. итеративное вычисление функции $W = f(Z)$ по формуле $W_{next} = \Psi[W_{prev}, a_h, \rho^{h+1}]$.

В табл. 5 перечислены такие операции. В табл. 6 указана связь между функциями, композициями и декомпозициями.

При использовании коллектора аппаратное вычисление некоторых функций $W = f(Z)$ состоит из двух шагов:

1. декомпозиция числа Z и заполнение коллектора,
2. композиция числа W при данном заполнении коллектора.

В табл. 5 перечислены такие операции. В табл. 6 указана связь между функциями, композициями и декомпозициями.

Таблица 5. Каталог двухшаговых операций.

Операция	D	C
$ Z = \prod (1 + \rho^{-h}) (1 + \tilde{\rho}^{-h})$	D4	C8
$\sqrt{Z} = \prod (1 + \rho^{-h})$	D4	C2
$\sqrt{\tilde{Z}} = \prod (1 + \tilde{\rho}^{-h})$	D4	C7
$e^Z = \prod (1 + \rho^{-h})$	D2	C2
$\ln(Z) = \sum \ln(1 + \rho^{-h})$	D3	C3
$\ln(Z) = 2 \cdot \sum \ln(1 + \rho^{-h})$	D4	C3
$\arg(Z) = -j \cdot \operatorname{Im} \sum \ln(1 + \rho^{-h})$	D3	C4
$\arg(Z) = -2j \cdot \operatorname{Im} \sum \ln(1 + \rho^{-h})$	D4	C4
$\ln Z = \operatorname{Re} \sum \ln(1 + \rho^{-h})$	D3	C5
$\frac{1}{Z} = \sum (\rho^{-h})$	D1	C1
$Z = Z \cdot e^{j\varphi} = Z \cdot \prod (1 + \rho^{-h})$	D2	C2A
$\arg(Z) = -2j \cdot \operatorname{Im} \left(\sum \ln(1 + \rho^{-h}) \right),$ $ Z = \prod (1 + \rho^{-h}) (1 + \tilde{\rho}^{-h})$	D4	C4 and C8

Таблица 6.

	D1	D2	D3	D4
C1	Division			
C2		Exponentiation	Identically	Square-rooting
C3		Identically	Logarithm	
C4			Argument	
C5			Modulus Logarithm	
C7				Conjugated Square-rooting
C8				Modulus
C9				Identically

Далее описываются схемы некоторых специализированных устройств, используемых при аппаратной реализации данного метода.

6.5. Схемы и операции с биномами

6.5.1 Схемы умножения на бином

Схема умножения на бином зависит от значения величины N – см. программу MultBinom. Пусть $N=4*k+m$, где $m=\{0, 1, 2, 3\}$. На фиг. 0, 1, 2, 3 показаны соответствующие схемы умножения на бином. На этих фигурах обозначено:

Shifter - сдвигатель вправо; рядом с управляющим сигналом указано количество разрядов, на которые производится сдвиг,

Adder - сумматор со входами 1 и 2; в скобках указан вид операции, т.е. знак, который присваивается каждому слагаемому перед сложением;

a - исходное число,

b - результат умножения на бином.

Таким образом, умножение на бином распадается на сдвиг, алгебраическое сложение и (возможно) перестановку частей комплексного числа (рокировку). В табл. 1 перечислены эти действия в зависимости от значения N .

Таблица 1.

N	Сдвиг на S разрядов	Алг. сложение реальной части	Алг. сложение мнимой части	Рокировка
$N=4*k+0$	$S=(-N)/2$	$1+2$	$1+2$	no
$N=4*k+1$	$S=(-N+1)/2$	$1+2$	$-1-2$	yes
$N=4*k+2$	$S=(-N)/2$	$1-2$	$1-2$	no
$N=4*k+3$	$S=(-N+1)/2$	$1-2$	$-1+2$	yes

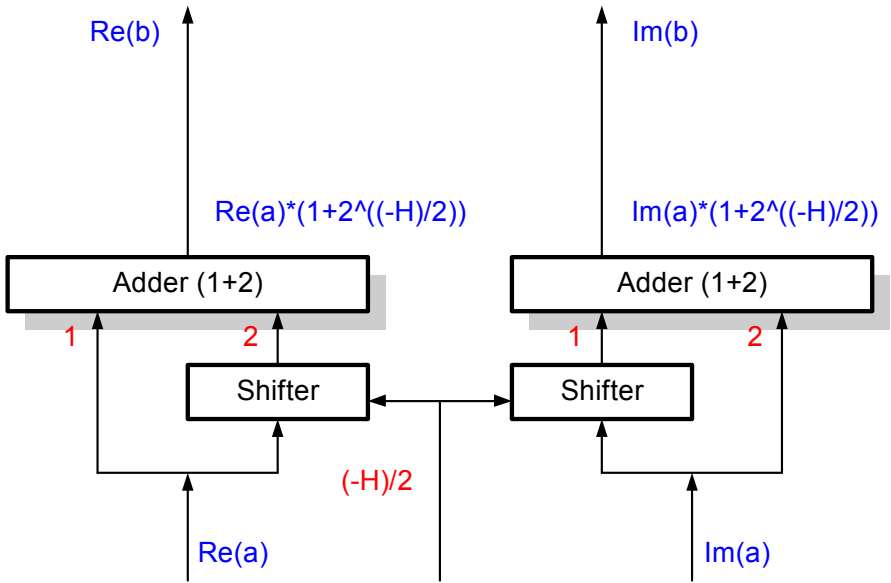


Fig. 0. $H=4 \cdot k+0$

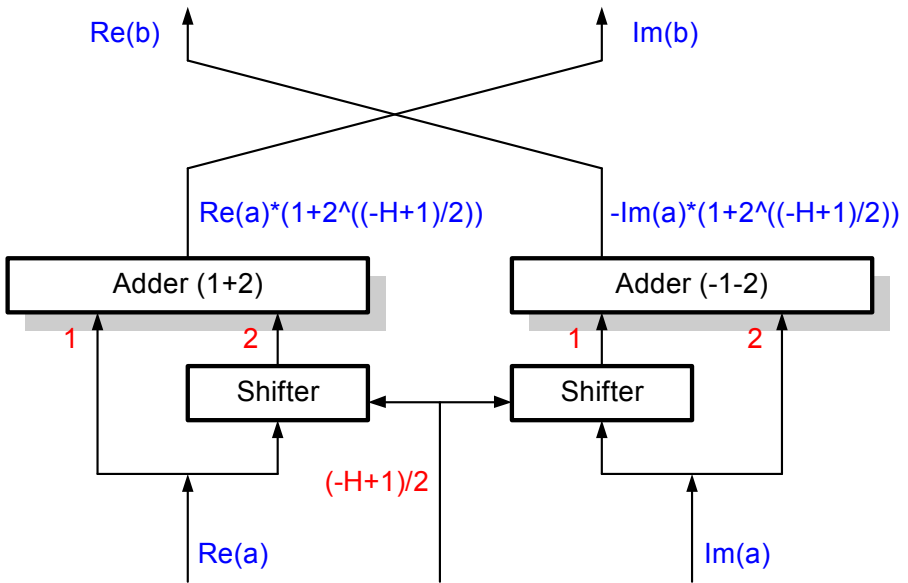


Fig. 1. $H=4 \cdot k+1$

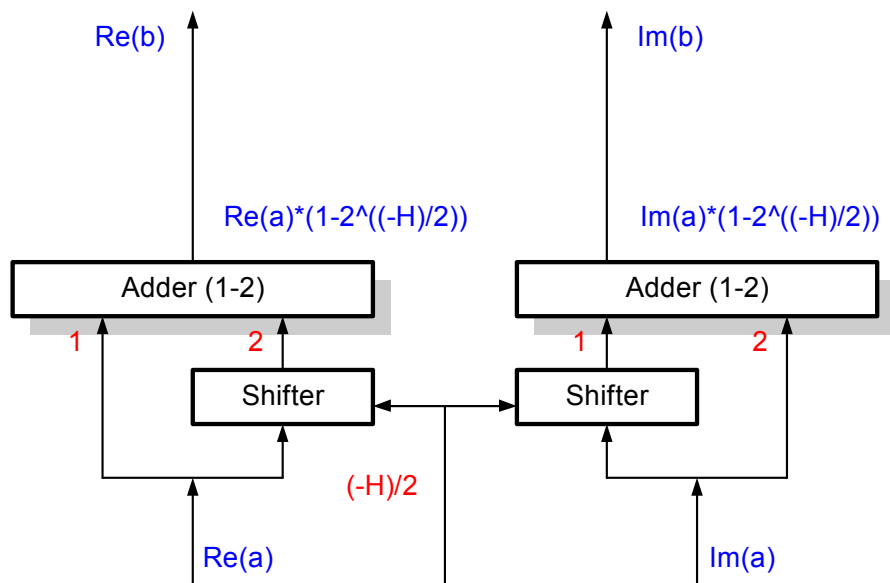


Fig. 2. $H=4*k+2$

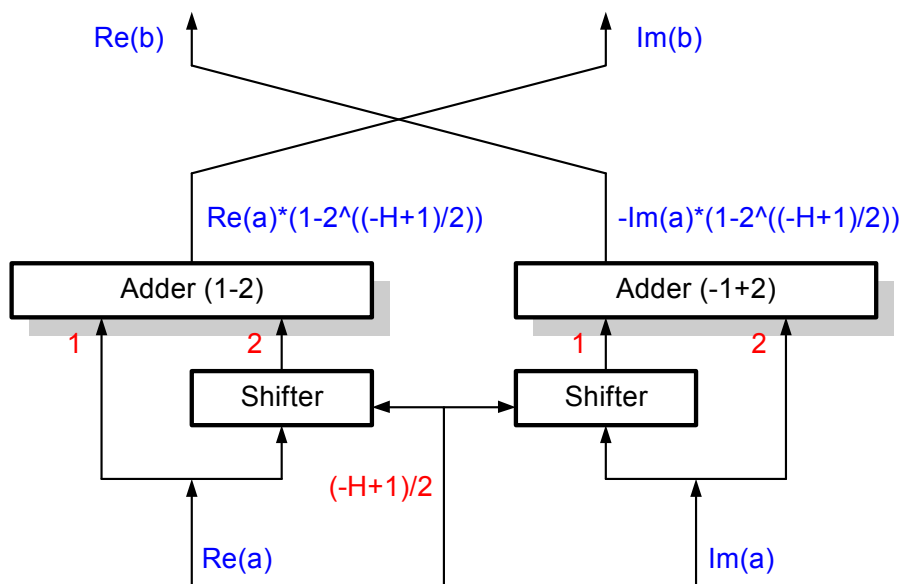


Fig. 3. $H=4*k+3$

6.5.2. Операции с биномами

В этих операциях одним из операндов является целое число H , представленное в традиционном коде. Операции прежде всего вычисляют определенную функцию от H . Экспоненты в операции не участвуют. Существуют следующие функции:

- H -бином $(1 + \rho^{-H})$, где $\rho^H = \begin{cases} (-2)^{H/2} & \text{if } H - \text{even} \\ j \cdot (-2)^{(H-1)/2} & \text{if } H - \text{odd} \end{cases}$,
- сопряженный H -бином $(1 + \tilde{\rho}^{-H})$, где $\tilde{\rho}^H = \begin{cases} (-2)^{H/2} & \text{if } H - \text{even} \\ -j \cdot (-2)^{(H-1)/2} & \text{if } H - \text{odd} \end{cases}$,
- натуральный логарифм H -бинома $\ln(1 + \rho^{-H})$.

Применяются следующие операции:

- Сложение с логарифмом H -бинома - AddLogarBinomH
- Вычитание логарифма H -бинома - SubLogarBinomH
- Умножение на H -бином - MultBinomH
- Умножение на сопряженный H -бином - MultConjugBinomH

Логарифмы бинома хранятся в постоянном запоминающем устройстве, включенным в состав арифметического устройства.

Умножение на бином состоит из сдвига и сложения. При ограниченной разрядности данных (N - количество разрядов мантиссы).

- бином $(1 + \rho^{-h}) = 1$ при $(-h) \geq N$ и умножение на бином исключается,
- бином $(1 + \rho^{-h}) = \rho^{-h}$ при $(-h) < N$ и умножение на бином заменяется сдвигом.

6.6. Коллектор

6.6.1. Устройство коллектора

Выше во многих операциях использовался *многозначный код*. Такой код состоит из разрядов, каждый из которых может принимать значения от 0 до 7. Для хранения и операций с таким кодом используется *коллектор*.

Коллектор представляет собой **N** трехразрядных счетчиков. Одноименные разряды счетчиков объединены в регистры, т.е. коллектор содержит 3 регистра **Acc1**, **Acc2**, **Acc3**. Коллектор может рассматриваться либо как множество трехразрядных счетчиков, либо как тройка регистров. Тройка одноименных **N**-разрядов этих регистров называется далее **N-триадой** или **N-разрядом** коллектора или **N-счетчиком**. С каждым **N**-счетчиком возможны следующие операции:

- Запись «0» по сигналу **AccRes**;
- Прибавление «1» по сигналу **AccPlus**; при этом может возникнуть и должен быть выдан сигнал переполнения **AccLimit**, а счетчик должен остаться в состоянии **AccMax**;
- Вычитание «1» по сигналу **AccMinus**; этот сигнал может возникнуть только при ненулевом состоянии счетчика;
- Считывание по сигналу **AccRead**.

Разряды коллектора нумеруются от 0 до $(N-1)$. С коллектором в целом возможны следующие операции поиска ненулевого **N**-разряда коллектора с выдачей его номера **H**:

- Поиск первого ненулевого **N**-разряда коллектора (поиск начинается с 0-разряда);
- Поиск следующего ненулевого **N**-разряда коллектора (поиск начинается после данного **P**-разряда);

На рис. 1 представлена схема коллектора. При этом приняты следующие обозначения:

AccCalc - вычислитель коллектора,

AccFind - блок поиска триады коллектора,

AccDec - декодер коллектора,

Mux – управляющий мультиплексор,

MuxNx3 – мультиплексор триад,

RegAcc - блок регистров коллектора,

AccMax – регистр максимально-допустимого значения триады,

AccResult – регистр вычисленного значения триады.

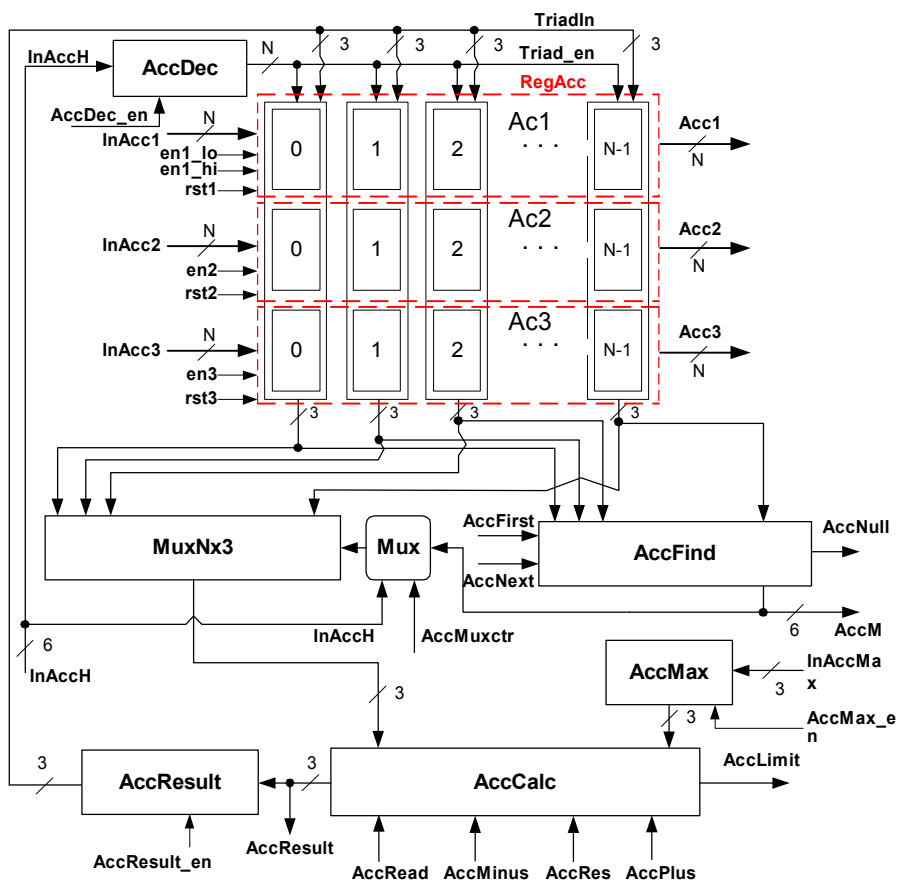


Рис. 1. Коллектор

Перечислим *входные* сигналы коллекора:

- InAcc1, InAcc2, InAcc3 – коды на входе регистров (N разрядов),
- AccFirst - сигнал поиска первой значащей триады,
- AccNext – сигнал поиска следующей значащей триады,
- InAccH – номер триады (6 бит),
- AccRes, AccRead, AccPlus, AccMinus – сигналы операций,
- InAccMax – максимальное значение триады (3 бита),
- AccDec_en – сигнал разрешения записи в триаду,
- AccMuxctr – сигнал разрешения чтения триады,
- AccMax_en - сигнал разрешения записи в регистр AccMax,

AccResult_en – сигнал разрешения записи в регистр AccResult,
en1_lo – сигнал разрешения записи в регистр Ac1,
en1_hi – сигнал разрешения записи в регистр Ac1,
en2, en3 – сигналы разрешения записи в регистры Ac2, Ac3,
rst1, rst2, rst3 – сигналы обнуления регистров Ac1, Ac2, Ac3.

Перечислим *выходные* сигналы коллектора:

AccLimit – сигнал, свидетельствующий о том, что значение триады превышает AccMax,
AccNull – сигнал, свидетельствующий о том, что значащая триада не найдена,
AccM – номер найденной значащей триады (6 бит),
AccResult – вычисленное значение триады (3 бита),
Acc1, Acc2, Acc3 – выход регистров Ac1, Ac2, Ac3 (N разрядов).

Далее рассматриваются блоки, составляющие коллектор.

6.6.2. Блок поиска триады – AccFind.

Этот блок входит в состав коллектора и осуществляет поиск ненулевого N -разряда коллектора с выдачей его номера N :

- Поиск первого ненулевого N -разряда коллектора (поиск начинается с 0-разряда);
- Поиск следующего ненулевого N -разряда коллектора (поиск начинается после данного P -разряда);

Схема блока представлена на рис. 2. При этом приняты следующие обозначения:

AccLogic – логический блок коллектора; он может быть выполнен в двух вариантах – AccLogic или AccLogic3 (см. ниже),

OrTriad – N -элементный блок OR,

RegOne – регистр идентификаторов коллектора, в нем отмечаются найденные триады (установкой ‘1’ в соответствующем разряде).

Перечислим *входные* сигналы коллектора:

Acc1, Acc2, Acc3 – выходы регистров коллектора (N разрядов),
AccFirst – сигнал поиска старшей значащей триады,
AccNext – сигнал поиска следующей значащей триады,

Перечислим *выходные* сигналы коллектора:

AccNull – сигнал отсутствия найденной триады,
AccM – номер найденной триады (6 бит).

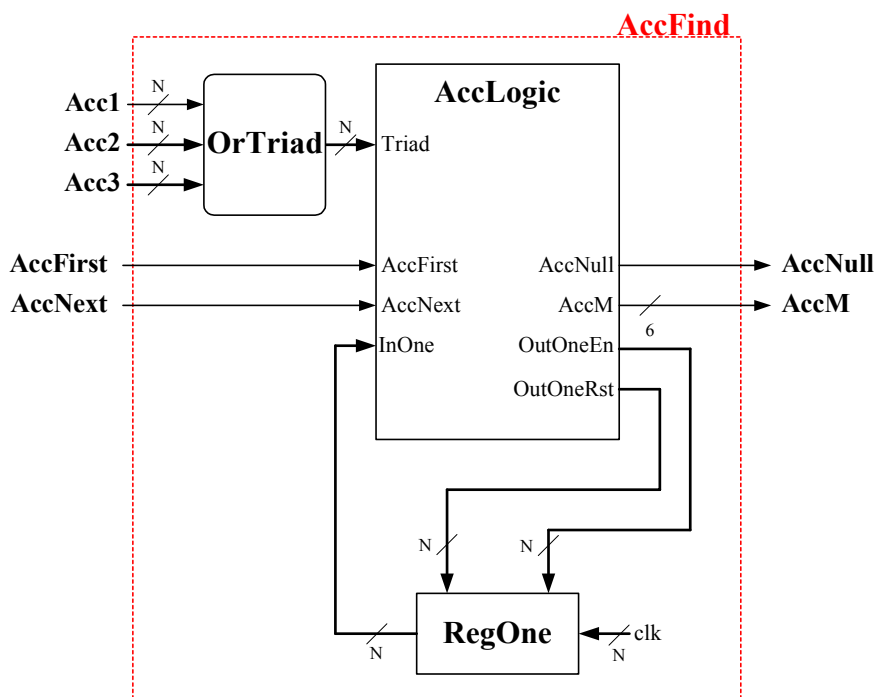


Рис. 2. Блок поиска триады коллектора

6.6.3. Логический блок – AccLogic.

На рис. 3 представлена схема блока, состоящая из последовательно включенных одноразрядных схем AccLog. При этом приняты следующие обозначения:

выходные сигналы:

Triad – триады (N триад),

AccFirst - сигнал поиска первой значащей триады,

AccNext - сигнал поиска следующей значащей триады,

InOne – выход регистра OutOne (N разрядов).

выходные сигналы:

AccNull - сигнал, свидетельствующий о том, что значащая триада не найдена,

AccM - номер найденной значащей триады (6 бит),

OutOneEn – сигнал установки в ‘1’ одного из разрядов регистра OutOne,
 OutOneRst – сигнал установки в ‘0’ регистра OutOne.

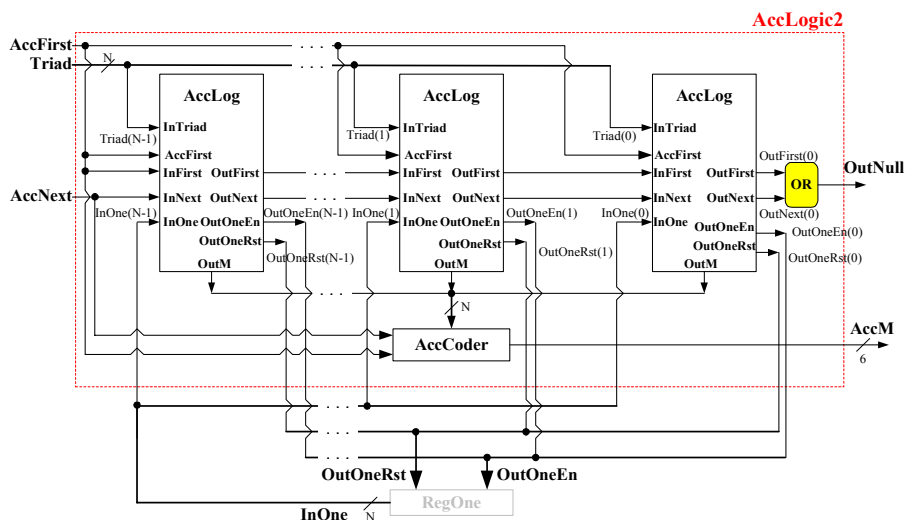


Рис. 3. Логический блок коллектора

Этот блок ищет младшую ненулевую триаду среди поступивших на вход Triad. Номер найденной триады выдается в AccM. При отсутствии такой триады выдается сигнал AccNull. В регистре OutOne все разряды, соответствующим пройденным ненулевым триадам устанавливаются в ‘0’ (сигналом OutOneRst), а разряд соответствующий найденной триаде, устанавливается в ‘1’ (сигналом OutOneEn).

Если поиск начинается по сигналу AccFind=1, то анализ начинается с 0-триады.

Если поиск начинается по сигналу AccNext=1, то сначала отыскивается младший единичный разряд в регистре OutOne. Он соответствует триаде, на которой завершился предыдущий поиск. Анализ продолжается со следующей триады.

6.6.4. Ускоренный логический блок – AccLogic3.

Этот блок отличается от блока AccLogic тем, что в нем приняты меры для ускорения поиска. Для этого в нем применена трехразрядная логическая схема AccLogGrp3, анализирующая одновременно три триады. На рис. 4 представлен блок AccLogic3.

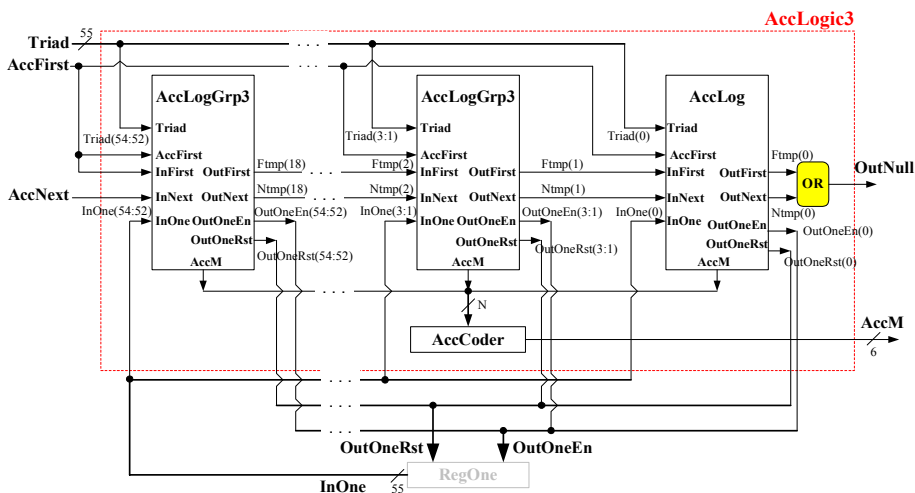


Рис. 4. Ускоренный логический блок коллектора

6.6.5. Трехразрядная логическая схема – AccLogGrp3.

Эта схема используется в ускоренном логическом блоке AccLogic3. Схема, в свою очередь, содержит три одноразрядных схемы AccLog. На рис. 5 представлена схема AccLogGrp3. При этом приняты следующие обозначения:

входные сигналы:

Triad, InOne – трехразрядные входы (3 бита),

AccFirst, InFirst, InNext – одноразрядные входы.

выходные сигналы:

AccM, OutOneEn, OutOneRst - трехразрядные выходы (3 бита),

OutFirst, OutNext – одноразрядные выходы.

Схема Block1 реализует следующее правило:

```

If ((InFirst = '1' and Triad = "000") or (InNext = '1' and
    ((InOne(2) = '1' and Triad(1) = '0' and Triad(0) = '0') or
    (InOne(1) = '1' and Triad(0) = '0') or (InOne(0) = '1'))))
    then OutFirst = '1'

```

```
Else OutFirst = '0'
```

Схема Block2 реализует следующее правило:

```
If (InNext = '1' and InOne = "000") then OutNext = '1'
Else OutNext = '0'
```

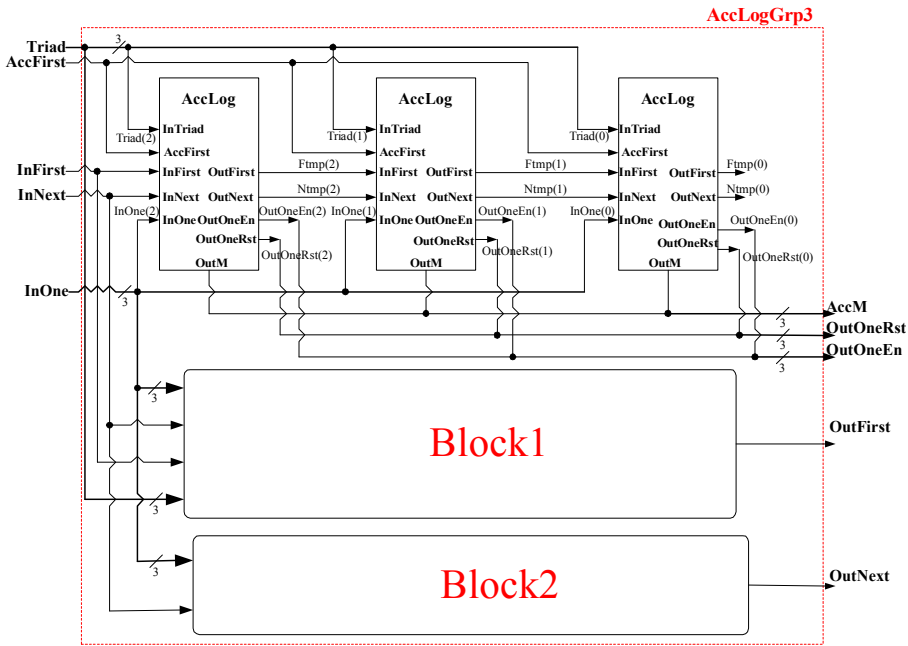


Рис. 5. Трехразрядная логическая схема.

6.6.6. Одноразрядная логическая схема – **AccLog**.

В этой схеме используются следующие сигналы:

входные сигналы:

AccFirst, **InTriad**, **InFirst**, **InNext**, **InOne**.

выходные сигналы:

OutFirst, **OutNext**, **OutOneEn**, **OutOneRst**, **OutM**.

Таблица 1.

InFirst	InNext	InTriad	InOne	OutFirst	OutNext	OutM	OutOneEn	OneRst
0	0	-	-	0	0	0	0	0
1	0	0	-	1	0	0	0	1
1	0	1	-	0	0	1	1	0
0	1	-	0	0	1	0	0	0
0	1	-	1	1	0	0	0	1

Внутри блока используется некоторый «промежуточный» сигнал обнуления **OneRst**. Таблица 1 истинности блока связывает входные сигналы **InTriad**, **InFirst**, **InNext**, **InOne** с выходными сигналами **OutFirst**, **OutNext**, **OutOneEn**, **OutM** и промежуточным сигналом **OneRst**. Выходной сигнал **OutOneRst** определяется по следующему правилу:

If OneRst = '1' or (AccFirst = '1' and OutOneEn = '0') then
 OutOneRst = '1'
 Else OutOneRst = '0'

6.6.7. Операции с коллектором.

6.6.7.1. Чтение H -разряда коллектора - ReadAccH

В этой операции значение H -разряда коллектора записывается в непосредственном виде (без преобразования) в младшие разряды мантиссы результата. Число H в Р-коде принимается из операнда.

6.6.7.2. Обнуление H -разряда коллектора - RstAccH

В этой операции значение H -разряда коллектора обнуляется. Число H в Р-коде принимается из операнда.

6.6.7.3. Добавление "1" в H -разряд коллектора - AddOneToAccH

В этой операции значение H -разряда коллектора увеличивается на 1. Если при этом результат становится больше числа AccMax, то выдается сигнал переполнения, а значение H -разряда не изменяется. Число H в Р-коде принимается из первого операнда.

6.6.7.4. Поиск в первого элемента коллекторе - ReadFirstAcc

В этой операции последовательно анализируются значения разрядов коллектора. Анализ начинается с ($K=0$)-разряда и продолжается в сторону увеличения K . При обнаружении некоторого M -разряда с ненулевым значением это значение и номер M выдаются. При этом значение M -разряда записывается в непосредственном виде (без преобразования) в младшие разряды мантиссы результата, а номер M в Р-коде записывается в экспоненту результата. При отсутствии значащих разрядов в коллекторе значение и номер M принимают нулевое значение.

6.6.7.5. Поиск в следующего элемента коллекторе - ReadNextAcc

Эта операция выполняется аналогично предыдущей, но анализ начинается с ($K=H+1$)-разряда и продолжается в сторону увеличения K . Число H в Р-коде принимается из операнда.

6.6.7.6. Установка максимума в счетчике коллектора - SetAccMax

В этой операции устанавливается значение счетчика AccMax в коллекторе. Это значение в Р-коде принимается из экспоненты операнда.

Глава 7. Деление

7.1. Метод деления

Деление позиционных кодов действительных чисел по отрицательному основанию и деление позиционных кодов комплексных чисел во многом аналогично обычному делению и состоит из циклов ‘сдвиг-вычитание-сравнение’. Отличие состоит только в правилах выполнения сравнения. Для действительных положительных чисел сравнение производится по предыдущему и последующему знаку остатка. Для кодов действительных чисел по отрицательному основанию знак остатка может быть определен по четности/нечетности номера старшего значащего разряда. Соответственно сравнение знаков остатков можно выполнить сравнением четности старших значащих разрядов. В случае комплексных чисел этот метод не годится, поскольку не приходится говорить о знаке комплексного числа. Сравнение по знаку можно было бы заменить сравнением по модулю. Для действительных чисел это одно и то же, ибо сохранение знака остатка свидетельствует об уменьшении его модуля.

По-существу, деление выполняется по описанному выше методу «цифра-за-цифрой».

Критерием правильности очередного вычитания служит сохранение или уменьшение размера кода. Ниже рассматривается алгоритм деления, основанный на сравнении длин двоичных кодов и пригодный для деления как действительных, так и комплексных чисел.

Итак, необходимо найти частное $Z = V / W$ от деления чисел - делимого и делителя, представленных своими двоичными позиционными кодами $\langle V \rangle$ и $\langle W \rangle$ соответственно.

Рассмотрим алгоритм деления, обозначив

P - предыдущий остаток

S - следующий остаток

v, w, p, s - размеры кодов V, W, P, S соответственно

Алгоритм 1.

1. Выполняется сдвиг делителя W на h разрядов влево при $h > 0$ или вправо при $h < 0$, где $h = v - w$.
2. Предыдущему остатку P присваивается значение делимого V , а частному Z - значение 0.
3. Сдвинутый делитель W вычитается из предыдущего остатка P . Разность является следующим остатком S .
4. Сравниваются размеры p и s с целью определения нового значения частного, сдвинутого делителя и величины остатка для следующего цикла деления. Эти данные определяются в соответствии с таблицей 1.
5. Выполняется переход к пункту 3 с новыми значениями сдвинутого делителя и предыдущего остатка.

Таблица 1

Условие	Результат сравнения		
	Частное	h	Предыдущий остаток для следующего цикла
$s < p$	увеличивается на единицу h -го разряда	увеличивается на единицу	$P = S$
$s = p$	увеличивается на единицу h -го разряда	остается без изменений	$P = S$
$s > p$	остается без изменений	увеличивается на единицу	$P = P$

Деление производится до получения заданного числа разрядов в частном, которое образуется суммированием единиц h -го разряда, полученных в пункте 4. Важно отметить, что случай ' $s = p$ ' может повторяться, то-есть делитель может вычитаться более одного раза, не сдвигаясь.

Пример 1. Деление кодов комплексных чисел в системе

$\langle \rho = j\sqrt{2}, \{0, 1\} \rangle$ - см. теорему 1.8. Пусть $\langle V \rangle = 1001$ и $\langle W \rangle = 1010$.

Воспользуемся алгоритмом 1.

	1001	
$h=0$:	<u>1010</u>	
	1011	$s=p, Z=1$
$h=0$:	<u>1010</u>	
	00010 \Rightarrow	$s < p, Z=10100$
$h=-1$:	<u>01010</u>	$s > p, Z$ не изменяется
	101000	
	0000100 \Leftarrow	
$h=-2$:	<u>0001010</u>	
	0000110 \Rightarrow	$s=p, Z=10100.01$
$h=-2$:	<u>0001010</u>	
	0101100	$s > p, Z$ не изменяется
	00001100 \Leftarrow	
$h=-3$:	<u>00001010</u>	
	00001110	$s=p, Z=10100.011$
$h=-3$:	<u>00001010</u>	
	00000100	$s < p, Z=10110.110$
$h=-3$:	<u>00001010</u>	
	00000110	$s=p, Z=10110.111$

и т. д. При прекращении деления на данном шаге получаем: $(1001-0.11):1010 = 10110.11$. Правильность деления легко проверить: $1001 - 0.11 = 11110.11$; $10110.111 * 1010 = 11110.11$.

Номер старшего значащего разряда в общем случае не может быть использован как размер кода, поскольку убывание такого размера не всегда соответствует убыванию модуля комплексного числа. Все дальнейшее изложение ведется в предположении, что размер кода – это модуль числа.

7.2. Деление комплексных чисел в системе с основанием ρ_2

Рассмотрим подробнее деление кодов комплексных чисел в системе с основанием ρ_2 . Программы для этого раздела находятся в каталоге CD/Devis.

Вначале рассмотрим операции декомпозиции и композиции.

Декомпозиция для деления состоит в разложении данного *нормализованного*

кода комплексного числа Z по формуле $(-2)^{-4} = \sum_h [Za_h \rho_2(h)]$.

Композиция состоит в вычислении по формуле $\frac{(-2)^{-4}}{Z} = \sum_h [a_h \rho_2(h)]$.

При аппаратной реализации этих операций используются сумматор, сдвигатель, компаратор (блок сравнения по модулю) и коллектор (см. также часть 3). При определенном выборе соотношения между модулями делителя и делимого количество вычитаний не превышает 2. Поэтому коллектор представляет собой два регистра для накопления разрядов частного, которые мы будем обозначать как Acc1 и Acc2.

Схема алгоритма декомпозиции DecompDevis представлена на рис. 1, где следующие обозначения:

Z — данное комплексное число,

R — текущее значение разряда коллектора,

Accum — функция записи в данный разряд регистров частного; эта функция устанавливает текущие разряды регистров Acc1 и Acc2 по следующему правилу:

R	Acc1	Acc2
0	0	0
1	1	0
2	0	1

H — текущий номер разряда, $H_{\min} \leq H \leq H_{\max}$

Even — функция проверки четности действительного целого положительного числа H (которая эквивалентна проверке значения младшего разряда кода числа),

Eprev — предыдущее значение результата,

Enext — следующее значение результата,

Compar— функция сравнения комплексных чисел по модулю, которая выполняется компаратором и возвращает значение «1», если $E_{next} < E_{prev}$ и «0», если $E_{next} > E_{prev}$,

DZ— приращение,

Iter— счетчик итераций,

A — промежуточный результат (комплексное число).

Важно отметить, что при декомпозиции число R не превышает 2, т.е. $a_h = \{0, 1, 2\}$. Поэтому композиция CompDevis эквивалентна сложению кодов, полученных в двух регистрах Acc1 и Acc2, как кодов по основанию ρ_2 , т.е. выполняется по формуле $W = \langle Acc1 \rangle - 2 \cdot \langle Acc2 \rangle$.

Декомпозиция и композиция могут быть совмещены. На рис. 2 представлена схема совмещенного алгоритма, где

W_{prev} — предыдущее значение результата,

W_{next} — следующее значение результата.

Теперь рассмотрим обращение числа Z , которое состоит в следующем:

1. Декомпозиция DecompDivis.

2. Композиция CompDivis.

При этом определяется число $\frac{1}{16Z}$.

Наконец рассмотрим деление чисел $V = \frac{Y}{Z}$. Пусть делитель

$Z = (-2)^k \cdot M$, где M - мантисса, k - экспонента. Деление целесообразно выполнять по следующему алгоритму:

1. Определение числа $W = \frac{1}{16 \cdot M}$, т.е. обращение числа M .

2. Умножение $V = (-2)^{-k+4} Y \cdot W$.

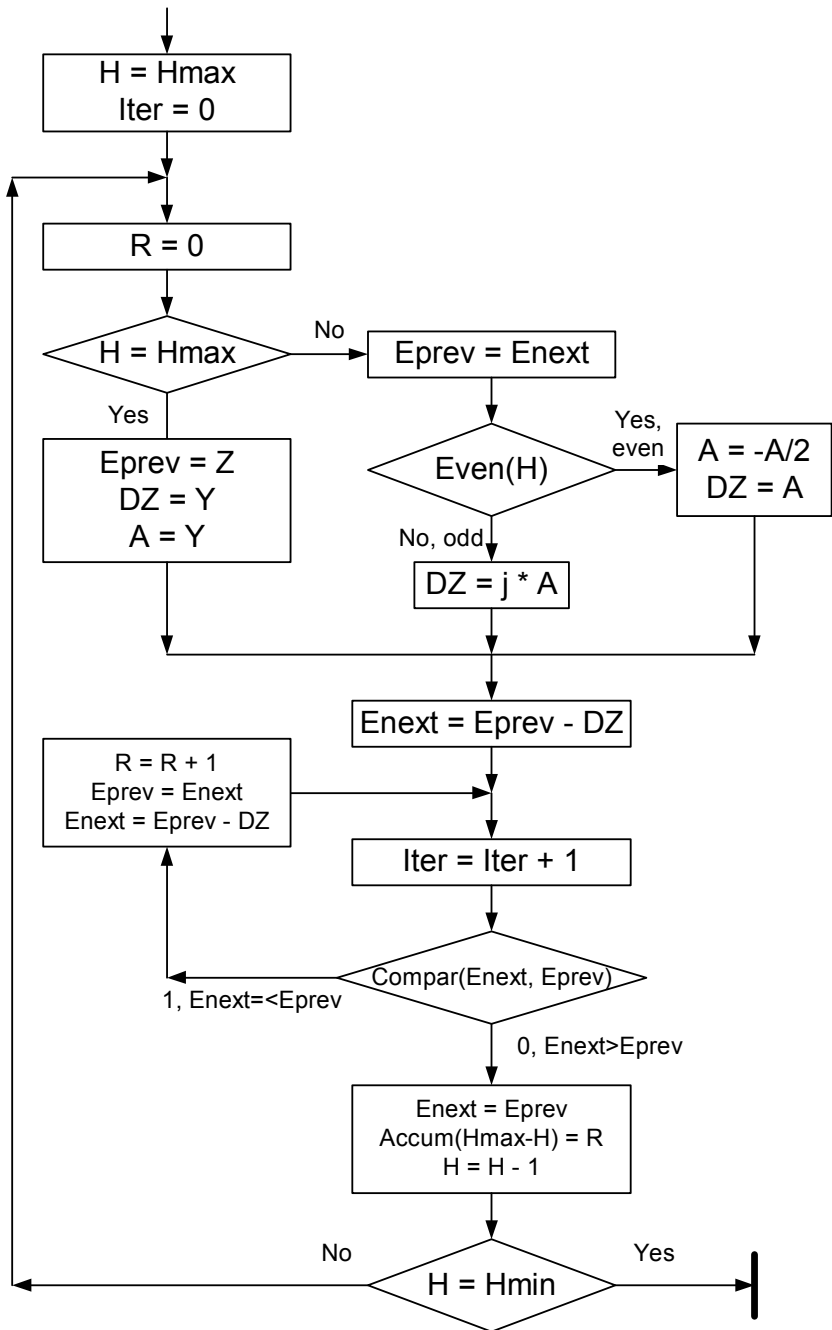


Рис. 1.

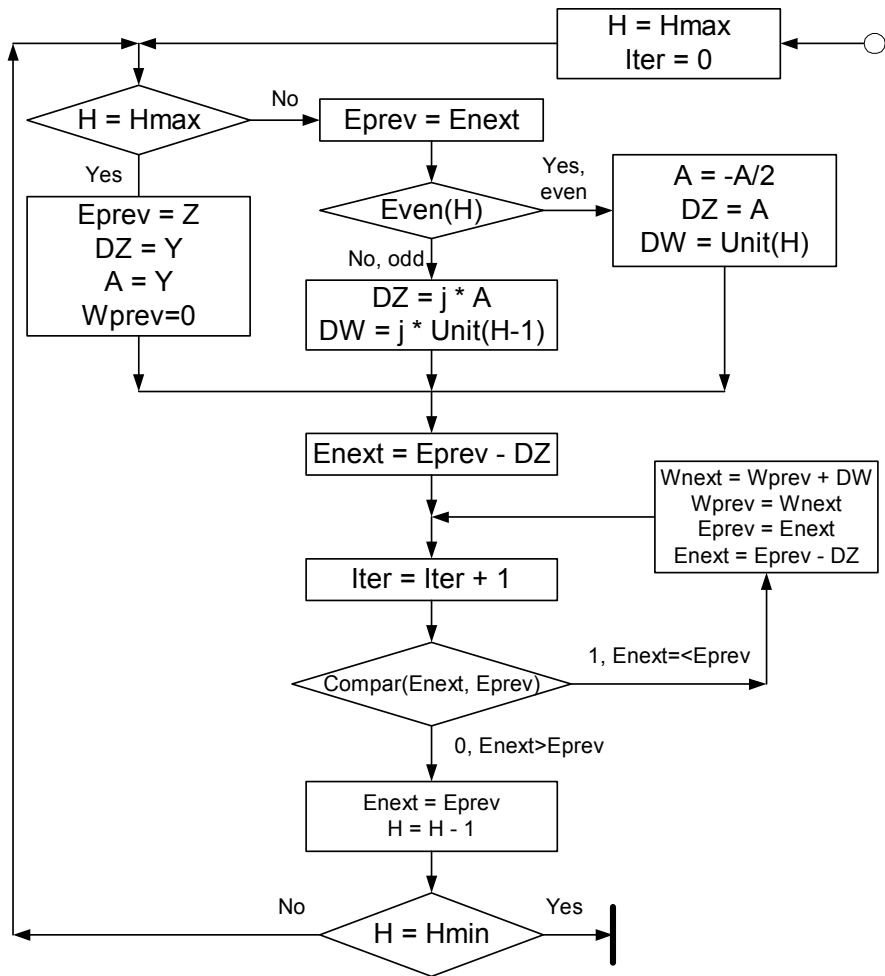


Рис. 2

Пример 1. Программы MATLAB для деления комплексных чисел в системе кодирования по основанию ρ_2 .

function [DZ]=Dev(H)

```

% функция вычисляет величину DZ - см. выше
if H == 2 * round(H / 2) % shift to the right on (H)
    DZ = (-2)^(-H/2);
else % shift to the right on (H-1)
    % and multiplication on (j)
    DZ = j * (-2)^((-H-1)/2);
end

```

```

function
[Iter,invZ,Accum,DM,Enext,IterPlus,MRD]=DecDev1(Z,Hmin,H
max)
% функция реализует алгоритм обращения числа Z,
    приведенный на рис. 2.
% MRD - модуль остатка при декомпозиции
% DM - модуль разности модулей остатка
Accum=zeros(1,Hmax+1);
hh=1-Hmin;
Iter = 1; % счетчик итераций
IterPlus=1; % счетчик удачных итераций
MRD(IterPlus)=0;
H = Hmin;
Accum(H+hh) = 0;
invZ=0; %результат обращения
Enext=1;
Eprev=Enext;
b1=1;
while 1
    Enext=Eprev-Z*Dev(H);
    c1=abs(Enext);
    compl = c1-b1;
    DM(Iter)=abs(compl);
    Iter = Iter + 1;
    if (compl <= 0)
        IterPlus=IterPlus+1;
        MRD(IterPlus)=c1;
        Eprev=Enext;
        b1=c1;
        Accum(H+hh) = Accum(H+hh) + 1;
        invZ=invZ+Dev(H);
    else
        H = H + 1;
        if H == Hmax
            return
        end
        Accum(H+hh) = 0;
    end
end
end

```

```

function
[Iter,IterPlus,Hmax,Z,invZ,Zcont,invZcont,Enext,er,ma] =
testDecDev1
% функция тестирует функцию DecDev1
Hmin=-1;
Hmax=55;
Z=(0.445+j*0.222);

```

```
[Iter,invZ,Accum,DM,Enext,IterPlus,MRD]=DecDev1(Z,Hmin,H
max);
invZcont=1/Z;
er=-log2(norm(invZ-invZcont)); % ошибка вычисления,
    % преобразованная в количество верных разрядов
    % результата (в данном примере - 27 разрядов)
ma=max(Accum); % максимальное значение разряда в Accum
it=1:1:Iter-1;
itp=1:1:IterPlus;
subplot(2,1,1); % зависимости log2(MRD)=f(Iter)
    % и log2(DM)=f(IterPlus) - см. верхнее
    % окно на рис. 3
plot(itp,log2(MRD),'r-',it,log2(DM),
    'b-',it,zeros(1,Iter-1),'k-');
title('MRD-module of rest of decomposition;
    DM-difference of modules');
hText=text(5,-20,'Red: Log2(DM)=f(IterPlus)');
hText=text(40,-5,'Blue: Log2(MRD)=f(Iter)');
subplot(2,1,2); % гистограмма значений разрядов
    % в Accum - см. нижнее окно на рис. 3
bar(Accum,0.4);
title('Accum = f(H)');
```

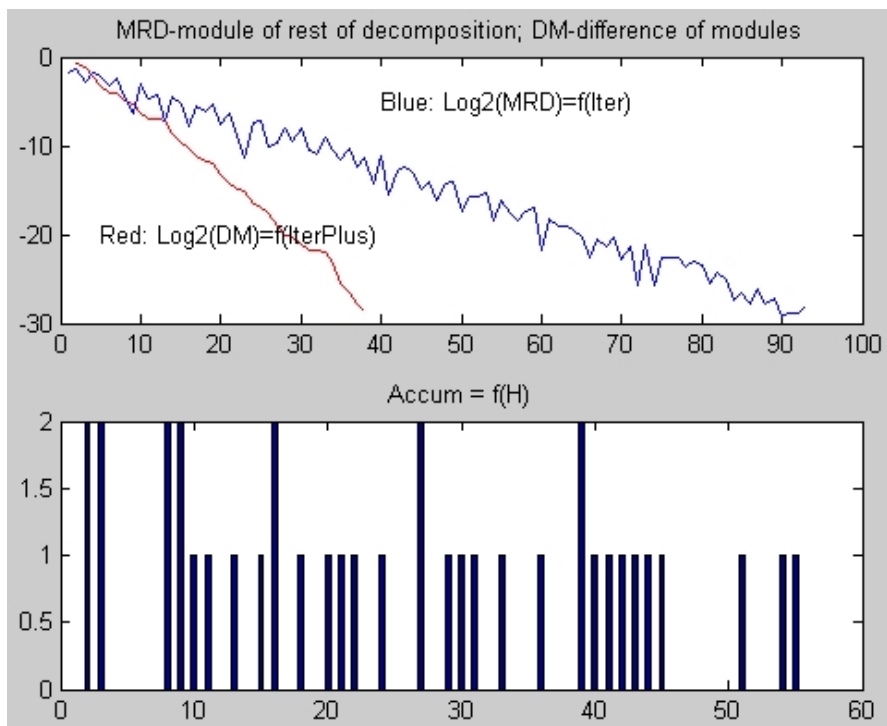


Рис. 3.

function testDecDevAn

```

% функция строит область определения исходного
% числа и его инверсии
pixel=93;
Hmin=-1;
Hmax=55;
sigmaI=0;
k=1;
kk=pixel;
koefx=0.5; %  $0.5 < x < 1$ 
ad=0.5;
while k <= kk
    x(k)=ad+k*koefx/kk;
    m=1;
    while m <= 2*kk
        y(m)=1-m/kk; %  $-1 < y < 0$ 
        W(k,m)=x(k)+j*y(m);
        [Iter, invW(k,m), Accum, DM, Enext(m), IterPlus,
         MRD] = DecDev1 (W(k,m), Hmin, Hmax);
        maccum(m)=max(Accum);
        sigmaI=sigmaI+Iter;
        W1(k,m)=0;
        W2(k,m)=0;
        invW1(k,m)=0;
        invW2(k,m)=0;
        if maccum(m)<2
            W1(k,m)=W(k,m);
            invW1(k,m)=invW(k,m);
        elseif maccum(m)==2
            invW2(k,m)=invW(k,m);
            W2(k,m)=W(k,m);
        end
        m=m+1;
    end
    maxMac(k)=max(maccum);
    err(k)=max(abs(Enext));
    k=k+1;
end
error=max(err) % максимальная ошибка - в данном
                % случае =  $(6+3j)*e-9$ 
maxMaccum=max(maxMac) % максимальное значение разряда
                    % в Accum - в данном случае = 2
subplot(1,2,1); % область определения исходного
                % числа - см. правое окно на рис. 4
title('maxAccum in Decomposition');
plot(real(W2),imag(W2),'r.')
    hold on
plot(real(W1),imag(W1),'b.')
grid on;

```

```
subplot(1,2,2); % область определения инверсии
% числа - см. левое окно на рис. 4
title('maxAccum in Devision');
plot(real(invW2),imag(invW2),'r.')
hold on
plot(real(invW1),imag(invW1),'b.')
grid on;
sigmaI; % общее число итераций
otnIH=sigmaI/(0.5*sigmaI*(Hmax-Hmin))
% относительное число итераций на разряд
% каждой части комплексного числа (равно 2)
```

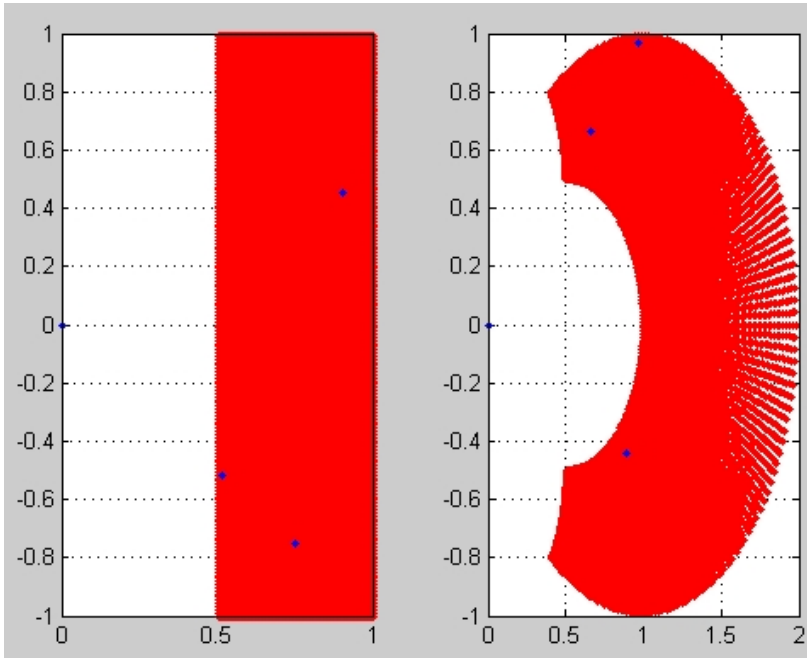


Рис. 4.

7.3. Деление действительных чисел в системе с основанием «-2»

Деление кодов действительных чисел по основанию «-2» во многом аналогично делению комплексных чисел, рассмотренному в разделе 7.2. При определенном выборе соотношения между модулями делителя и делимого количество вычитаний не превышает 2. Оно может выполняться в два этапа:

- декомпозиция делимого, т.е. накопление единиц в разрядах частного;
- композиция, т.е. сложение единиц в разрядах частного.

Декомпозиция для деления в данном случае состоит в разложении данного комплексного числа Z по формуле $\frac{1}{16} = \sum a_h Z(-2)^h$. Схема алгоритма декомпозиции при делении действительного числа `DecompDevisReal` представлена на рис. 1.

Композиция `CompDevis` в данном случае состоит в вычислении по формуле $\frac{1}{16Z} = \sum a_h (-2)^h$. Она эквивалентна сложению кодов, полученных в двух регистрах `Acc1` и `Acc2`, по формуле: $W = \langle Acc1 \rangle - 2 \cdot \langle Acc2 \rangle$.

Обращение и деление действительного числа выполняется также, как и для комплексного числа – см. раздел 7.2.

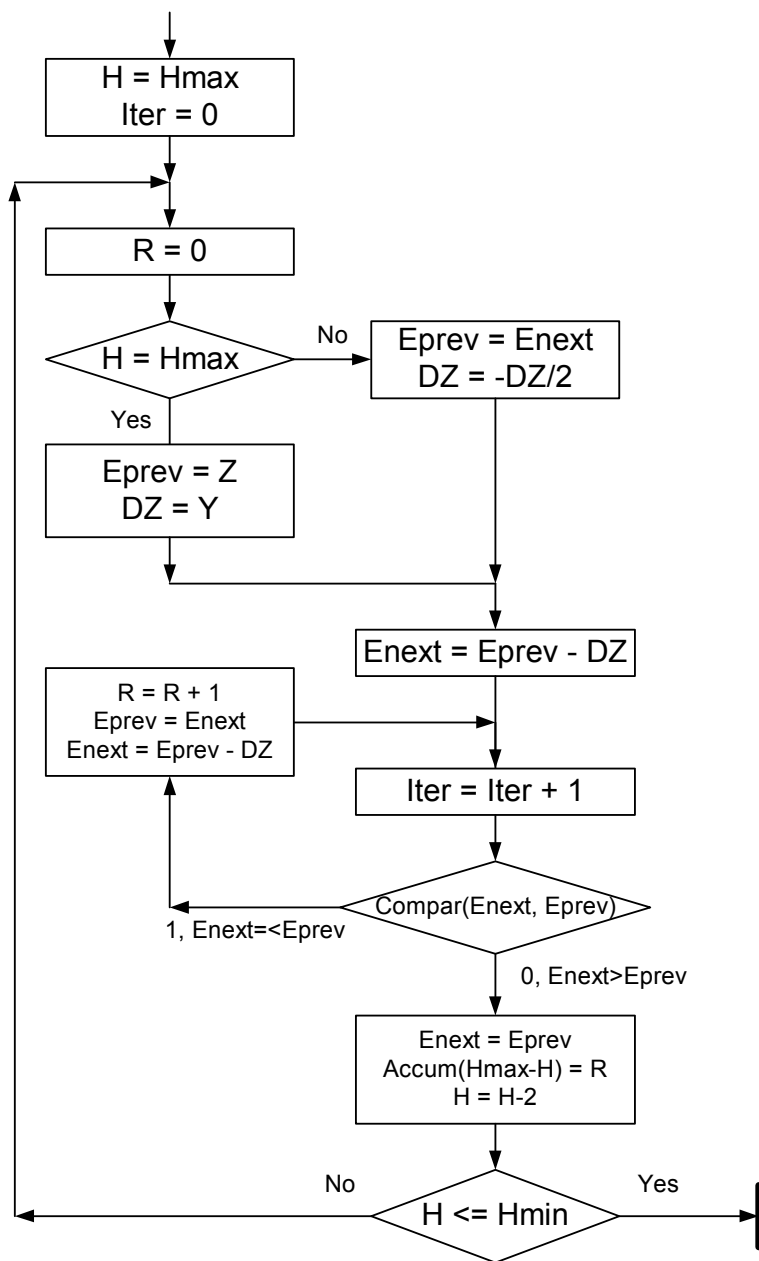


Рис. 1.

7.4. Деление трехмерных векторов

Рассмотрим **деление векторов**, как операцию, обратную умножению в алгебре, рассмотренной в главе 5. Программы для этого раздела находятся в каталоге CD/Vectory. Итак,

$$\text{if } U = U_1 * U_2, \text{ then } U_1 = U / U_2.$$

Рассмотрим частный случай – определение обратного вектора, когда $U=1$. Пусть

$$U_1 = x \cdot i + y \cdot j + z \cdot k,$$

$$U_2 = a \cdot i + b \cdot j + c \cdot k.$$

Тогда

$$ax - yc - zb = 1,$$

$$bx + ya - zc = 0,$$

$$cx + yb + za = 0.$$

Эта система уравнений неразрешима, если ранг R расширенной матрицы коэффициентов не равен рангу P матрицы коэффициентов. Решение существует и единственно, если $R=P=3$. Матрица коэффициентов имеет вид

$$A = \begin{vmatrix} a & -c & -b \\ b & a & -c \\ c & b & a \end{vmatrix},$$

а расширенная матрица коэффициентов имеет вид

$$B = \begin{vmatrix} a & -c & -b & 1 \\ b & a & -c & 0 \\ c & b & a & 0 \end{vmatrix}.$$

Применяя разложение определителя $\det(B)$ матрицы B по четвертому столбцу, находим, что он и определитель $\det(A)$

матрицы A связаны соотношением: $\det(B) = \det(A)$. Следовательно, $R=P$.

Ранг матрицы коэффициентов $P=3$, если ее определитель не равен нулю, т.е.

$$(a^3 - b^3 + c^3 + 3abc) \neq 0. \quad (1)$$

Таким образом, $R=P$ и существует единственное решение при выполнении условия (1). Другими словами, при выполнении условия (1) возможно вычисление обратного вектора.

Преобразуем (1) к виду

$$a(bc + a^2) - b(ac + b^2) + c(ab + c^2) \neq 0. \quad (2)$$

Деление может быть заменено умножением на обратный вектор. Следовательно, деление векторов возможно, если координаты делителя удовлетворяют условию (1) или (2).

Пример 2. Программы MATLAB для деления векторов в системе кодирования по основанию (1.29) и $R=2$.

function [DZ]=vec3(H)

```
% функция вычисляет трехмерный вектор, аналогичный
% величине DZ для комплексного числа - см. пример 1
DZ=[0,0,0];
a=-H;
b=-H-1;
c=-H-2;
if a == 3 * round(a/3)
    DZ(1)=(-2)^(a/3);
elseif b == 3 * round(b/3)
    DZ(2)=(-2)^(b/3);
elseif c == 3 * round(c/3)
    DZ(3)=(-2)^(c/3);
End
```

function [c]=mult3(a,b)

```
% умножение векторов в специальной алгебре
c(1)=a(1)*b(1)-a(2)*b(3)-a(3)*b(2);
c(2)=a(1)*b(2)+a(2)*b(2)-a(3)*b(3);
c(3)=a(1)*b(3)+a(2)*b(1)+a(3)*b(1);
```

**function [Iter,invZ,Accum,DM,Enext,IterPlus,MRD]
=DecVec(Z,Hmin,Hmax)**

```
% функция реализует алгоритм обращения вектора Z,  
    аналогичный обращению комплексного числа,  
    приведенного на рис. 2.
```

```
% MRD – модуль остатка при декомпозиции
% DM – модуль разности модулей остатка
Accum=zeros(1,Hmax+1);
hh=1-Hmin;
Iter = 1; % счетчик итераций
IterPlus=1; % счетчик удачных итераций
MRD(IterPlus)=0;
H = Hmin;
Accum(H+hh) = 0;
invZ==[0,0,0]; % результат обращения
Enext=[1,0,0];
Eprev=Enext;
b1=1;
while 1
    Enext=Eprev-mult3(Z,vec3(H));
    c1 = sum(Enext.^2) % квадрат модуля вектора
    compl = c1-b1;
    DM(Iter)=abs(compl);
    Iter = Iter + 1;
    if (compl <= 0)
        IterPlus=IterPlus+1;
        MRD(IterPlus)=c1;
        Eprev=Enext;
        b1=c1;
        Accum(H+hh) = Accum(H+hh) + 1;
    if Accum(H+hh)>8
        return
    end
    invZ=invZ+vec3(H);
else
    H = H + 1;
    if H == Hmax
        return
    end
    Accum(H+hh) = 0;
end
end

function
[Iter,IterPlus,Hmax,Z,invZ,Zcont,invZcont,Rest,er,ma]
    = testDecVec
% функция тестирует функцию DecVec
Hmin=-1;
Hmax=55;
Z=[0.445,0.222,-0.1];
[Iter,invZ,Accum,DM,Rest,IterPlus,MRD]=
    DecVec(Z,Hmin,Hmax);
er=sqrt(0.33*sum([1,0,0]-mult3(Z,invZ)).^2));% ошибка
ma=max(Accum); % максимальное значение разряда в Accum
```

```

it=1:1:Iter-1;
itp=1:1:IterPlus;
subplot(2,1,1); % зависимости log2(MRD)=f(Iter)
                  % и log2(DM)=f(IterPlus) - см. верхнее
                  % окно на рис. 5
plot(itp,log2(MRD),'r-',it,log2(DM),
      'b-',it,zeros(1,Iter-1),'k-');
title('MRD-module of rest of decomposition;
      DM-difference of modules');
hText=text(5,-20,'Red: Log2(DM)=f(IterPlus)');
hText=text(40,-5,'Blue: Log2(MRD)=f(Iter)');
subplot(2,1,2); % гистограмма значений разрядов
                  % в Accum - см. нижнее окно на рис. 5
bar(Accum,0.4);
title('Accum = f(H)');

```

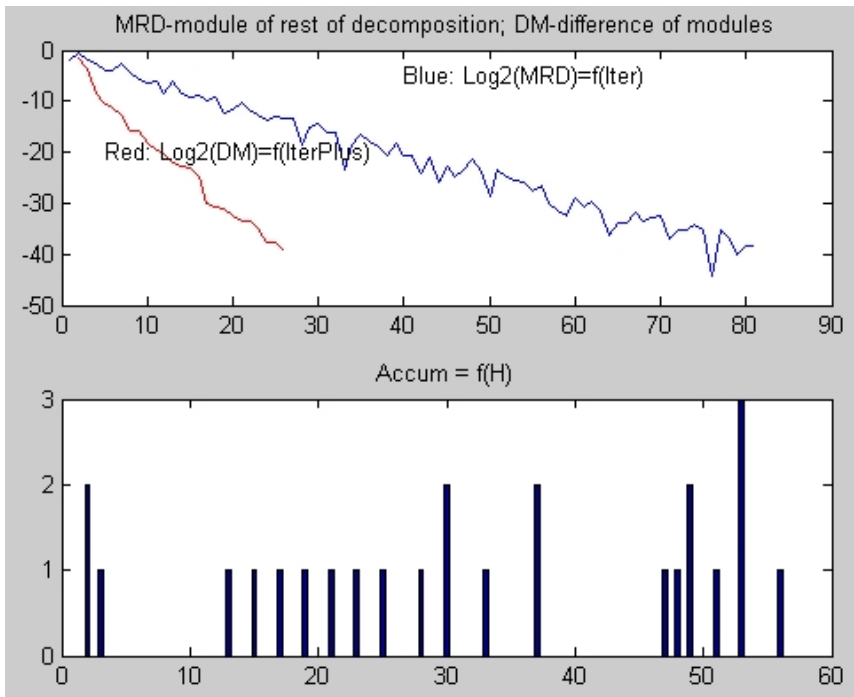


Рис. 5.

function testDecVecSeg2

```

% функция анализирует точность обращения вектора,
% расположенного в определенной области - кубе
% положительных значений нормализованных координат
% вектора:

```

$$0.5 \leq x \leq 1, \quad 0.5 \leq y \leq 1, \quad 0.5 \leq z \leq 1.$$

```

kk=10;

```

```
xmin=0.5; xmax=1; ymin=0.5; ymax=1; zmin=0.5; zmax=1;
Hmin=-1; Hmax=85;
sigmaI=0;
k=1;
while k <= kk
    x(k)=xmin+k*(xmax-xmin)/kk;
    m=1;
    while m <= kk
        y(m)=ymin+k*(ymax-ymin)/kk;
        n=1;
        while n <= kk
            z(n)=zmin+k*(zmax-zmin)/kk;
            vec=[x(k),y(m),z(n)]; % инвертируемый вектор
            [Iter,invec,Accum,DM,Rest,IterPlus,MRD]
            =DecVec(vec,Hmin,Hmax);
            invec; % результат инверсии
            ero(m)=sqrt(0.33*sum([1,0,0]-
            (mult3(vec,invec)).^2)); %ошибка
            maxcum(m)=max(Accum); % максимальное
            % значение в Accum
            sigmaI=sigmaI+Iter;
            n=n+1;
        end
        maxMacZ(m)=max(maxcum);
        errZ(m)=max(ero);
        m=m+1;
    end
    maxMacY(k)=max(maxMacZ);
    errY(k)=max(errZ);
    k=k+1;
end
sigmaI;
otnIH=sigmaI/(0.3*sigmaP*(Hmax-Hmin))
    % относительное число итераций на разряд
    % каждой части вектора (равно 5)
error=max(errY) % максимальная ошибка - в данном
    % случае = 8e-6
maxMassum=max(maxMacY) % максимальное
    % значение в Accum, равное 6
```

function testDecVecAn3

% функция анализирует точность обращения вектора,
 % расположенного в определенной области - призме,
 % изображенной на рис. 6 и удовлетворяющей условию вида

$$0.5 \leq x \leq 1, 0 \leq y \leq x, 0 \leq z \leq y. \quad (1)$$

```
kk=20;
xmin=0.5; xmax=1;
Hmin=-1; Hmax=85;
sigmaI=0;
```

```

k=1;
while k <= kk
    x(k)=xmin+k*(xmax-xmin)/kk;
    m=1;
    ymin=0;
    ymax=x(k);
    while m <= kk
        y(m)=ymin+k*(ymax-ymin)/kk;
        n=1;
        zmin=0;
        zmax=y(m);
        while n <= kk
            z(n)=zmin+k*(zmax-zmin)/kk;
            vec=[x(k), y(m), z(n)]; % инвертируемый вектор
%
% далее также, как в функции testDecVecSeg2
%
    otnIH=sigmaI/(0.3* sigmaI*(Hmax-Hmin))
        % относительное число итераций на разряд
        % каждой части вектора (равно 5)
    error=max(errY) % максимальная ошибка - в данном
        % случае = 6.2e-9
    maxMassum=max(maxMasY) % максимальное
        % значение в Assum, равное 7

```

Рассмотрим вновь *призму* положительных значений нормализованных координат вектора - см. (1) и рис. 6. Из анализа функции **testDecVecAn3** следует, что любой вектор в указанной области может быть инвертирован при малой кратности значений в Assum. Исключением является выделенное на рис. 6 *ребро AB*, где НЕ выполняется условие (1.1).

Покажем еще, что любая область дробных координат вектора, среди которых хотя бы одна нормализована, может быть преобразована в область (1).

Рассмотрим табл. 1, где

x, y, z - координаты инвертируемого вектора,

R – операция предварительного преобразования инвертируемого вектора,

xI, yI, zI - координаты преобразованного для инверсии вектора, расположенного в области (2).

Операция R , как правило, представляет собой умножение на орт или на отрицательный орт. Символом $(y \Leftrightarrow z)$ обозначена *перестановка координат*, предшествующая умножению на орт. Значения координат xI, yI, zI получены в соответствии с табл. 5.1. Видно, что все координаты xI, yI, zI являются положительными.

После получения вектора с координатами $x1, y1, z1$ необходимо переставить эти координаты так, чтобы выполнить условие (1). Обозначим эту перестановку символом P . После этой перестановки формируется вектор с координатами $x2, y2, z2$. Наконец, после инвертирования вектора с координатами $x2, y2, z2$ получается вектор с координатами $x3, y3, z3$.

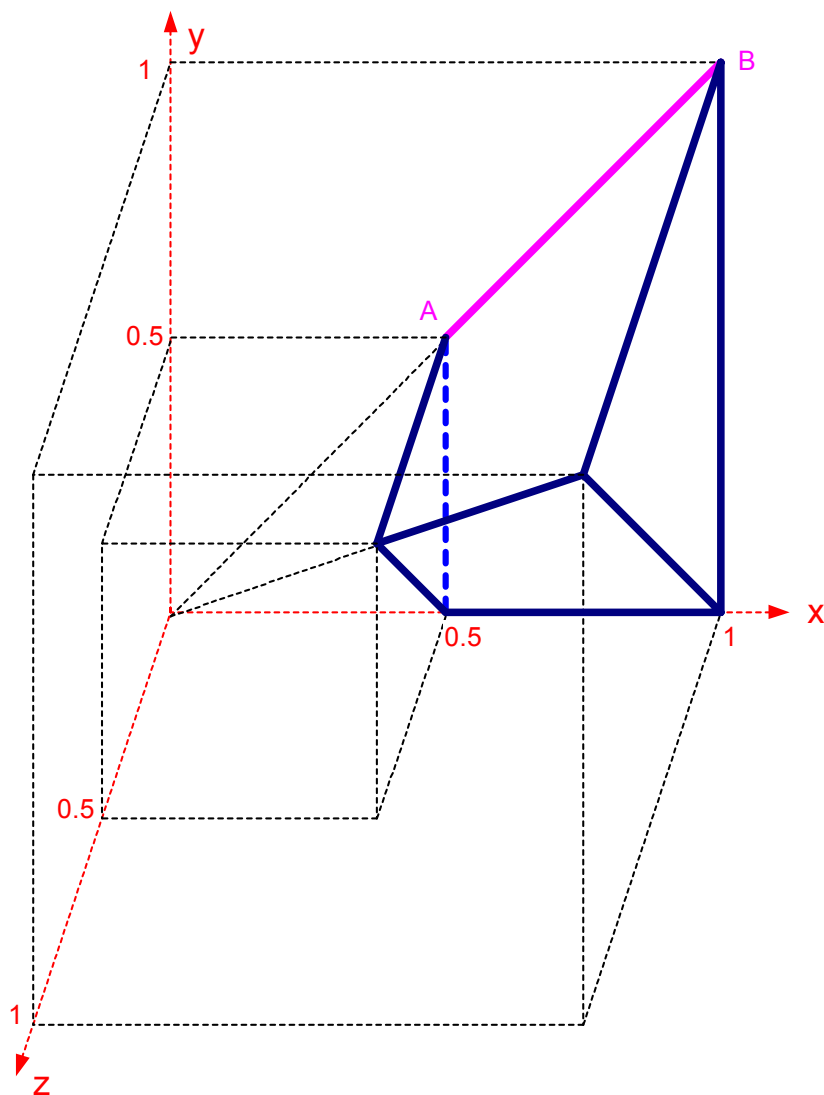


Рис. 6.

После получения вектора с координатами x_3, y_3, z_3 необходимо выполнить с этим вектором операцию Q , обратную операции P , и операцию S , обратную операции R . Операция S представлена в табл. 1. Важно отметить, что перестановка координат ($y \leftrightarrow z$) должна быть выполнена и у результирующего вектора.

Все указанные операции представляют собой перестановки координат и, может быть, изменение знака некоторых координат. Таким образом, предлагаемый метод позволяет *инвертировать* вектор с нормализованными координатами и, следовательно *делить* такие векторы.

Таблица 1.

R	x	y	z	xI	yI	zI	S
i	+	+	+	$+x$	$+y$	$+z$	i
$-k$	+	+	-	$-z$	$+y$	$+x$	j
$-k \cdot (y \leftrightarrow z)$	+	-	+	$-z$	$+y$	$+x$	$j \cdot (y \leftrightarrow z)$
$-j$	+	-	-	$-y$	$-z$	$+x$	k
j	-	+	+	$+y$	$+z$	$-x$	$-k$
$k \cdot (y \leftrightarrow z)$	-	+	-	$+z$	$-x$	$-y$	$-j \cdot (y \leftrightarrow z)$
k	-	-	+	$+z$	$-x$	$-y$	$-j$
$-i$	-	-	-	$-x$	$-y$	$-z$	$-i$

Глава 8. Вычисление функций комплексного переменного

8.1. Логарифмирование

8.1.1. Определение натурального логарифма комплексного числа (вариант 1).

Логарифмирование основано на следующем: если $y = \prod (1 + \rho^{-h})^{a_h}$, то $\ln(y) = \sum a_h \ln(1 + \rho^h)$. Алгоритм определения натурального логарифма комплексного числа состоит в следующем:

1. Дано число $Z = (-2)^k y$, где y - мантисса, k - порядок.
2. Декомпозиция по формуле $y = \prod (1 + \rho^{-h})^{a_h}$ - “*DecompBinom*”.
3. Композиция по формуле $\ln(y) = w = \sum a_h \ln(1 + \rho^h)$ - “*CompLogar*”.
4. Вычисление $\ln(Z) = [k \cdot \ln(-2) + \ln(y) + S \cdot \ln(-1)]$. При этом $\ln(-2) = \ln(-1) + \ln(2) = j\pi + \ln(2)$. Таким образом, $\ln(Z) = k \cdot \ln(2) + w + (k + S)j\pi = \text{Re } Z + \text{Im } Z$, где $\text{Re } Z = k \cdot \ln(2) + \text{Re } w$, $\text{Im } Z = j\pi(k + S) + \text{Im } w$.
5. Определение главного значения натурального логарифма. В нем мнимая часть находится в пределах $-\pi \leq \text{Im } Z \leq \pi$. Поэтому мнимая часть главного значения определяется по формуле:

$$g = \text{int} \left[\frac{\text{Im } Z}{2\pi} \right], \quad (\text{Im } Z)_{\text{main}} = \begin{cases} g & \text{if } g \leq \pi, \\ g - 2\pi & \text{if } g > \pi. \end{cases}$$

6. Нормализация результата

8.1.2. Вычисление логарифма модуля комплексного числа

Алгоритм вычисления логарифма модуля комплексного числа основан на том, что $\ln(|Z|) = \operatorname{Re} \ln(Z)$, и отличается от алгоритма логарифмирования по варианту 1 только тем, что в п. 3 вместо композиции “*CompLogar*” используется композиция “*CompLogarModul*”:

3. Композиция по формуле $W = \operatorname{Re} \sum a_h \left[\ln(1 + \rho^{-h}) \right]$ - “*CompLogarModul*”.

8.1.3. Определение натурального логарифма комплексного числа (вариант 2)

Логарифмирование в этом случае основано на следующем: если $y = \prod (1 + \rho^{-h})^{2 \cdot a_h}$, то $\ln(y) = 2 \cdot \sum a_h \ln(1 + \rho^h)$. Точность логарифмирования по этому варианту меньше точности логарифмирования по первому варианту, но, как будет ясно из дальнейшего, этот вариант хорошо совмещается с другими вычислениями. Алгоритм состоит в следующем:

1. Дано число $Z = (-2)^k y$, где y - мантисса, k - порядок.
2. Декомпозиция по формуле $y = \prod (1 + \rho^{-h})^{2 \cdot a_h}$ - (“*DecompBinom2*”).
3. Композиция по формуле $w = \sum a_h \ln(1 + \rho^h)$ - (“*CompLogar*”).
4. Вычисление $\ln(Z) = [k \cdot \ln(-2) + \ln(y)]$. При этом $\ln(-2) = \ln(-1) + \ln(2) = j\pi + \ln(2)$. Таким образом, $\ln(Z) = k \cdot \ln(2) + Y + kj\pi = \operatorname{Re} Z + \operatorname{Im} Z$, где $\operatorname{Re} Z = k \cdot \ln(2) + \operatorname{Re} Y$, $\operatorname{Im} Z = kj\pi + \operatorname{Im} Y$.
5. Определение главного значения натурального логарифма. В нем мнимая часть находится в пределах $-\pi \leq \operatorname{Im} Z \leq \pi$. Поэтому мнимая часть главного значения определяется по формуле:

$$g = \operatorname{int} \left[\frac{\operatorname{Im} Z}{2\pi} \right], \quad (\operatorname{Im} Z)_{\text{main}} = \left\{ \begin{array}{ll} g & \text{if } g \leq \pi, \\ g - 2\pi & \text{if } g > \pi. \end{array} \right\}.$$

6. Нормализация результата

8.1.4. Определение натурального логарифма положительного действительного числа

Алгоритм определения натурального логарифма
положительного действительного числа состоит в следующем:

1. Дано число $Z = (-2)^k y$, где $y > 0$ - мантисса, k - порядок.
2. Декомпозиция числа $y > 0$ по формуле $y = \prod (1 + \rho^{-h})^{a_h}$ -
“*DecompBinomReal*”.
3. Композиция по формуле $w = \sum a_h \ln(1 + \rho^h)$ -
 (“*CompLogar*”).
4. Вычисление $\ln(Z) = [k \cdot \ln(-2) + w]$. При этом,
 - если k – четное, то $\ln(Z) = k \cdot \ln(2) + w$,
 - если k – нечетное, то $\ln(Z) = \pi + k \cdot \ln(2) + w$.
5. Нормализация результата

8.1.5. Определение натурального логарифма отрицательного действительного числа

Алгоритм определения натурального логарифма
отрицательного действительного числа состоит в следующем:

1. Дано число $Z = (-2)^k y$, где $y < 0$ - мантисса, k - порядок.
2. Декомпозиция числа $(-y) > 0$ по формуле

$$y = \prod (1 + \rho^{-h})^{a_h} \text{ - (“DecompBinomReal”).}$$
3. Композиция по формуле $w = \sum a_h \ln(1 + \rho^h)$ -
 (“*CompLogar*”).
4. Вычисление $\ln(Z) = [k \cdot \ln(-2) + w + \ln(-1)]$. При этом,
 - если k – четное, то $\ln(Z) = \pi + k \cdot \ln(2) + w$,
 - если k – нечетное, то $\ln(Z) = k \cdot \ln(2) + w$.
5. Нормализация результата.

8.1.6. Определение натурального логарифма действительного числа

Вычисление в этом случае начинается с анализа знака логарифмируемого числа и обращении к логарифмированию положительного или отрицательного числа.

8.2.2. Потенцирование

8.2.1. Потенцирование комплексного числа - Potentiating

Потенцирование состоит в определении числа $Z = e^X$ в комплексной степени $X = (-2)^p m$, где m - мантисса, p - порядок. Алгоритм потенцирования основан на следующем: если

$X = \sum a_h \ln(1 + \rho^h)$, то $e^X = \prod (1 + \rho^{-h})^{a_h}$. Заметим, что

$$e^{\beta \ln(-2)} = (-2)^\beta, \quad e^{\beta \ln(2)} = (2)^\beta,$$

$$\ln(-2) = \ln(-1) + \ln(2) = j\pi + \ln(2).$$

Преобразуем данное число X :

$$X = \operatorname{Re} X + j \operatorname{Im} X = 2x' + \beta \ln(-2) + j \operatorname{Im} X =$$

$$= 2x' + \beta \ln(-2) + j(2x'' + \mu \frac{\pi}{2}) =$$

$$= 2x' + \beta \ln(2) + j\beta\pi + j(2x'' + \mu \frac{\pi}{2}) =$$

$$= \beta \ln(2) + j \frac{\pi}{2} (2\beta + \mu) + 2(x' + jx'').$$

Таким образом,

$$\operatorname{Re} X = \beta \ln(2) + 2x', \quad j \operatorname{Im} X = j \left(\frac{\pi}{2} (2\beta + \mu) + 2x'' \right).$$

В этих соотношениях

$$\beta - \text{целая часть от частного } X' = \frac{\operatorname{Re} X}{\ln(2)},$$

$$\mu - \text{целая часть от частного } X'' = \frac{\operatorname{Im} X}{\pi/2}.$$

$$2x' - \text{дробная часть от частного } X' = \frac{\operatorname{Re} X}{\ln(2)},$$

$$2x'' - \text{дробная часть от частного } X'' = \frac{\operatorname{Im} X}{\pi/2}.$$

Отсюда следует, что

$$\begin{aligned}\exp(X) &= \exp\left[(\beta \ln(2) + 2x') + j\left(\frac{\pi}{2}(2\beta + \mu) + 2x''\right)\right] = \\ &= (-2)^\beta e^{2(x' + jx'')} e^{j\frac{\pi}{2}(2\beta + \mu)} = (-2)^\beta e^{2y} e^{j\frac{\pi}{2}\gamma} \beta_0 = \\ &= (-2)^\beta e^{2y} \theta,\end{aligned}$$

где $y = (x' + jx'')$, $\gamma = (2\beta + \mu)$, $\vartheta = e^{j\frac{\pi}{2}\gamma} \beta_0$,

$$\beta_0 = \begin{cases} 1 & \text{if } \beta - \text{even} \\ -1 & \text{if } \beta - \text{odd} \end{cases}. \text{ При этом } \vartheta = \begin{cases} j & \text{if } \eta \cdot \beta_0 = 1 \\ 1 & \text{if } \eta \cdot \beta_0 = 0 \\ -j & \text{if } \eta \cdot \beta_0 = -1 \\ -1 & \text{if } \eta \cdot \beta_0 = -2 \end{cases}, \text{ где}$$

$\eta = \{-2, -1, 0, 1\}$ - остаток от деления целого числа μ на 4.

Алгоритм потенцирования состоит в следующем:

1. Дано число $X = (-2)^p m$, где m - мантисса, p - порядок.
2. Если $X = 0$, то $Z = 1$.

3. Вычисление $X' = \frac{\operatorname{Re} X}{\ln(2)}$, $X'' = \frac{\operatorname{Im} X}{\pi/2}$.

4. Выделение из чисел X' , X'' целых β , μ и дробных

x' , x'' частей соответственно.

5. Вычисление дробных частей $y' = x' \frac{\ln(2)}{2}$, $y'' = x'' \frac{\pi}{4}$ и

формирование кода числа $y = (y' + jy'')$, который не содержит экспоненты.

6. Декомпозиция числа $y = (y' + jy'')$ по формуле

$$y = \sum a_h \ln(1 + \rho^h) - \text{“DecompLogar”}.$$

7. Композиция по формуле $W = \prod (1 + \rho^{-h})^{ah}$ - "CompBinom".
8. Вычисление величины θ по вышеприведенной формуле в зависимости от младших разрядов кодов чисел μ, β .
9. Определение $\exp(X) = (-2)^\beta \cdot \theta \cdot W^2$
10. Нормализация результата

8.2.2. Потенцирование действительного числа - PotentiatingReal

Потенцирование действительного числа состоит в определении числа $Z = e^X$ в действительной степени $X = (-2)^p m$, где m - мантисса, p - порядок. По аналогии с предыдущим заметим, что

$$e^{\beta \ln(-2)} = (-2)^\beta, \quad e^{\beta \ln(2)} = (2)^\beta, \\ \ln(-2) = \ln(-1) + \ln(2) = j\pi + \ln(2).$$

Преобразуем данное число X :

$$X = x' + \beta \ln(-2) = x' + \beta \ln(2) + j\pi\beta.$$

В этих соотношениях

$$\beta - \text{целая часть от частного } X' = \frac{X}{\ln(2)},$$

$$y - \text{дробная часть от частного } X' = \frac{X}{\ln(2)}.$$

Отсюда следует, что

$$\exp(X) = \exp[(\beta \ln(2) + 2x') + j\pi\beta] = \\ = 2^\beta e^{2x'} e^{j\pi\beta} = (-2)^\beta e^{2x'} e^{j\pi\beta} \beta_0 = (-2)^\beta e^{2x'} \theta, \quad \text{где} \\ \theta = e^{j\pi\beta} \beta_0, \quad \beta_0 = \begin{cases} 1 & \text{if } \beta - \text{even} \\ -1 & \text{if } \beta - \text{odd} \end{cases}. \quad \text{Очевидно, } \theta = 1.$$

Следовательно, $\exp(X) = (-2)^\beta e^y$.

Алгоритм потенцирования действительного числа состоит в следующем:

1. Дано число $X = (-2)^p m$, где m - мантисса, p - порядок.

2. Если $X = 0$, то $Z = 1$.

3. Вычисление $X' = \frac{X}{\ln(2)}$.

4. Выделение из числа X' целой β и дробной γ частей.

5. Декомпозиция числа γ по формуле $\gamma = \sum a_h \ln(1 + \rho^h)$ - "*DecompLogarReal*".

6. Композиция по формуле $W = \prod (1 + \rho^{-h})^{a_h}$ - "*CompBinom*".

7. Определение $\exp(X) = (-2)^\beta \cdot W$

8. Нормализация результата

8.3. Операции с логарифмическими формами.

8.3.1. Логарифмическая форма представления комплексного числа

Логарифмическая форма представлена на рис. 1 и используется для представления комплексного числа в виде $Z = 2^{\omega} m \cdot e^{j\varphi}$, где ω - целое число, m - дробное положительное число, φ - действительное число, главное значение аргумента, которое находится в пределах $(-\pi) \leq \varphi \leq \pi$. При этом

- в экспоненте записывается ω ,
- в реальной части мантиссы записывается $\log_2(m)$,
- в мнимой части мантиссы записывается число $j\phi = \frac{j\varphi}{16}$;

при этом $-\frac{\pi}{16} \leq \phi \leq \frac{\pi}{16}$.

63	62	54	53	52	2	1	0
Экспонента ω				Логарифм мантиссы $\log_2(M)$					

Рис. 1.

8.3.2. Формирования логарифмической формы - FormLogar

1. Дано число $Z = (-2)^k \cdot M$
2. Оно преобразуется к виду

$$Z = 2^{\omega} z, \text{ where } z = \frac{M}{d},$$

$$\left\{ \begin{array}{l} \omega = k, \quad d = 1 \text{ if } k - \text{even} \\ \omega = k + 1, \quad d = -2 \text{ if } k - \text{odd} \end{array} \right\}$$

3. Логарифмирование мантиссы Z - вычисление натурального логарифма $\ln(z) = \ln(|z|) + j\varphi$; при этом в мнимой части записывается главное значение аргумента

логарифма мантиссы, который должен находиться в пределах $(-\pi) \leq \varphi \leq \pi$;

4. Вычисление $m = \log_2(|z|) = \ln(|z|) \cdot \frac{1}{\ln(2)}$
5. Запись в экспоненту числа ω
6. Запись в реальную часть числа $m = \log_2(|z|)$,
7. Запись в мнимую часть числа $j\phi = \frac{j\varphi}{16}$

Обращаясь к общему алгоритму логарифмирования, рассмотрим алгоритм логарифмирования величины Z , который состоит в следующем:

1. Преобразование числа M к виду, допускающему необходимую для логарифмирования декомпозицию. В результате преобразования определяется число S и комплексное число $y = Q^*z$.
2. Декомпозиция по формуле $y = \prod (1 + \rho^{-h})^{a_h}$ - “*DecompBinom*”.
3. Композиция по формуле $\ln(y) = w = \sum a_h \ln(1 + \rho^h)$ - “*CompLogar*”.
4. Вычисление $\ln(Z) = \text{Re } Z + \text{Im } Z$, где
 $\text{Re } Z = \text{Re } w$, $\text{Im } Z = jS\pi + \text{Im } w$.
5. Определение главного значения натурального логарифма. В нем мнимая часть находится в пределах $-\pi \leq \text{Im } Z \leq \pi$. Поэтому мнимая часть главного значения определяется по формуле:

$$g = \text{int} \left[\frac{\text{Im } Z}{2\pi} \right], \quad (\text{Im } Z)_{\text{main}} = \begin{cases} g & \text{if } g \leq \pi, \\ g - 2\pi & \text{if } g > \pi. \end{cases}$$

8.3.3. Возврат из логарифмической формы – RetLogar

1. Дано число $V = \omega + \log(w) + j\phi$
2. Выделение из экспоненты числа ω
3. Выделение из действительной части числа $m = \log_2(w)$,

4. Выделение из мнимой части числа $j\phi = \frac{j\phi}{16}$

5. Вычисление $Z = q(-2)^\omega M$, где

$$M = \exp[\ln(2) \cdot m + 16j\phi], \quad q = \begin{cases} 1, & \text{if } \omega - \text{even} \\ -1, & \text{if } \omega - \text{odd} \end{cases}$$

Обращаясь к общему алгоритму потенцирования, рассмотрим алгоритм потенцирования величины X , который состоит в следующем:

1. Если $X = 0$, то $M = 1$.

3. Вычисление $X' = \frac{\operatorname{Re} X}{\ln(2)} = m$, $X'' = \frac{\operatorname{Im} X}{\pi/2} = \frac{32\phi}{\pi}$.

4. Выделение из чисел X' , X'' целых $\beta = 0$, μ и дробных

$x' = m$, x'' частей соответственно. Выделение целых и дробных частей выполняется таким образом, что дробные части числа $y = (x' + jx'')$ находятся в следующих пределах: $-\frac{2\ln(2)}{3} \leq x' \leq \frac{\ln(2)}{3}$, $-\frac{\pi}{3} \leq x'' \leq \frac{\pi}{6}$.

5. Декомпозиция числа $y = (x' + jx'')$ по формуле

$$y = \sum a_h \ln(1 + \rho^h) - \text{“DecompLogar”}.$$

6. Композиция по формуле $W = \prod (1 + \rho^{-h})^{a_h}$ - “CompBinom”.

7. Определение $\exp(X) = \theta \cdot W$. При этом

$$\theta = \begin{cases} j & \text{if } \eta = 1 \\ 1 & \text{if } \eta = 0 \\ -j & \text{if } \eta = -1 \\ -1 & \text{if } \eta = -2 \end{cases},$$

где η - остаток от деления целого числа μ на 4.

8.3.4. Алгебраическое сложение логарифмических форм

Умножение и деление чисел, представленных в логарифмической форме, эквивалентно алгебраическому сложению этих форм. Действительно, если $Z' = 2^{\omega'} m' \cdot e^{j\varphi'}$ и $Z'' = 2^{\omega''} m'' \cdot e^{j\varphi''}$, то $Z'Z'' = 2^{\omega'+\omega''} m'm'' \cdot e^{j(\varphi'+\varphi'')}$. В этой группе предусмотрены следующие операции:

- Сложение - **AddLog**
- Вычитание - **SubLog**
- Инвертирование - **InvLog**

Алгебраическое сложение логарифмических форм выполняется по следующей схеме:

- Алгебраическое сложение порядков
- Алгебраическое сложение мантисс

8.3.5. Умножение логарифмической формы на целое число - **PowerLogar**

При положительном целом числе эта операция эквивалентна возведению в целую степень. При отрицательном целом числе эта операция эквивалентна делению «1» на корень целой степени. Операция выполняется по следующей схеме:

- Умножение порядка логарифмической формы на целое число
- Умножение мантиссы логарифмической формы на целое число

В этой группе предусмотрены отдельно следующие операции:

- Возведение в квадрат - **QuadrLogar**
- Деление «1» на квадратный корень - **MinusQuadrLogar**

8.3.6. Переполнения

При алгебраическом сложении логарифмических форм и умножении логарифмической формы на целое число могут возникать *переполнения*.

- При переполнении логарифма $m = \log_2(|z|)$ на действительное целое число S имеем: $\log_2(|z|) = [s + \log_2(|z'|)]$ и $\omega' = [\omega + s]$, то-есть при возникновении переноса S из действительной части мантиссы этот перенос складывается с порядком
- При возникновении переноса S из мнимой части мантиссы из нее вычитается число $s\pi/8$.

8.4. Извлечение квадратного корня

8.4.1. Извлечение квадратного корня из комплексного числа

Извлечение квадратного корня основано на следующем: если

$X = \prod (1 + \rho^{-h})^{2 \cdot a_h}$, то $\sqrt{X} = \prod (1 + \rho^{-h})^{a_h}$. Алгоритм извлечения квадратного корня из комплексного числа состоит в следующем:

1. Дано число $Z = (-2)^k \cdot M$, где M - мантисса, k - порядок.
2. Приведение данного числа к виду $Z = (-2)^{2m} \cdot a \cdot M$. При этом,
если k - четное, то $2m=k$, $a=1$;
если k - нечетное, то $2m=k-1$, $a=-1$.
3. Декомпозиция по формуле $X = \prod (1 + \rho^{-h})^{2 \cdot a_h}$ - "*DecompBinom2*".
4. Композиция по формуле $W = \prod (1 + \rho^{-h})^{a_h}$ - "*CompBinom*".
5. Формирование результата $\sqrt{Z} = (-2)^m Y$.
6. Нормализация результата.

8.4.2. Извлечение квадратного корня из сопряженного числа

Алгоритм извлечения квадратного корня из сопряженного комплексного числа отличается от алгоритма извлечения квадратного корня из комплексного числа только тем, что в п. 5 вместо композиции "*CompBinom*" используется композиция "*CompBinomConjug*":

5. Композиция по формуле $W = \prod (1 + \tilde{\rho}^{-h})^{a_h}$ - "*CompBinomConjug*".

8.4.3. Извлечение корня из положительного действительного числа

Алгоритм извлечения квадратного корня из положительного действительного числа состоит в следующем:

1. Дано число $Z = (-2)^k \cdot M$, где M - мантисса, k - порядок.

2. Декомпозиция по формуле $X = \prod (1 + \rho^{-h})^{2 \cdot ah}$ -
“DecompBinom2real”.
3. Композиция по формуле $W = \prod (1 + \rho^{-h})^{ah}$ - *“CompBinom”*.
4. Формирование результата $\sqrt{Z} = (-2)^m Y$.
5. Нормализация результата.

8.5. Полярные координаты

Ниже описываются алгоритмы вычисления полярных координат, а также некоторые вспомогательные алгоритмы.

8.5.1. Вычисление модуля комплексного числа

Алгоритм вычисления модуля комплексного числа отличается от алгоритма извлечения квадратного корня из комплексного числа только тем, что в п. 5 вместо композиции “*CompBinom*” используется композиция “*CompBinomModul*”:

5. Композиция по формуле $W = \prod \left[(1 + \rho^{-h}) (1 + \tilde{\rho}^{-h}) \right]^{a_h}$ - “*CompBinomModul*”.

Это следует из формулы $|Z| = \sqrt{Z \cdot \tilde{Z}}$

8.5.2. Вычисление аргумента комплексного числа (вариант 1)

Алгоритм вычисления аргумента комплексного числа отличается от алгоритма логарифмирования по варианту 1 только тем, что в п. 3 вместо композиции “*CompLogar*” используется композиция “*CompLogarAngle*”:

3. Композиция по формуле $W = -j \cdot \text{Im} \sum a_h \left[\ln(1 + \rho^{-h}) \right]$ - “*CompLogarAngle*”.

Это следует из формулы $\arg(Z) = -j \cdot \text{Im}(\ln(Z))$.

8.5.3. Вычисление аргумента комплексного числа (вариант 2) - AngleSqr

Вычисление аргумента комплексного числа основано на формуле $\arg(Z) = -j \cdot \text{Im} \left(2 \cdot \ln(\sqrt{Z}) \right)$. Алгоритм состоит в следующем:

1. Дано число $Z = (-2)^k \cdot M$, где M - мантисса, k - порядок.
2. Приведение данного числа к виду $Z = (-2)^{2m} \cdot a \cdot M$. При этом,
если k - четное, то $2m=k$, $a=1$;
если k - нечетное, то $2m=k+1$, $a=-1/2$.
3. Преобразование $X = f_1(a \cdot M)$, где X находится в первом полуквадранте – см. операцию «Сегментация перед извлечением корня».

4. Декомпозиция по формуле $X = \prod (1 + \rho^{-h})^{2 \cdot a_h}$ -
“*DecompBinom2*”.
5. Композиция по формуле $W = -2j \cdot \text{Im} \sum a_h [\ln(1 + \rho^{-h})]$ -
“*CompLogar.Angle*”.
6. Преобразование $Y = f_3(W_a)$ - см. операцию «Сегментация для логарифмирования». При этом $Y = \text{Im}(\ln(M))$.
7. Вычисление $\arg(Z) = \text{Im}(\ln(Z)) = \text{Im}[2m \cdot \ln(-2) + \ln(M)]$.
При этом $\ln(-2) = \ln(-1) + \ln(2) = j\pi + \ln(2)$. Таким образом, $\arg(Z) = Y + 2mj\pi$. Слагаемое, кратное 2π , можно отбросить. Поэтому $\arg(Z) = Y$.
8. Нормализация аргумента $\arg(Z) = Y$

8.5.4. Вычисление полярных координат - CartesianToPolar

Преобразование прямоугольных координат в полярные эквивалентно вычислению аргумента и модуля комплексного числа. Соответствующий алгоритм состоит в следующем:

1. Дано число $Z = (-2)^k \cdot M$, где M - мантисса, k - порядок.
2. Приведение данного числа к виду $Z = (-2)^{2m} \cdot a \cdot M$. При этом,
если k - четное, то $2m=k+2$, $a=1/4$;
если k - нечетное, то $2m=k+3$, $a=-1/8$.
3. Преобразование $X = f_1(a \cdot M)$, где X находится в первом полуквадранте – см. операцию «Сегментация перед извлечением корня».
4. Декомпозиция по формуле $X = \prod (1 + \rho^{-h})^{2 \cdot a_h}$ -
“*DecompBinom2*”.
5. Композиция по формуле $W_a = -2j \cdot \text{Im} \sum a_h [\ln(1 + \rho^{-h})]$ -
“*CompLogar.Angle*”.
6. Композиция по формуле $W_m = \prod [(1 + \rho^{-h})(1 + \tilde{\rho}^{-h})]^{a_h}$ -
“*CompBinomModul*”.
7. Преобразование $V = f_2(W_m)$ – см. операцию «Сегментация после извлечения корня».

8. Формирование результата в виде $|Z| = 2^{m+3} V$ - см. пункт 2.
9. Преобразование $Y = f_3(W_a)$ - см. операцию «Сегментация для логарифмирования». При этом $Y = \text{Im}(\ln(M))$.
10. Вычисление $\arg(Z) = \text{Im}(\ln(Z)) = \text{Im}[k \cdot \ln(-2) + \ln(M)]$.
При этом $\ln(-2) = \ln(-1) + \ln(2) = j\pi + \ln(2)$. Таким образом, $\arg(Z) = Y + kj\pi$.
11. Определение главного значения аргумента. В нем мнимая часть находится в пределах $-\pi \leq \text{Im } Z \leq \pi$. Поэтому главное значение аргумента определяется по формуле:

$$g = \text{int}\left[\frac{\arg Z}{2\pi}\right], (\arg Z)_{\text{main}} = \begin{cases} g & \text{if } g \leq \pi, \\ g - 2\pi & \text{if } g > \pi. \end{cases}$$

8.5.5. Возврат из полярных координат - PolarToCartesian

Преобразование полярных координат в прямоугольные состоит в вычислении по формуле $X = |X| \cdot e^{j\varphi}$, где $|X|$, φ - полярные координаты, действительные числа.

Алгоритм состоит в следующем:

1. Дано число $X = |X| \cdot e^{j\varphi}$, представленное мантиссой m и экспонентой p . При этом

$$|X| = (-2)^p \cdot \text{Re}(m), \quad \text{Im}(m) = j\varphi / 16.$$

2. Если $|X| = 0$, то $Z = 1$.
3. Формирование числа $j\varphi = (-2)^4 \cdot \text{Im}(m)$.
4. Вычисление $e^{j\varphi}$ (операция *Ort* - см. ниже).
5. Умножение $Z = |X| \cdot e^{j\varphi}$.

8.5.6. Вычисление синуса и косинуса действительного числа - Ort

Эта задача состоит в определении числа $Z = e^{jX}$, где действительное число $X = (-2)^p m$, m - мантисса, p - порядок.

Получаемое в результате число $Z = \cos X + j \sin X$. Это вычисление во многом аналогично потенцированию.

Преобразуем данное число $jX = j\left(\frac{\pi}{2}\mu + 2x''\right)$. В этом соотношении

$$\mu - \text{целая часть от частного } X'' = \frac{X}{\pi/2}.$$

$$2x'' - \text{дробная часть от частного } X'' = \frac{X}{\pi/2}.$$

Отсюда следует, что

$$\exp(jX) = \exp\left[j\left(\frac{\pi}{2}\mu + 2x''\right)\right] = e^{2(jx'')}e^{j\frac{\pi}{2}\mu} = e^{2y}\theta, \quad \text{где}$$

$$y = (jx''), \quad \theta = e^{j\frac{\pi}{2}\mu}. \quad \text{При этом } \theta = \begin{cases} j & \text{if } \eta = 1 \\ 1 & \text{if } \eta = 0 \\ -j & \text{if } \mu = -1 \\ -1 & \text{if } \mu = -2 \end{cases}, \quad \text{где}$$

$\eta = \{-2, -1, 0, 1\}$ - остаток от деления целого числа μ на 4.

Алгоритм вычисления состоит в следующем:

1. Дано число $X = (-2)^p m$, где m - мантисса, p - порядок.
2. Если $X = 0$, то $Z = 1$.
3. Вычисление $X'' = \frac{X}{\pi/2}$.
4. Выделение из числа X'' целой μ и дробной x'' частей соответственно.
5. Вычисление дробных частей $y = x''\frac{\pi}{4}$ и формирование кода числа y , который НЕ содержит экспоненты.

6. Декомпозиция числа y по формуле $y = \sum a_h \ln(1 + \rho^h)$ - “*DecompLogar*”.
7. Композиция по формуле $W = \prod (1 + \rho^{-h})^{a_h}$ - “*CompBinom*”.
8. Вычисление величины θ по вышеприведенной формуле в зависимости от младших разрядов кода числа μ .
9. Определение $\exp(X) = \theta \cdot W^2$
10. Нормализация результата

8.5.7. Определение полуквантанта - Semiquadrant

В этой операции определяется номер полуквантанта, в котором находится комплексное число – см. рис. 1.

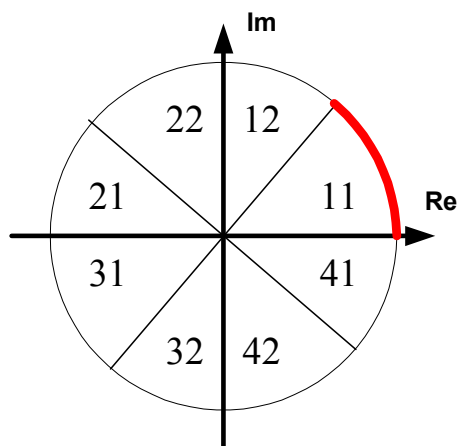


Рис. 1.

8.5.8. Сегментация перед извлечением корня - SemiquadrantTransOne

В этой операции выполняется такое преобразование комплексного числа $X = |X|e^{j\varphi}$, при котором оно перемещается в полуквантант 11 – см. таблицу 1. В дальнейшем это преобразование будем обозначать как $f_1(X)$. При этом преобразовании число X преобразуется в число $X' = |X|e^{j\vartheta}$ с аргументом $45^\circ \geq \vartheta \geq 0$. В табл. 1 обозначено:

Q - номер квадранта,

q - номер полуквантанта.

Таблица 1.

Q	q	$\text{Re}X$	$\text{Im}X$	$ \text{Re} < > \text{Im} $	$f_1(X)$
1	11	$\text{Re}X > 0$	$\text{Im}X > 0$	$ \text{Re} > \text{Im} $	Без изменений
	12			$ \text{Re} < \text{Im} $	Зеркальное отображение относительно биссектрисы первого квадранта
2	21	$\text{Re}X < 0$	$\text{Im}X > 0$	$ \text{Re} > \text{Im} $	Зеркальное отображение относительно оси ординат
	22			$ \text{Re} < \text{Im} $	Поворот на (-90)
3	31	$\text{Re}X < 0$	$\text{Im}X < 0$	$ \text{Re} > \text{Im} $	Поворот на (-180)
	32			$ \text{Re} < \text{Im} $	Зеркальное отображение относительно биссектрисы четвертого квадранта
4	41	$\text{Re}X > 0$	$\text{Im}X < 0$	$ \text{Re} > \text{Im} $	Зеркальное отображение относительно оси абсцисс
	42			$ \text{Re} < \text{Im} $	Поворот на $(+90)$

8.5.9. Сегментация после извлечения корня - SemiquadrantTransTwo

В этой операции выполняется преобразование комплексного числа $\sqrt{X'} = \sqrt{|X|} \cdot e^{j\varphi/2}$, где $X' = f_1(X)$. При этом преобразовании число $\sqrt{X'}$ перемещается в тот полуквадрант, где его аргумент равен $\frac{\varphi}{2}$, т.е. число принимает значение $\sqrt{X} = \sqrt{|X|} \cdot e^{j\varphi/2}$ – см. табл. 2. В дальнейшем это преобразования будем обозначать как $f_2(X)$. В табл. 2 обозначено:

Q - номер квадранта,

q - номер полуквадранта.

Таблица 2. Функции преобразования $f_1 \rightarrow f_2$.

Q	q	$\operatorname{Re} X$	$\operatorname{Im} X$	$ \operatorname{Re} < > \operatorname{Im} $	$f_2(X)$
1	11	$\operatorname{Re} X > 0$	$\operatorname{Im} X > 0$	$ \operatorname{Re} > \operatorname{Im} $	Без изменений
	12			$ \operatorname{Re} < \operatorname{Im} $	Поворот на (+45) и зеркальное отображение относительно биссектрисы первого квадранта
2	21	$\operatorname{Re} X < 0$	$\operatorname{Im} X > 0$	$ \operatorname{Re} > \operatorname{Im} $	Зеркальное отображение относительно биссектрисы первого квадранта
	22			$ \operatorname{Re} < \operatorname{Im} $	Поворот на (+45)
3	31	$\operatorname{Re} X < 0$	$\operatorname{Im} X < 0$	$ \operatorname{Re} > \operatorname{Im} $	Поворот на (-90)
	32			$ \operatorname{Re} < \operatorname{Im} $	Поворот на (+135) и зеркальное отображение относительно биссектрисы второго квадранта
4	41	$\operatorname{Re} X > 0$	$\operatorname{Im} X > 0$	$ \operatorname{Re} > \operatorname{Im} $	Зеркальное отображение относительно оси абсцисс
	42			$ \operatorname{Re} < \operatorname{Im} $	Поворот на (-45)

8.5.10. Сегментация для логарифмирования - SemiquadrantTransThree

В этой операции выполняется преобразование комплексного числа

$$\ln X' = \ln \left(\sqrt{|X|} \cdot e^{j\vartheta/2} \right) = \ln(\sqrt{|X|}) + j\vartheta/2, \quad \text{где}$$

$X' = f_1(X)$. При этом образуется число

$$\ln X = 2 \cdot \ln \left(\sqrt{|X|} \cdot e^{j\vartheta/2} \right) = \ln(|X|) + j\varphi.$$

Преобразование описывается табл. 3. В дальнейшем это преобразования будем обозначать как $f_3(X)$. В табл. 3 обозначено:

Q - номер квадранта,

q - номер полуквадранта.

Таблица 3. Функции преобразования $f_1 \rightarrow f_3$.

Q	q	$\operatorname{Re} X$	$\operatorname{Im} X$	$ \operatorname{Re} \lessgtr \operatorname{Im} $	$f_3(X)$
1	11	$\operatorname{Re} X > 0$	$\operatorname{Im} X > 0$	$ \operatorname{Re} > \operatorname{Im} $	$\operatorname{Re} f_3(X) = 2 \operatorname{Re} X$ $\operatorname{Im} f_3(X) = 2 \operatorname{Im} X$
	12			$ \operatorname{Re} < \operatorname{Im} $	$\operatorname{Re} f_3(X) = 2 \operatorname{Re} X$ $\operatorname{Im} f_3(X) = \frac{\pi}{2} - 2 \operatorname{Im} X$
2	21	$\operatorname{Re} X < 0$	$\operatorname{Im} X > 0$	$ \operatorname{Re} > \operatorname{Im} $	$\operatorname{Re} f_3(X) = 2 \operatorname{Re} X$ $\operatorname{Im} f_3(X) = \pi - 2 \operatorname{Im} X$
	22			$ \operatorname{Re} < \operatorname{Im} $	$\operatorname{Re} f_3(X) = 2 \operatorname{Re} X$ $\operatorname{Im} f_3(X) = \frac{\pi}{2} + 2 \operatorname{Im} X$
3	31	$\operatorname{Re} X < 0$	$\operatorname{Im} X < 0$	$ \operatorname{Re} > \operatorname{Im} $	$\operatorname{Re} f_3(X) = 2 \operatorname{Re} X$ $\operatorname{Im} f_3(X) = -\pi + 2 \operatorname{Im} X$
	32			$ \operatorname{Re} < \operatorname{Im} $	$\operatorname{Re} f_3(X) = 2 \operatorname{Re} X$ $\operatorname{Im} f_3(X) = -\frac{\pi}{2} - 2 \operatorname{Im} X$
4	41	$\operatorname{Re} X > 0$	$\operatorname{Im} X > 0$	$ \operatorname{Re} > \operatorname{Im} $	$\operatorname{Re} f_3(X) = 2 \operatorname{Re} X$ $\operatorname{Im} f_3(X) = -2 \operatorname{Im} X$
	42			$ \operatorname{Re} < \operatorname{Im} $	$\operatorname{Re} f_3(X) = 2 \operatorname{Re} X$ $\operatorname{Im} f_3(X) = -\frac{\pi}{2} + 2 \operatorname{Im} X$

8.6. Операции с полярными формами

8.6.1. Полярная форма представления комплексного числа

На рис. 1 представлена полярная форма комплексного числа $2^{\omega} \cdot D \cdot \exp(j\varphi)$. Эта форма используется для представления комплексного числа в полярных координатах, где

$2^{\omega} \cdot D$ – модуль, действительное число,

φ – действительное число, главное значение аргумента, которое находится в пределах $(-\pi) \leq \varphi \leq \pi$.

φ

63	62	54	53	52	2	1	0
Экспонента ω				Логарифм мантиисы M					

Рис. 1.

При этом

- в экспоненте записывается ω ,
- в реальной части мантиисы записывается $\text{Re } M = D$,
- в мнимой части мантиисы записывается число

$$j\phi = -\frac{j\varphi}{8}; \text{ при этом } \frac{-\pi}{8} \leq \phi \leq \frac{\pi}{8}.$$

8.6.2. Умножение показательных форм

В этой операции операнды представлены в показательной форме вида $2^{\omega} \cdot D \cdot \exp(j\varphi)$, т.е. в полярных координатах.

Умножение двух чисел, представленных в такой форме состоит в следующем:

- умножение реальных частей мантиис – модулей;
- сложение экспонент;
- сложение мнимых частей мантиис – аргументов; при возникновении переноса S из мнимой части результирующей мантиисы из нее вычитается число $-\pi/4$.

8.6.3. Поворот показательной формы

В этой операции первый операнд представлен в показательной форме вида $A = 2^{\omega} \cdot D \cdot \exp(j\varphi)$ - см выше. Второй операнд представлен в виде мнимой части мантииссы, которая имеет величину ϑ и представляет угол поворота ψ в виде $j\vartheta = -\frac{j\psi}{8}$,

где $-\frac{\pi}{8} \leq \vartheta \leq \frac{\pi}{8}$. Таким образом, второй операнд также представлен в показательной форме числа $\exp(j\psi)$. Поворот комплексного числа A на угол ψ состоит в следующем:

- передача действительной части мантииссы операнда A в действительную часть мантииссы результата;
- передача экспоненты операнда A в экспоненту результата;
- сложение мнимых частей мантиисс первого и второго операндов; при возникновении переноса S из мнимой части результирующей мантииссы из нее вычитается число $-\pi/4$.

8.7. Сложные функции

Последовательность элементарных функций, выполненная в виде подпрограммы или макрокоманды, позволяет вычислить более сложные функции. Для иллюстрации в табл. 7.1 перечислены тригонометрические и гиперболические функции, которые могут быть вычислены таким образом. В табл. 7.2 перечислены функции, которые выше определены как элементарные, но могут быть также вычислены через другие элементарные функции. В этой таблице φ - аргумент комплексного числа, действительное число.

Таблица 7.1.

$\operatorname{Sin} Z = \frac{j}{2} (e^{-jZ} - e^{jZ})$	$\operatorname{arcSin} Z = -j \ln \left(jZ + \sqrt{1 - Z^2} \right)$
$\operatorname{Cos} Z = \frac{j}{2} (e^{-jZ} + e^{jZ})$	$\operatorname{arcCos} Z = -j \ln \left(Z + \sqrt{Z^2 - 1} \right)$
$\operatorname{Tg} Z = j \left(\frac{e^{-jZ} - e^{jZ}}{e^{jZ} + e^{-jZ}} \right)$	$\operatorname{arSh} Z = \ln \left(Z + \sqrt{1 + Z^2} \right)$
$\operatorname{Ctg} Z = j \left(\frac{e^{jZ} + e^{-jZ}}{e^{jZ} - e^{-jZ}} \right)$	$\operatorname{arCh} Z = \ln \left(Z + \sqrt{Z^2 - 1} \right)$
$\operatorname{Sh} Z = \frac{1}{2} (e^Z - e^{-Z})$	$\operatorname{arcTg} Z = -\frac{j}{2} \ln \frac{1 + jZ}{1 - jZ}$
$\operatorname{Ch} Z = \frac{1}{2} (e^Z + e^{-Z})$	$\operatorname{arcCtg} Z = \frac{j}{2} \ln \frac{jZ + 1}{jZ - 1}$
$\operatorname{Th} Z = j \left(\frac{e^{-Z} - e^Z}{e^Z + e^{-Z}} \right)$	$\operatorname{arTh} Z = \frac{1}{2} \ln \frac{1 + Z}{1 - Z}$
$\operatorname{Cth} Z = j \left(\frac{e^{-Z} + e^Z}{e^Z - e^{-Z}} \right)$	$\operatorname{arCth} Z = \frac{1}{2} \ln \frac{Z + 1}{Z - 1}$

Таблица 7.2

$\arg(Z) = -j \cdot \text{Im}(\ln(Z))$	Вычисление аргумента комплексного числа
$\sqrt{Z} = (\sqrt{Z})$	Извлечение квадратного корня из сореженного числа
$\ln(Z) = \text{Re} \ln(Z)$	Вычисление логарифма модуля комплексного числа
$e^{j\varphi} = e^{\text{Im}[\ln Z]}$	Орт - вариант 1
$e^{-j\varphi} = e^{-\text{Im}[\ln Z]}$	Сопреженный орт - вариант 1
$e^{j\varphi} = \sqrt{Z/\tilde{Z}}$	Орт - вариант 2
$e^{-j\varphi} = \sqrt{\tilde{Z}/Z}$	Сопреженный орт - вариант 2
$ Z = e^{\text{Re}[\ln Z]}$	Модуль - вариант 2
$ Z = \sqrt{\tilde{Z} \bullet Z}$	Модуль - вариант 3
$\log_A B = \ln B / \ln A$	Логарифмирование
$A^B = e^{B \cdot \ln A}$	Возведение в степень
$X = (-b \pm \sqrt{b^2 - 4ac})/2a$	Квадратный трехчлен
$e^\varphi = \text{Re}(e^{j\varphi}) + \text{Im}(e^{j\varphi})$	Орт с данным углом
$\ln Z ; \varphi = -j \cdot \text{Im}(\ln Z)$	Преобразование обычной формы числа 'Z' в логарифмическую форму
$Z = e^{\ln Z + j\varphi}$	Преобразование логарифмической формы числа 'Z' в обычную форму

В качестве отдельных команд могут использоваться команды декомпозиции и композиции. Они позволяют составлять пользовательские программы для вычисления составных функций. Эти функции являются суммой таких элементарных функций, которые вычисляются как композиции. В табл. 7.3 приведены примеры вычисления составных функций относительно аргумента $W = |W|e^{j\varphi}$.

Перед такими вычислениями необходимо проверить, что величина $W = |W|e^{j\varphi}$ находится в пределах, допустимых для данного разложения. В противном случае эта величина должна быть приведена в допустимый интервал, подобно тому, как это сделано в операциях логарифмирования, потенцирования, извлечения корня и т.д. Алгоритмы для таких вычислений соответствует алгоритму вычисления функции, приведенному выше.

Таблица 7.3

Составная функция	Пояснение
$Z = f(W, \ln W, \varphi, \ln W , W , \sqrt{W}, \sqrt{\widetilde{W}})$	Декомпозиция $D4$ $W = \prod (1 + \rho^{-h})^2$
$Z = f(W, \ln W, \varphi, \ln W)$	Декомпозиция $D3$ $W = \prod (1 + \rho^{-h})$
$Z = f(W, \exp W)$	Декомпозиция $D2$ $W = \sum \ln(1 + \rho^{-h})$

Глава 9. Решение уравнений

9.1. Метод «цифра за цифрой» и трансцендентные уравнения.

В предыдущей главе рассматривался метод вычисления функций W аргумента Z , заданных в явной форме $W = f(Z)$. Далее рассматривается метод вычисления функций, заданных в неявной форме $Z = F(W)$. Этот метод также использует *алгоритм вычисления функций* методом «цифра за цифрой», описанный выше.

По-прежнему, вычисление функции W будем связывать с опросом чисел E_p и E_n . По определению $E_p = Z - Z_p$ и $E_n = Z - Z_n$. Но $Z_p = F(W_p)$ и $Z_n = F(W_n)$. Следовательно, $E_p = Z - F(W_p)$ и

$$E_n = Z - F(W_n). \quad (1)$$

В частности, $E_o = Z - F(W_o)$.

Пусть

$$F(W) = F_1(W) + F_2(W) + \dots + F_k(W) + \dots + F_v(W). \quad (2)$$

Тогда $F(W_p) = \sum_{k=1}^v F_k(W_p)$ и $F(W_n) = \sum_{k=1}^v F_k(W_n)$.

При этом вычисление неявно заданной функции производится по *алгоритму вычисления функций*, описанному в главе «Вычисление функций комплексного переменного». Для определения E_n используется формула (1).

Изложенный метод вычисления неявных функций может быть использован для решения трансцендентных уравнений. Возможность и целесообразность решения трансцендентных уравнений этим методом, по-прежнему, определяется тем, существуют ли функции $\Delta\psi_k(x_h)$ и $\Phi(x_h)$ достаточно простого

вида. В частности, этим методом может быть решено уравнение вида

$$Z = \sum_k F_k(W). \quad (3)$$

содержащее такие функции $F_k(W)$, каждая из которых может быть вычислена методом «цифра за цифрой».

Пример 1. К решению трансцендентных уравнений относительно комплексной переменной могут быть сведены многие задачи навигации, сопровождения и преследования морских и воздушных целей. Очевидна необходимость быстрого решения таких задач. Пример подобной задачи приведен на рис. 1. Управляемая точка (самолет-истребитель) движется по траектории OACDEG. Неуправляемая точка (цель) одновременно движется по траектории FG и встречается с первой в точке встречи G. Необходимо найти параметры траектории управляемой точки, зная отношение скоростей преследователя и цели. Эта задача сводится к системе трансцендентных уравнений с комплексными числами. Для CAU она может быть сформулирована очень лаконично. Можно показать при этом, что решение этой задачи на CAU выполняется в 10-20 раз быстрее, чем решение соответствующей системы трансцендентных уравнений на традиционном компьютере.

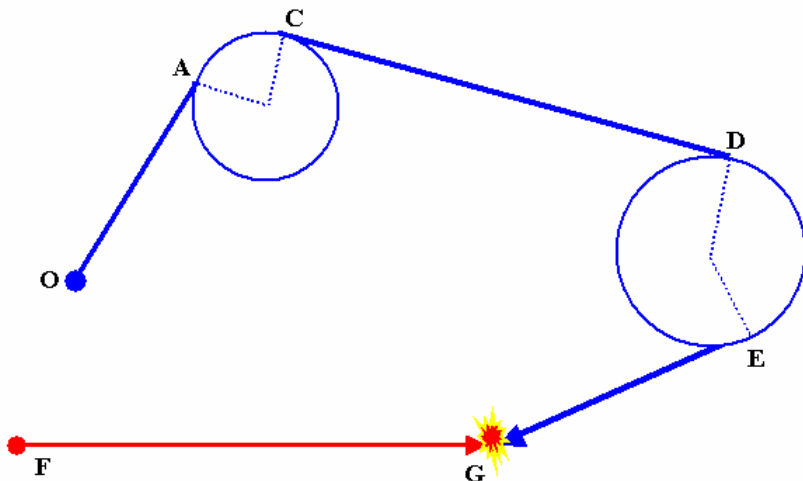


Рис. 1.

Итак, указанная задача сводится к решению уравнения

$$W + A \cdot e^{j\varphi} = B$$

относительно неизвестного $W = |W|e^{j\varphi}$, где A, B - известные комплексные числа. Это уравнение может быть преобразовано к виду

$$A \cdot |W| - B \cdot \overline{W} + W \cdot \overline{W} = 0$$

Обозначим $F_1(W) = A \cdot |W|$, $F_2(W) = -B \cdot \overline{W}$, $F_3(W) = W \cdot \overline{W}$.

Будем определять образующие искомой величины в виде

$W_n = W_p (1 + \rho^{-h})^2$, причем $W_o = 1$. Тогда

$$F_1(W_n) = (1 + (\overline{\rho})^{-h}) \cdot (1 + \rho^{-h}) \cdot F_1(W_p)$$

$$F_2(W_n) = (1 + (\overline{\rho})^{-h})^2 \cdot F_2(W_p)$$

$$F_3(W_n) = (1 + (\overline{\rho})^{-h})^2 \cdot (1 + \rho^{-h})^2 \cdot F_3(W_p),$$

причем $F_1(W_o) = A$, $F_2(W_o) = -B$, $F_3(W_o) = 1$. Величину W можно вычислять по алгоритму вычисления функций, описанному в главе «Вычисление функций комплексного переменного». При этом величину E_n следует вычислять согласно формуле (1): $E_n = -F(W_n)$, т.к. в этом уравнении $Z = 0$. Здесь $F(W_n)$ определяется по (2) и вычисление состоит только из коротких операций операций сравнения по модулю.

Пример 2. Решение уравнения

$$\varphi + a|W| + b\sqrt{W} + cW^2 = Z$$

относительно неизвестного $W = |W|e^{j\varphi}$, где A, B, C, Z - известные комплексные числа. Будем определять образующие

искомой величины в виде $W_n = W_p (1 + \rho^{-h})^2$, причем $W_o = 1$.

Обозначим: $F_1(W) = \varphi$, $F_2(W) = a|W|$, $F_3(W) = b\sqrt{W}$,

$F_4(W) = cW^2$. Тогда, используя результаты предыдущего параграфа, получим:

$$F_1(W_n) = F_1(W_p) - j \cdot \text{Im}(2 \cdot \ln(1 + \rho^{-h})),$$

$$F_2(W_n) = F_2(W_p) \left(1 + \rho^{-h} \right) \left(1 + \left(\rho^* \right)^{-h} \right),$$

$$F_3(W_n) = F_3(W_p) \left(1 + \rho^{-h} \right),$$

$$F_4(W_n) = F_4(W_p) \left(1 + \rho^{-h} \right)^4,$$

причем $F_1(W) = 0$, $F_2(W) = a$, $F_3(W) = b$, $F_4(W) = c$.

Выражение для E_n имеет вид:

$$E_n = Z - F_1(W_n) - F_2(W_n) - F_3(W_n) - F_4(W_n)$$

Это выражение содержит лишь операции вычитания, сложения и вычитания со сдвигом. Числа $j \cdot \text{Im} \left(2 \cdot \ln \left(1 + \rho^{-h} \right) \right)$ выбираются из датчика констант для логарифмирования.

Важно отметить, что сходимость вычислений должна исследоваться для каждой задачи.

9.2. Определение корней степенного полинома

Рассмотрим теперь степенной полином комплексного переменного W вида

$$Z = \sum_k a_k W^{s_k}, \quad (1)$$

где

a_k, Z - комплексные числа,

s_k - показатель степени, действительное дробное

положительное число.

Этот степенной полином является частным случаем уравнения (1.3). Поэтому корни полинома могут быть найдены предложенным методом. Рассмотрим этот вопрос подробнее. Пусть $s_k = r_k/d$, где r_k, d - целые положительные числа. Будем определять образующие неизвестного W в виде:

$$W_n = W_p (1 + \rho^{-h})^d,$$

причем $W_o = 1$. Обозначив $F_k(W) = a_k W^{r_k/d}$, находим

$$F_k(W_n) = (1 + \rho^{-h})^{r_k} F_k(W_p), \quad \text{причем} \quad F_k(W_o) = a_k.$$

Следовательно,

$$E_n = Z - \sum_k \left[(1 + \rho^{-h})^{r_k} F_k(W_p) \right].$$

Рассмотрим частный случай - степенной полином комплексного переменного W с целыми показателями степени ($d=1$). В этом случае будем определять образующие неизвестного W в виде:

$$W_n = W_p (1 + \rho^{-h}).$$

Для этого случая и иллюстрации всего вышеизложенного на рис. 2 представлен алгоритм вычисления корня степенного полинома (1) - функция `DecompCompBinomTwo`. На этом рисунке обозначено:

- $A = \{a_k\}$, $B = \{b_k\}$, $k = \overline{1, N}$, $Z = -a_0$,
- $\text{ModulesComparision}(E_{next}, E_{prev})$ – функция сравнения модулей чисел E_{next} и E_{prev} .

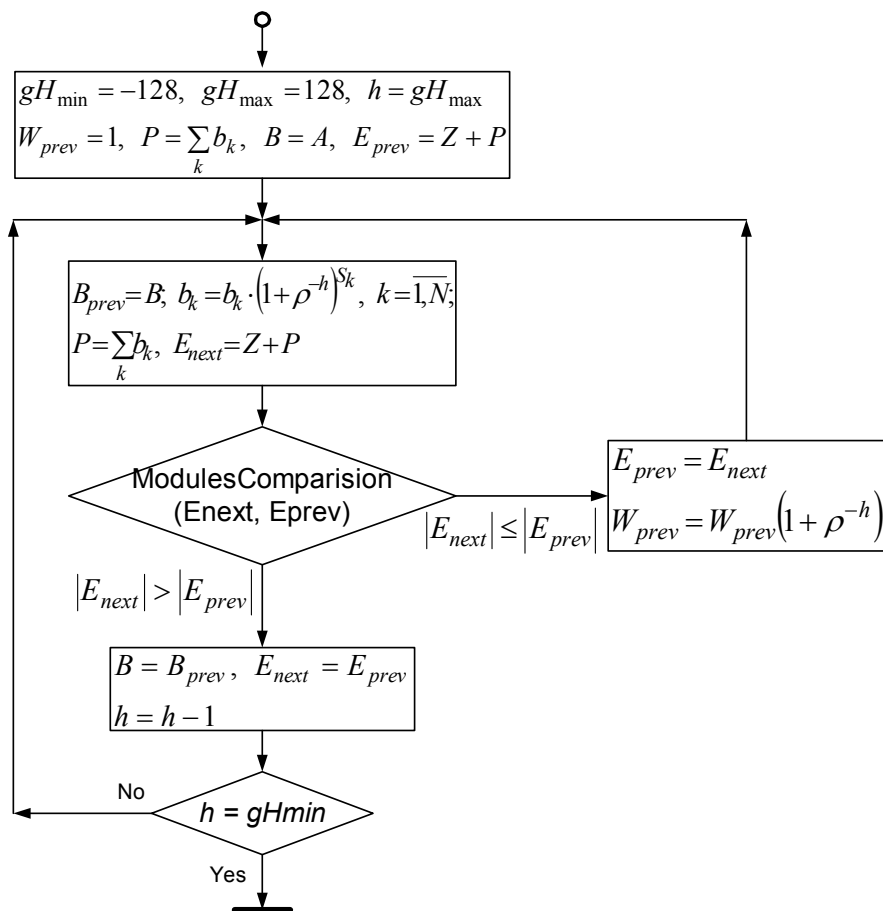


Рис. 2.

Рассмотрим вопрос *сходимости* вычислений. Предложенный метод можно трактовать как поиск минимума действительной функции M комплексного переменного W , где M – модуль значения полинома. Если найденный локальный минимум имеет нулевое значение, то он соответствует какому-то корню полинома. Метод сходится, если все локальные минимумы имеют нулевое значение. Для определения количества локальных минимумов заметим, что производная квадрата модуля полинома степени N по

комплексной переменной W является полиномом степени $(2N-1)$. Следовательно, функция M имеет $(2N-1)$ локальных экстремумов. Из них N экстремумов являются минимумами, поскольку функция M обязана иметь N локальных минимумов с нулевым значением. Минимумы должны перемежаться максимумами и этих максимумов должно быть не меньше числа минимумов (с точностью до 1). Следовательно, остальные $(N-1)$ экстремумов являются максимумами. Тем самым мы показали, что все минимумы имеют нулевое значение, т.е. соответствуют какому-либо корню.

В целом определение всех корней степенного полинома выполняется итеративно по следующей схеме:

- определение корня степенного полинома степени N ,
- понижение порядка полинома при известном корне,
- определение корня степенного полинома степени $(N-1)$,
- и т.д. до получения бинома - полинома степени 1,
- вычисление корня бинома $Z = a \cdot W$ делением:

$$W = Z/a.$$

Итак, для определения всех корней степенного полинома используются следующие специализированные операции:

- сравнение комплексных чисел по модулю,
- умножение на бином,
- деление комплексных чисел.

Вычисление корня полинома степени 1 вида $Z = a \cdot W$, а, следовательно, и деление $W = Z/a$, может быть выполнено по вышеприведенному алгоритму для степенного полинома произвольной степени. При этом алгоритм деления (функция `DecompCompBinomDelen`) принимает следующий вид – см. рис. 3.

Основное отличие этого алгоритма от алгоритма деления, приведенного в главе 3, состоит в следующем: в рассматриваемом алгоритме делимое W представляется в виде $W = \prod (1 + \rho^{-h})^{a_h}$, а в главе 3 делитель Z представляется в виде $\frac{1}{Z} = \sum a_h \rho^h$.

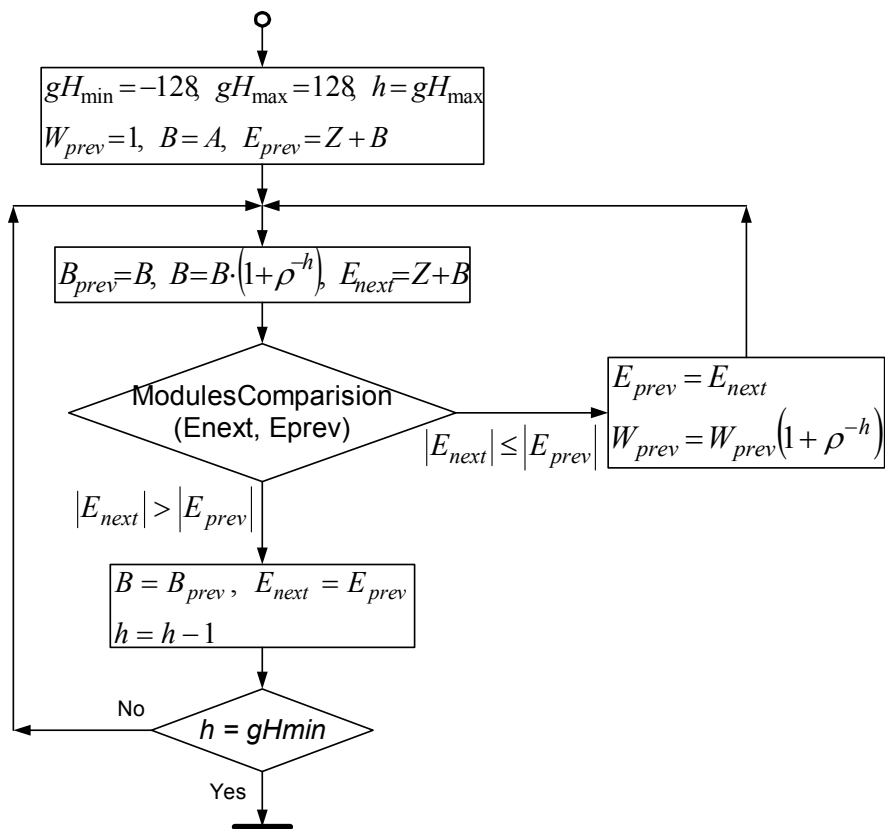


Рис. 3.

9.3. Дифференциальные уравнения

Один из методов решения системы дифференциальных уравнений основан на предварительном определении корней характеристического уравнения. Время решения характеристического уравнения составляет существенную часть общего времени решения системы дифференциальных уравнений. С другой стороны, характеристическое уравнение представляет собой степенной полином. Его корни могут быть определены вышеописанным способом. При этом количество исполняемых команд (как показывает моделирование) *уменьшается в 20-50 раз*. Такие характеристики позволяют нам предлагать САУ, как специализированный процессор для автоматического регулирования.

В системах автоматического регулирования широко используются вычисления с комплексными числами и решение систем дифференциальных уравнений. Поэтому можно рекомендовать использование комплексной арифметики в устройствах автоматического регулирования.

9.4. Решение квадратных уравнений

Для квадратного уравнения общего вида

$$ax^2 + bx + c = 0 \tag{1}$$

решение, как известно, имеет следующий вид:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

При комплексных коэффициентах это

выражение не требует каких-либо преобразований, если вычисления выполняются на комплексном процессоре, т.к. все операции с комплексными числами в этом выражении эквивалентны элементарным операциям комплексного процессора.

Таблица 4.7.1.

Тип	Уравнение	Ограничение	Второй корень
1	$ax^2 + x = 1$	$ a \leq 1, \operatorname{Re}(a) \geq 0$	$x_2 = -x_1 - \frac{1}{a}$
2	$x^2 + bx = 1$	$ b \leq 1, \operatorname{Im}(b) \geq 0$	$x_2 = -x_1 - b$
3	$ax^2 + x = -1$	$ a \leq 1$	$x_2 = -x_1 - \frac{1}{a}$
4	$x^2 + bx = -1$	$ b \leq 1$	$x_2 = -x_1 - b$
5	$ax^2 - x = 1$	$ a \leq 1$	$x_2 = -x_1 - \frac{1}{a}$
6	$x^2 - bx = 1$	$ b \leq 1, \operatorname{Im}(b) \geq 0$	$x_2 = -x_1 - b$
7	$ax^2 - x = -1$	$ a \leq 1, \operatorname{Re}(a) \leq 0$	$x_2 = -x_1 - \frac{1}{a}$
8	$x^2 - bx = -1$	$ b \leq 1$	$x_2 = -x_1 - b$

Существуют некоторые типы квадратных уравнений, для которых решение может быть получено методом «цифра за цифрой». Для них могут быть предусмотрены соответствующие команды. Такие квадратные уравнения, называемые далее аппаратно-решаемыми квадратными уравнениями, перечислены в нижеследующей таблице 4.7.1. В этой таблице указаны ограничения, которым должны удовлетворять коэффициенты квадратного уравнения для того, чтобы итерационный процесс метода «цифра за цифрой»

сходился. Этот метод позволяет вычислить один из корней. Вторым корнем вычисляется по формуле, приведенной в этой таблице.

Обозначим корень уравнения k -типа через $x^{(k)}$. Вычитая уравнения типа 6, 8, 5, 7 из уравнений типа 2, 4, 1, 3 соответственно, замечаем, что $x^{(2)} - x^{(6)} = -b$, $x^{(4)} - x^{(8)} = -b$, $x^{(1)} - x^{(5)} = -\frac{1}{a}$,

$x^{(3)} - x^{(7)} = -\frac{1}{a}$. В связи с этим пары уравнений типа (1,5), (2,6),

(3,7), (4,8) будем называть *сопряженными*. Решение уравнения некоторого типа может быть получено из решения уравнения сопряженного типа. Этим фактом можно воспользоваться следующим образом:

если некоторое уравнение не удовлетворяет ограничению на знак действительной или мнимой части коэффициента, то можно решить сопряженное уравнение, у которого это ограничение отсутствует или является противоположным.

Рассмотрим теперь алгоритм преобразования квадратного уравнения общего вида к виду аппаратно-решаемого квадратного уравнения. Для уравнения вида (1) найдем величину $m^2 = ac/b^2$.

Если $|m^2| \leq 1$, то $|m| \leq 1$ и уравнение (1) может быть преобразовано к виду аппаратно-решаемого квадратного уравнения

$$a_1 y^2 \pm y = s, \quad s = \pm 1, \quad y = \pm \frac{sb}{c} x, \quad a_1 = -sm^2, \quad |a_1| \leq 1.$$

После его решения корень исходного уравнения вычисляется по формуле $x = \mp yc/sb$.

Если $|m^2| \geq 1$, то $|\frac{1}{m}| \leq 1$ и уравнение (1) может быть преобразовано к виду аппаратно-решаемого квадратного уравнения

$$z^2 \pm b_1 z = s, \quad s = \pm 1, \quad z = \mp x \sqrt{\frac{-sa}{c}}, \quad b_1 = \sqrt{\frac{-s}{m^2}}, \quad |b_1| \leq 1.$$

После его решения корень исходного уравнения вычисляется по формуле $x = \pm z \sqrt{-c/sa}$.

9.5. Решение трансцендентных уравнений методом Мюллера

Для решения трансцендентных уравнений вида $y(x) = 0$, где $y(x)$ - аналитическая функция комплексной переменной, широко используется метод Мюллера [50]. Этот метод состоит в следующем. Имеются три точки $\{x_{n-2}, x_{n-1}, x_n\}$ и соответствующие им значения функции $\{y_{n-2}, y_{n-1}, y_n\}$. Составляется единственная квадратичная функция $z(x)$. В качестве x_{n+1} берется тот из корней этой функции, который ближе к x_n . Далее операции повторяются с тремя последними точками. Нужно иметь в виду, что в случае сходимости метода старший коэффициент функции $z(x)$ стремится к нулю, и в ближайшей окрестности корня необходим переход к линейной интерполяции. Задание хороших начальных точек требует знания специфики задачи. Чтобы найти новое решение нужно задать новые начальные точки. В качестве одной из них целесообразно выбрать точку, сопряженную с ранее найденным решением.

Из описания метода следует, что его программная реализация использует следующие операции

1. Вычисление $y(x)$; сложность этих вычислений целиком определяется видом данной функции;
2. Формирование коэффициентов квадратичной функции $z(x)$ по трем парам комплексных чисел; эта задача сводится, как известно к решению трех линейных уравнений с комплексными коэффициентами;
3. Решение квадратного уравнения $z(x) = 0$; решение этой задачи рассмотрено выше;
4. Вычитание комплексных чисел; это – короткая операция для комплексного процессора;
5. Сравнение комплексных чисел по модулю; это – короткая операция для комплексного процессора.

Глава 10. Моделирование устройства извлечения квадратного корня из комплексных чисел

1.1. Введение

В этой главе описывается устройство извлечения квадратного корня из комплексных чисел. На основе вышеописанного метода «цифра-за-цифрой» подробно рассматривается алгоритм извлечения квадратного корня из комплексных чисел и его моделирование в системе MATLAB. Рассматривается также моделирование в системе MATLAB некоторых известных алгоритмов вычисления по методу «цифра-за-цифрой». На основе сопоставления результатов моделирования делается сравнительный анализ быстродействия предлагаемого алгоритма извлечения квадратного корня из комплексного числа и возможных способов традиционного выполнения этой же операции.

Показано, что по количеству тактов предлагаемый алгоритм сравним с алгоритмами вычисления элементарных функций действительного аргумента. Очевидно, вычисление корня квадратного из комплексного числа можно выполнить в DSP и PC по программе, в которой применяются операции вычисления элементарных функций действительного аргумента. По количеству тактов в таких программах предлагаемый алгоритм в 2.5 - 4.5 раза короче.

Приведенной информации (вместе с некоторыми специальными схемами, описанными в части 3) достаточно для практической реализации устройства.

Программы для этого раздела находятся в каталоге CD/Root/, за исключением программ для раздела , которые находятся в каталоге CD/Trad/.

1.2. Извлечение квадратного корня

1.2.1. Композиции

Далее мы применим метод «цифра за цифрой» к вычислению корня квадратного из комплексного числа – см. также раздел 8.4.1 в части 1. Рассмотрим разложение комплексного числа в следующем виде:

$$Z = \prod (1 + \rho^{-h})^{2 \cdot ah}, \quad (1)$$

где ρ – основание системы счисления 1 – см. раздел 1 и формулу (1.3), откуда следует, что

$$\rho^h = \begin{cases} (-2)^{h/2} & \text{if } h - \text{even,} \\ j \cdot (-2)^{(h-1)/2} & \text{if } h - \text{odd.} \end{cases} \quad (2)$$

Очевидно,

$$\sqrt{Z} = \prod (1 + \rho^{-h})^{ah}, \quad (3)$$

и

$$q\sqrt{Z} = q \cdot \prod (1 + \rho^{-h})^{ah}, \quad (3a)$$

Рассмотрим еще разложение комплексного числа в следующем виде:

$$Z = g \cdot \prod (1 + \rho^{-h})^{2 \cdot ah}. \quad (3b)$$

Очевидно, что в этом случае

$$\sqrt{\frac{Z}{g}} = \prod (1 + \rho^{-h})^{ah}, \quad (3c)$$

В дальнейшем будем применять разложения (1, 3b, 3, 3a, 3c) соответственно в следующем виде:

$$Z = \prod_{H=H \min}^{H \max} (1 + \rho^{-H})^{2 \cdot aH} \quad (4)$$

$$Z = g \cdot \prod_{H=H \min}^{H \max} (1 + \rho^{-H})^{2 \cdot aH}, \quad (4a)$$

$$\sqrt{Z} = \prod_{H=H \min}^{H \max} (1 + \rho^{-H})^{aH} \quad (5)$$

$$q\sqrt{Z} = q \cdot \prod_{H=H \min}^{H \max} (1 + \rho^{-H})^{aH} \quad (5a)$$

$$\sqrt{\frac{Z}{g}} = \prod_{H=H \min}^{H \max} (1 + \rho^{-H})^{aH} \quad (5b)$$

Результат разложения представляет собой множество чисел

$$Accum = \{ \alpha^{H \min}, \dots, \alpha^H, \dots, \alpha^{H \max} \}. \quad (6)$$

По известному множеству (6) может быть выполнена композиция числа Z . На фиг. 1 представлена блок-схема алгоритма композиции по формуле (5), где приняты следующие обозначения:

W_{prev} — предыдущее значение результата,

W_{next} — следующее значение результата,

DW — приращение,

k — счетчик значения разряда α^H массива $Accum$,

R — текущее значение разряда α^H ,

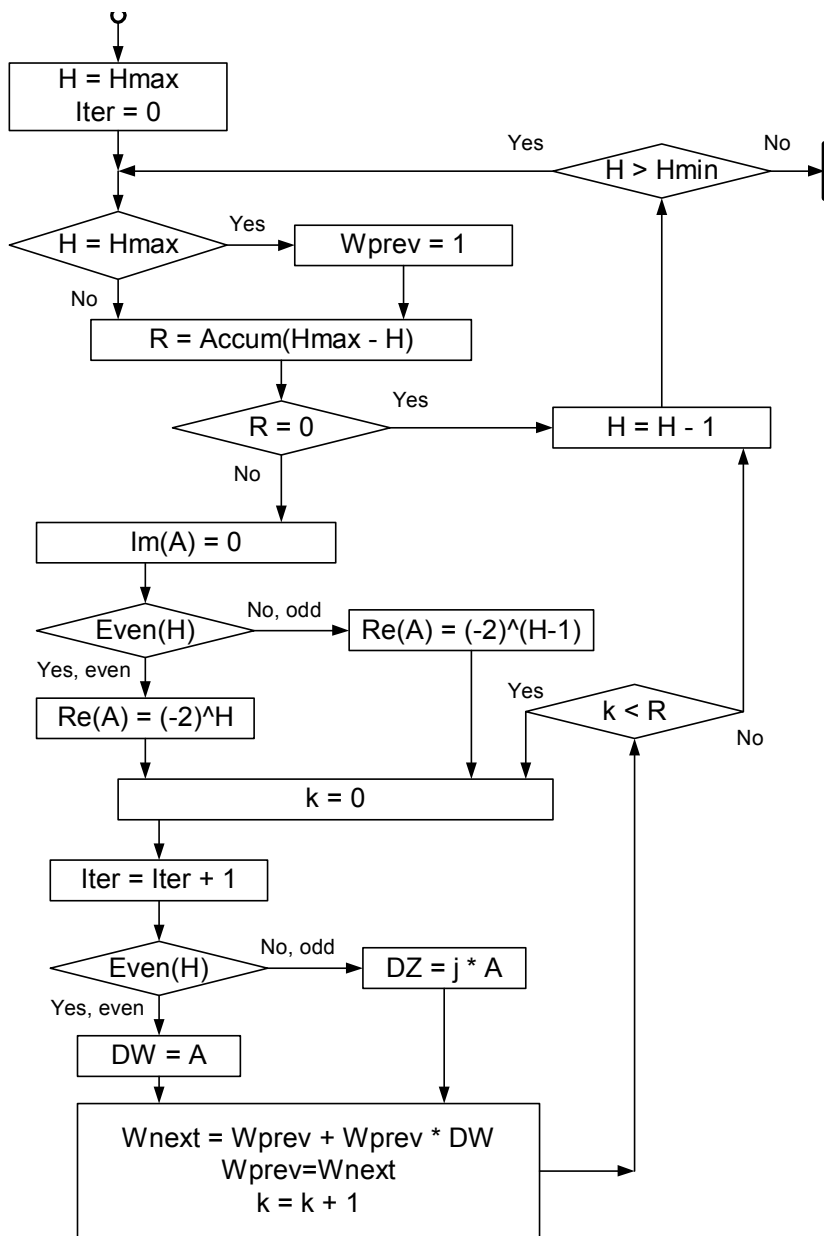
H — текущий номер разряда, $H \min \leq H \leq H \max$

$Even$ — функция проверки четности действительного целого положительного числа H ,

$Iter$ — счетчик итераций,

A — промежуточный результат (комплексное число).

Этот же алгоритм реализован в функции **CompBinom1**. В ней используется функция **MultBinom0**, которая умножает комплексное число на бином $(1 + \rho^{-h})$. Именно простота аппаратной реализации этой функции предопределяет применение указанных разложений для комплексных чисел.



Фиг. 1.

1.2.2. Декомпозиции

Декомпозиция для вычисления корня квадратного находит коэффициенты (6) разложения (1). Блок-схема алгоритма декомпозиции представлена на фиг. 2, где приняты следующие обозначения (в дополнение к обозначениям для алгоритма композиции):

E_{prev} – предыдущее значение остатка,

E_{next} – следующее значение остатка,

$Compar$ – функция сравнения комплексных чисел по модулю, которая возвращает значение «1», если $E_{next} < E_{prev}$ и «0», если $E_{next} > E_{prev}$,

DZ – приращение,

A, B – промежуточные результаты (комплексные числа).

Функция $DecBin2$ реализует алгоритм декомпозиции комплексных чисел в разложение (4а) и композиции по (5а). В ней используются функция $MultBinom0$, описанная выше, и функция $CNcompare$, которая вычисляет модуль комплексного числа для последующего сравнения модулей остатков E_{next} и E_{prev} . Используются принятые выше обозначения и следующие:

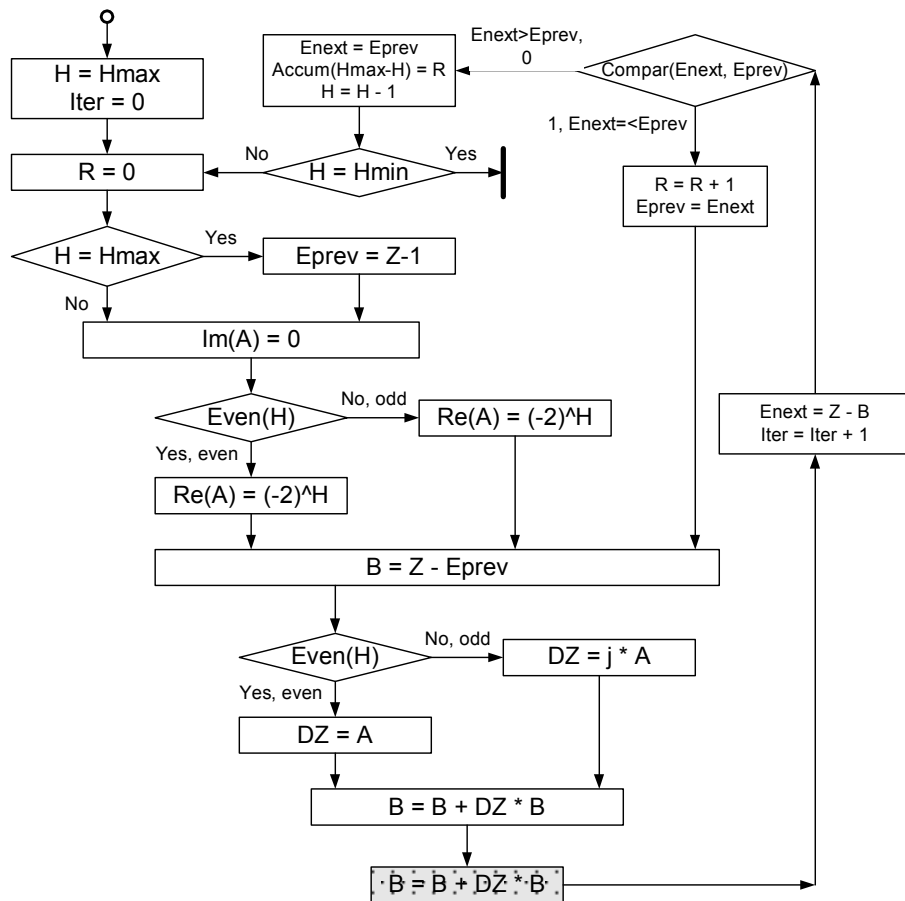
\sqrt{Z} – число, вычисляемое по алгоритму композиции параллельно с декомпозицией,

$Iter$ – номер итерации,

$IterPlus$ – номер удачной итерации,

DM – разность модулей остатков E_{next} и E_{prev} после очередной итерации $Iter$,

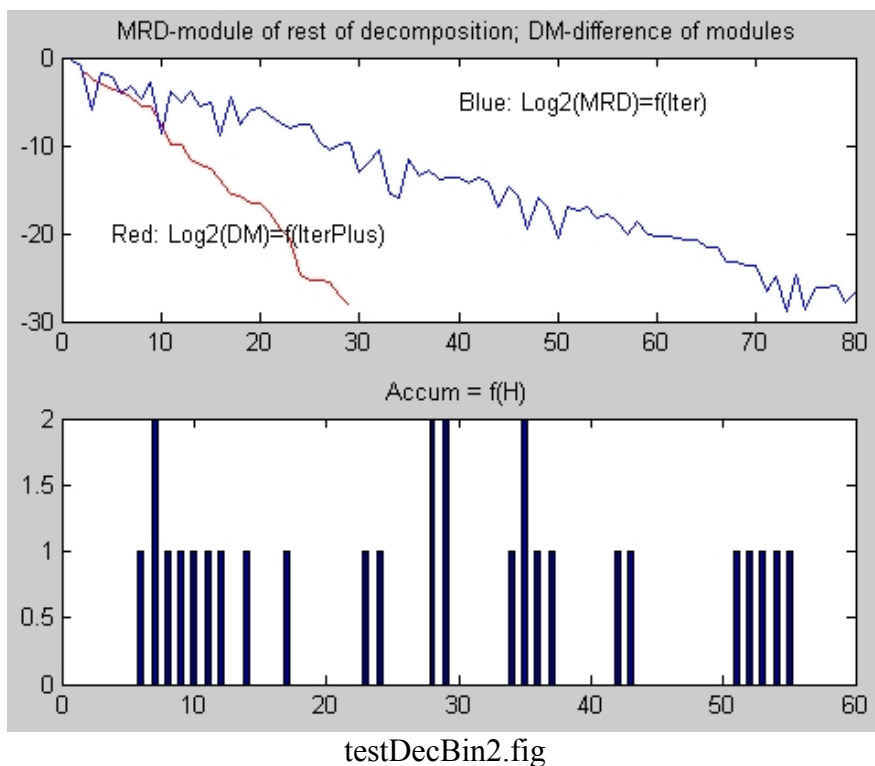
MRD – модуль следующего значения остатка E_{next} после очередной удачной итерации $IterPlus$.



Фиг. 2.

Функция `testDecBin2` позволяет при обращении к функции `DecBin2` анализировать алгоритм декомпозиции. В этой функции

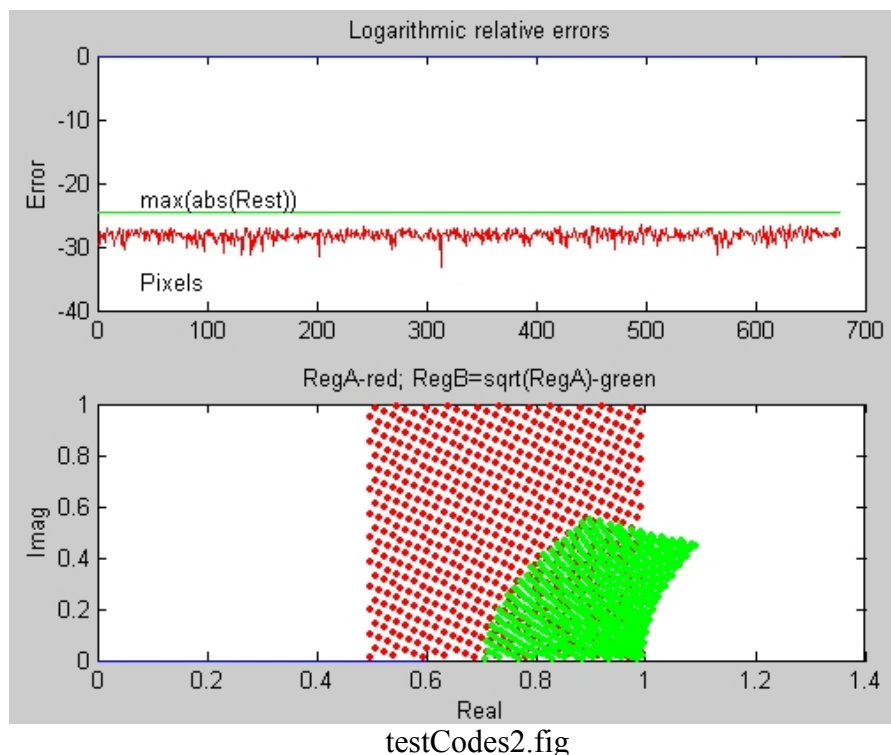
- вычисляется логарифмическая ошибка извлечения квадратного корня, т.е. количество верных разрядов `er`,
- строятся зависимости `DM(Iter)` и `MRD(IterPlus)` в первом окне - см. фиг. `testDecompBinom2.fig`,
- строится гистограмма значений разрядов массива (6) во втором окне - см. фиг. `testDecBin2.fig`.



Функция `testCodes2` перебирает все коды заданной разрядности в определенной области и выполняет декомпозицию этих кодов и композицию путем обращения к функции `DecBin2`.

Для ускорения проверки функция вычисляет только определенную выборку из всех кодов. Количество кодов в выборке равно *pixels*. Функция строит

- зависимость относительной логарифмической ошибки декомпозиции и максимальную абсолютную ошибку остатка – см. первое окно на фиг. `testCodes2.fig`,
- область исходных чисел и область корней из этих чисел
область исходных чисел и область корней из этих чисел
– см. второе окно на фиг. `testCodes2.fig`,



testCodes2.fig

1.2.3. Область представления чисел.

На фиг. 3 изображены числа – биномы вида $b_h = (1 + \rho^{-h})$, где ρ определено по (4.2). Пусть $h > 0$ и $h = 4k + m$, где $m = \{0, 1, 2, 3\}$.

На фиг. 1

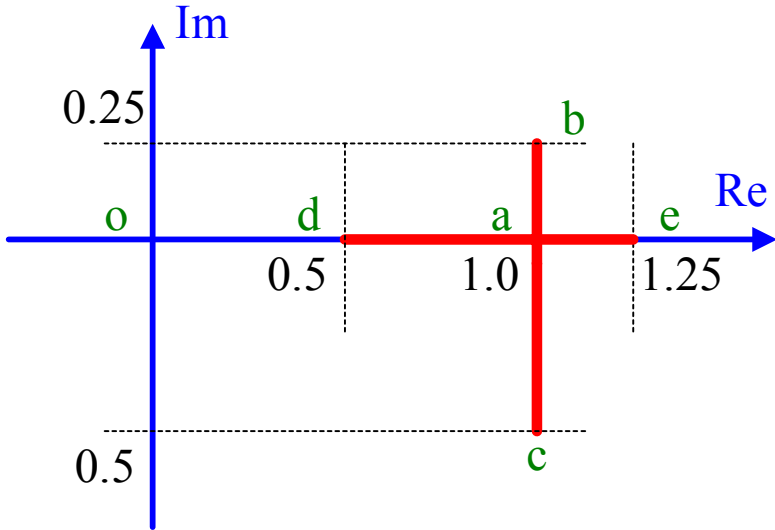
ae - область изменения биномов при $m=0$,

ac - область изменения биномов при $m=1$,

ad - область изменения биномов при $m=2$,

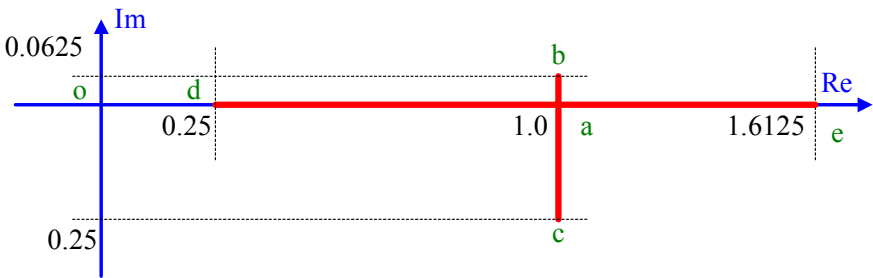
ab - область изменения биномов при $m=3$.

Отсюда следует, что любое комплексное число с реальной неотрицательной частью при $H_{\min} > 0$ может быть представлено произведением биномов (5), если кратность α^H вхождения биномов в это произведение не ограничена.



Фиг. 3

Аналогично рассмотрим квадраты биномов $b_h^2 = (1 + \rho^{-h})^2$. Области изменения этих биномов представлены на фиг. 4. Также любое комплексное число с реальной НЕотрицательной частью при $H_{min} > 0$ может быть представлено произведением биномов (4), если кратность α^H вхождения биномов в это произведение НЕ ограничена.



Фиг. 4

Вопрос заключается в том, насколько велика эта кратность, ибо для аппаратной реализации кратность должна быть мала. Для анализа кратности можно воспользоваться функцией `testDecBinAn`.

На следующих фигурах представлены некоторые области чисел для декомпозиции по формуле (4). Для каждого из чисел определено максимальное значение кратности `maxAccum`

вхождения квадратов биномов в разложение (4). Цвет точки указывается в зависимости от значения $\max\text{Accum}$ так:

- синий, если $\max\text{Accum} < 2$,
- красный, если $\max\text{Accum} = 2$,
- зеленый, если $\max\text{Accum} = 3$,
- черный, если $\max\text{Accum} > 4$.

Область чисел, подвергаемых декомпозиции, показывается в левом окне, а область чисел, получаемых композицией, показывается в правом окне. Таким образом, в левом окне показывается область левого окна, преобразованного по формуле \sqrt{Z} . Принимаются $H_{\min}=3$, $H_{\max}=55$. и вычисляются максимальная кратность для всей области $\max(\max\text{Accum})$ и точность декомпозиции er .

На фиг. f64_2dc в правом окне представлена область чисел вида $0.25 \leq x \leq 1, |y| \leq 1$.

В результате получены $\max(\max\text{Accum})=6$ и точность декомпозиции $\text{er}=5.6\text{e-}8$.

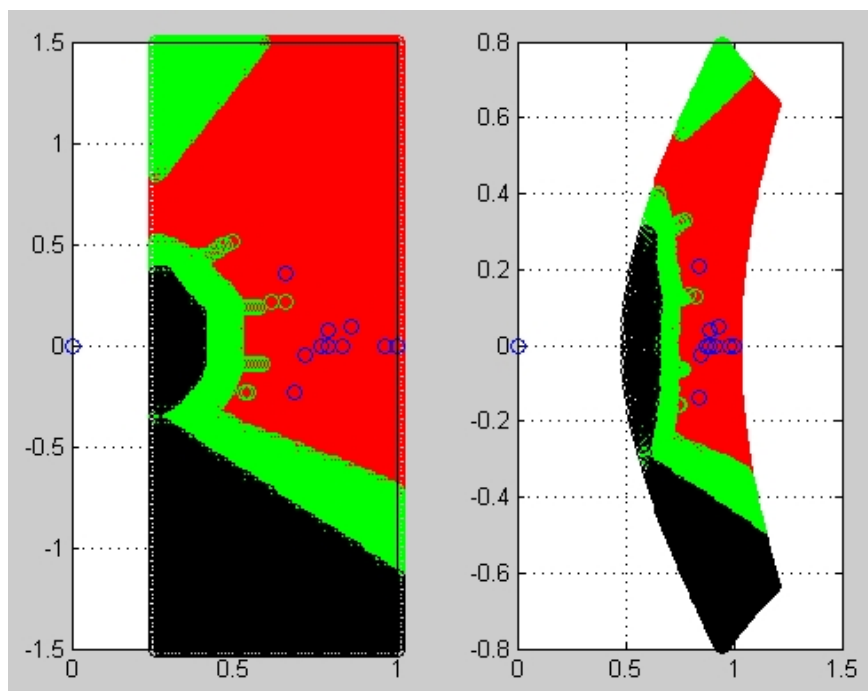
На фиг. f64_2fu в правом окне представлена область чисел вида $0.5 \leq x \leq 1, |y| \leq 1$.

В результате получены $\max(\max\text{Accum})=4$ и точность декомпозиции $\text{er}=4.5\text{e-}8$.

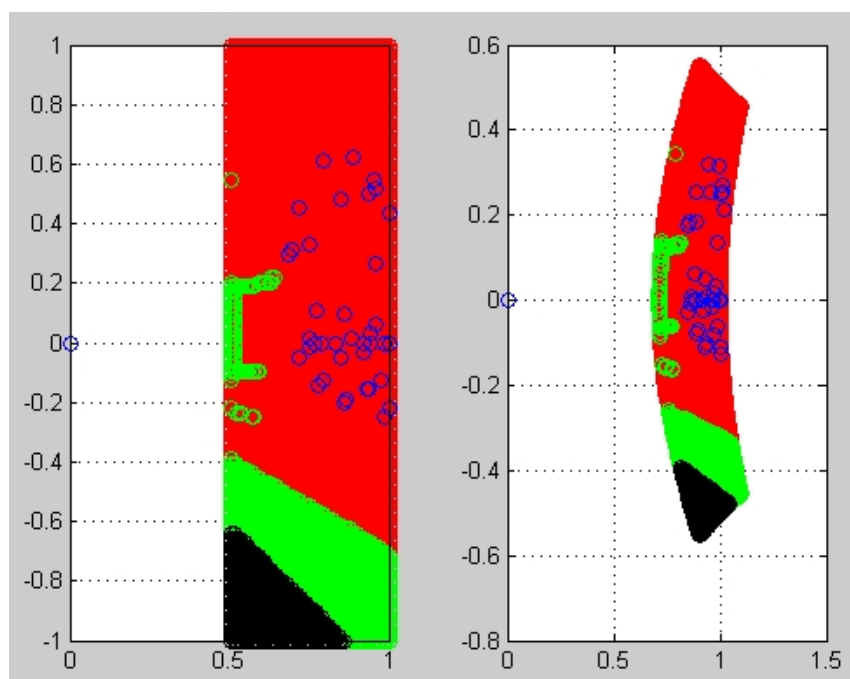
На фиг. f64_2od в левом окне представлена область чисел вида $0.5 \leq x \leq 1, 0 \leq y \leq 1$. (7)

В результате получены $\max(\max\text{Accum})=3$ и точность декомпозиции $\text{er}=4.5\text{e-}8$.

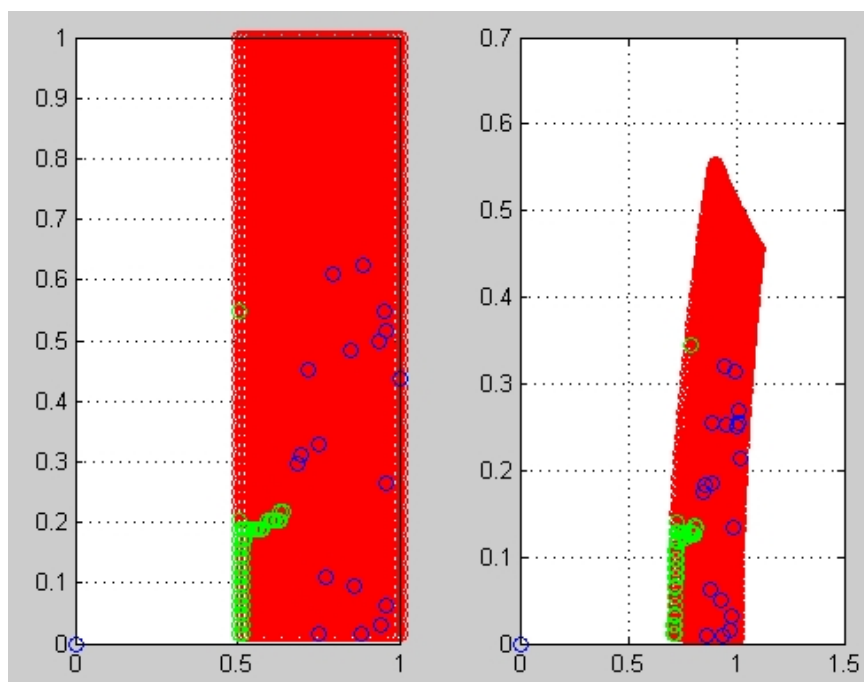
В отличие от предыдущего случая на фиг. f64_2od2 в правом окне показывается область левого окна, преобразованного по формуле $\sqrt{2Z}$ (а не \sqrt{Z}).



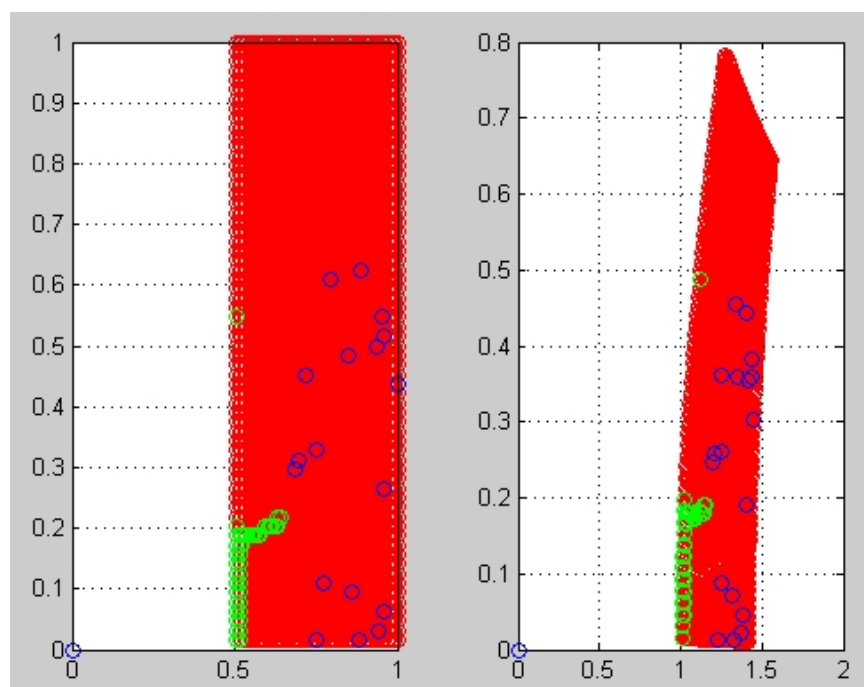
f64_2dc



f64_2fu



f64_2od



f64_2od2

Область (7) характеризуется наименьшей кратностью и поэтому используется в дальнейшем. Рассмотрим важные для дальнейшего свойства области (7):

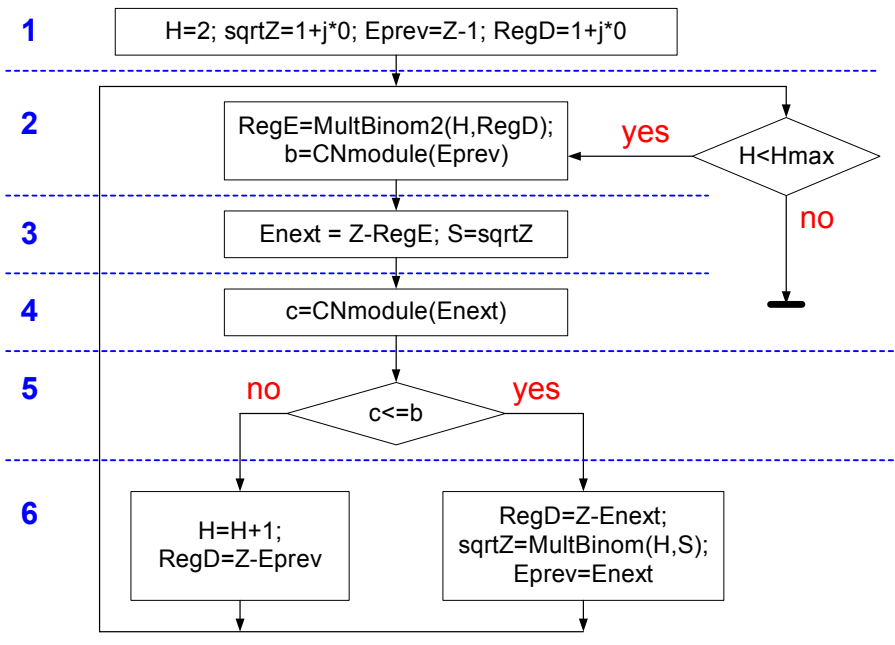
1. кратность биномов не более 3,
2. расположение в первом квадранте,
3. нормализованная величина реальной части исходного числа,
4. дробная величина реальной и мнимой части исходного числа,

$$0 \leq \text{Im}\sqrt{Z} < 1, \quad 0 \leq \text{Im}\sqrt{2Z} < 1. \quad (8)$$

6. дробная величина реальной части результирующих чисел
- $$0.5 < \text{Re}\sqrt{Z} < 2, \quad 0.5 < \text{Re}\sqrt{Z} < 2. \quad (9)$$

1.2.4. Оценка быстродействия.

Рассмотрим функцию `dcb`, которая отличается от функции `DecBin2` отсутствием всех операторов и переменных, необходимых для отображения и анализа результатов. Функция `testDCB` тестирует функцию `dcb`.



Фиг. 3.

На фиг. 3 функция **dcb** изображена в виде блок-схемы, где операторы, которые при аппаратной реализации могут выполняться параллельно, представлены в одном блоке. Цифрами на этой схеме обозначены номера тактов. По этой схеме можно оценить время выполнения декомпозиции в тактах. Каждая итерация выполняется за 5 тактов. Количество итераций

$$Iter \approx \frac{3}{2} H_{\max}$$

и

$$N = H_{\max}/2,$$

где N – число разрядов в коде мантиссы каждой части комплексного числа. Таким образом, количество тактов декомпозиции

$$T \approx 15N.$$

Например, для кода с 23-разрядной мантиссой $T \approx 350$, для кода с 16-разрядной мантиссой $T \approx 240$.

Отметим еще, что этот же алгоритм может быть применен для вычисления корня квадратного из действительного положительного числа. Декомпозиция такого числа содержит только биномы с действительными ρ^H – см. формулу (2). Поэтому при декомпозиции достаточно анализировать только элементы с четными значениями H и время декомпозиции сокращается вдвое.

Итак, *время вычисления корня квадратного из комплексного числа вдвое больше времени вычисления корня квадратного из действительного положительного числа.*

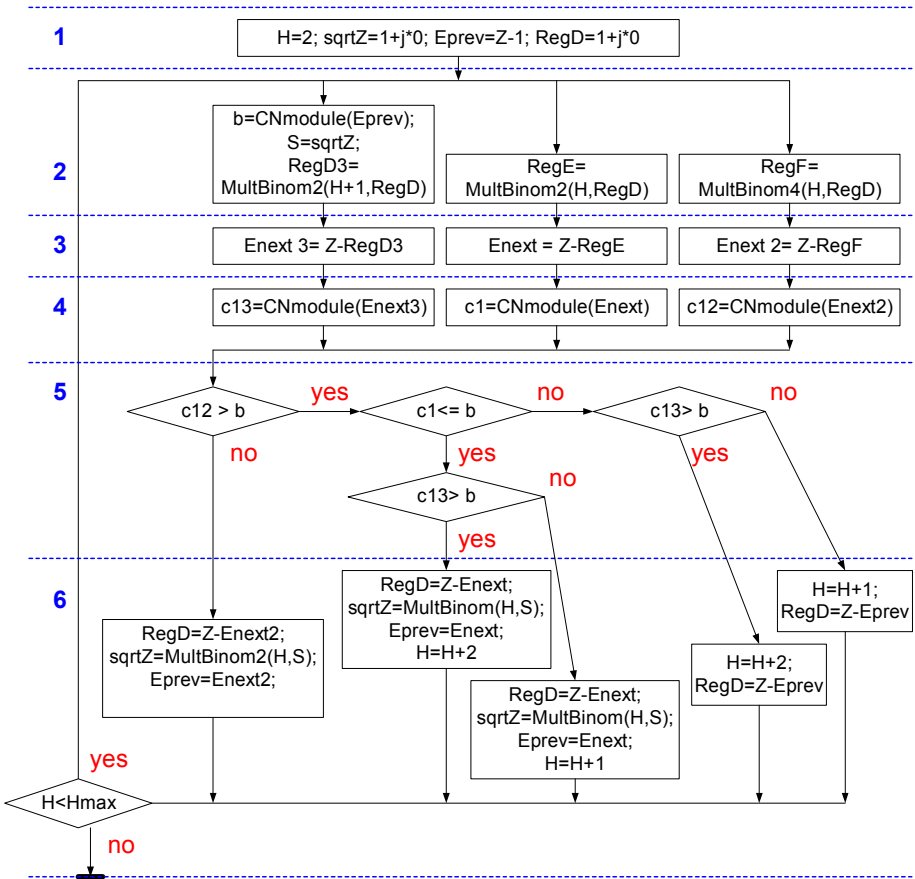
1.2.5. Оптимизация по быстродействию

Рассмотрим теперь возможные способы ускорения вычисления. В основном алгоритме делается пробное умножение на квадрат бинорма при данном H . *Первый способ* ускорения состоит в том, что выполняется пробное умножение в двух вариантах: на квадрат бинорма (что эквивалентно проверке условия $\alpha^H = 1$) и на четвертую степень бинорма (что эквивалентно проверке условия $\alpha^H = 2$). *Второй способ* ускорения состоит в том, что выполняется пробное умножение на квадрат следующего бинорма (что эквивалентно проверке условия $\alpha^{H+1} = 1$). Анализ всех этих вариантов позволяет *в более чем в два раза уменьшить количество итераций.*

Оба способа реализованы в функции **DecompBinom3**. В рассмотренных выше тестах она может заменить функцию **DecBin2**.

Рассмотрим функцию **deb3**, которая отличается от функции **DecompBinom3** отсутствием всех операторов и переменных, необходимых для отображения и анализа результатов. Функция **testDCB3** тестирует функцию **deb3**.

На фиг. 4 функция **deb3** изображена в виде блок-схемы, где операторы, которые при аппаратной реализации могут выполняться параллельно, представлены в одном блоке или в параллельно включенных блоках.



Фиг. 4.

Каждая итерация выполняется за 5 тактов. Количество итераций

$$Iter \approx 0.7 \cdot H_{\max}.$$

Таким образом, количество тактов декомпозиции

$$T \approx 7N.$$

Например, для кода с 23-разрядной мантиссой $T \approx 160$, для кода с 16-разрядной мантиссой $T \approx 110$.

Дальнейшего ускорения можно достигнуть, выполняя пробное умножение на квадрат бинорма, эквивалентного проверке условия $\alpha^{H+2} = 1$ и т.д. Практический эффект от такого прогноза быстро падает с увеличением глубины прогноза. Так, при $H_{\max} = 55$ и

- при отсутствии всех способов ускорения математическое ожидание числа итераций $\bar{it} = 80$ - см. функцию DecBin2,
- при отсутствии прогноза $\bar{it} = 59$,
- при ускорении по способу 1 и прогнозе одного бинорма $\bar{it} = 40$ - см. функцию DecomBinom3,
- при ускорении по способу 1 и прогнозе двух бинормов $\bar{it} = 33$ - см. функцию DecomBinom4,
- при ускорении по способу 1 и прогнозе трех бинормов $\bar{it} = 32$.

Таким образом, целесообразно остановиться на прогнозе двух бинормов. Поэтому рассмотрим функцию **dcb4**, которая отличается от функции DecomBinom4 отсутствием всех операторов и переменных, необходимых для отображения и анализа результатов. Функция testDCB4 тестирует функцию **dcb4**.

На фиг. 5 функция **dcb4** изображена в виде блок-схемы. В отличие от фиг. 4 на фиг. 5 переменные обозначены очевидными сокращенными именами.

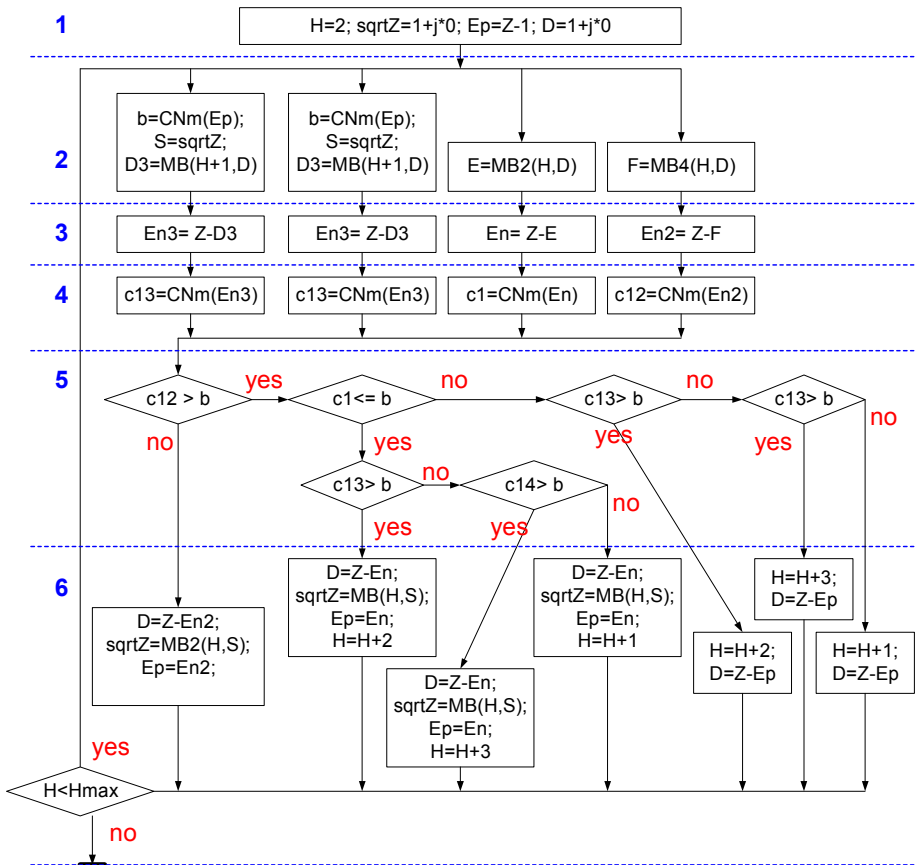
В функции **dcb4** каждая итерация также выполняется за 5 тактов. Количество итераций

$$Iter \approx 0.6 \cdot H_{\max}.$$

Таким образом, количество тактов декомпозиции

$$T \approx 6N.$$

Например, для кода с 23-разрядной мантиссой $T \approx 140$, для кода с 16-разрядной мантиссой $T \approx 90$.



Фиг. 5.

1.3. Sqrt - нормализация

Выше показано, что для декомпозиции (4.4) с ограниченной кратностью квадратов биномов комплексное число Z должно находиться в первом квадранте и удовлетворять условию (4.7). Таким образом, исходное число должно перед декомпозицией преобразовано так, чтобы удовлетворять этим условиям. Преобразование, которое переводит исходное число в то число, которое удовлетворяет этим условиям, будем называть *sqrt-нормализацией*.

Исходное число Z должно быть представлено в виде $Z = a + jb$, где

1. оба числа a и b имеют общую экспоненту $ExpZ$,
2. хотя бы одно из чисел a и b имеет нормализованную мантиссу.

Если при этом мантисса числа a не нормализована (что легко проверить по значению старшего разряда мантиссы), то исходное число преобразуется в число $Z' = a' + jb' = jZ$, где $a' = -b$, $b' = a$. Итак,

$$Z' = v \cdot Z, \text{ when } v = \begin{cases} 1, & \text{if } |a| \geq 0.5, \\ j, & \text{if } |a| < 0.5. \end{cases} \quad (1)$$

Далее, если число $a' < 0$, то число Z' преобразуется в число $Z'' = -Z'$, т.е.

$$Z'' = w \cdot Z', \text{ when } w = \begin{cases} 1, & \text{if } a' \geq 0, \\ -1, & \text{if } a' < 0. \end{cases} \quad (2)$$

Очевидно, число Z'' удовлетворяет условию (4.7) и его можно разложить по формуле (4.4).

Экспонента $ExpZ$ исходного числа Z при извлечении корня уменьшается вдвое. Однако, если она нечетная, то оставшийся после деления экспоненты на 2 множитель 2 превращается в множитель $\sqrt{2}$ результата. Итак, результат с экспонентой X и мантиссой $ExpX$ определяется по следующим формулам:

$$ExpX = \begin{cases} \frac{ExpZ}{2}, & \text{if } ExpZ - \text{even}, \\ \frac{ExpZ - 1}{2}, & \text{if } ExpZ - \text{odd}, \end{cases} \quad (4)$$

$$X = \begin{cases} \sqrt{s \cdot Z''}, & \text{if } ExpZ - \text{even}, \\ \sqrt{2s \cdot Z''}, & \text{if } ExpZ - \text{odd}, \end{cases} \quad (5)$$

где

$$s = \frac{1}{v \cdot w}. \quad (6)$$

В следующей таблице приведены значения числа s :

	w=1	w=-1
v=1	1	j
v=j	j	-j

Выше показано, что результат (или) может превышать «1» - см. фиг. f64_2od и f64_2od2. Для получения нормализованного результата увеличим на 1 экспоненту $ExpX$ и уменьшим вдвое мантиссу X . Тогда формулы (4) и (5) примут следующий вид:

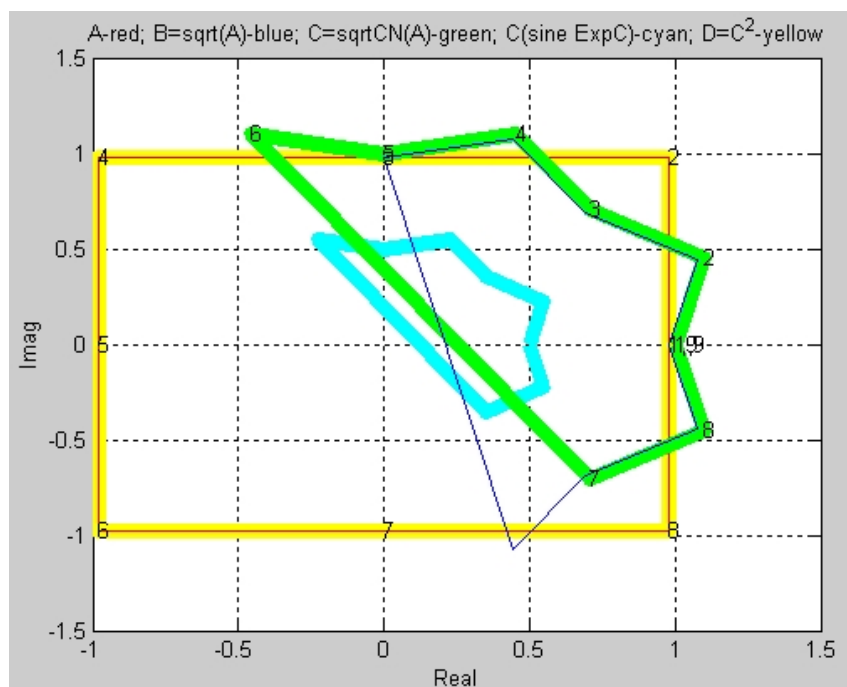
$$ExpX = \begin{cases} 1 + \frac{ExpZ}{2}, & \text{if } ExpZ - \text{even}, \\ \frac{ExpZ + 1}{2}, & \text{if } ExpZ - \text{odd}, \end{cases} \quad (7)$$

$$X = \begin{cases} \sqrt{0.25s \cdot Z''}, & \text{if } ExpZ - \text{even}, \\ \sqrt{0.5s \cdot Z''}, & \text{if } ExpZ - \text{odd}. \end{cases} \quad (8)$$

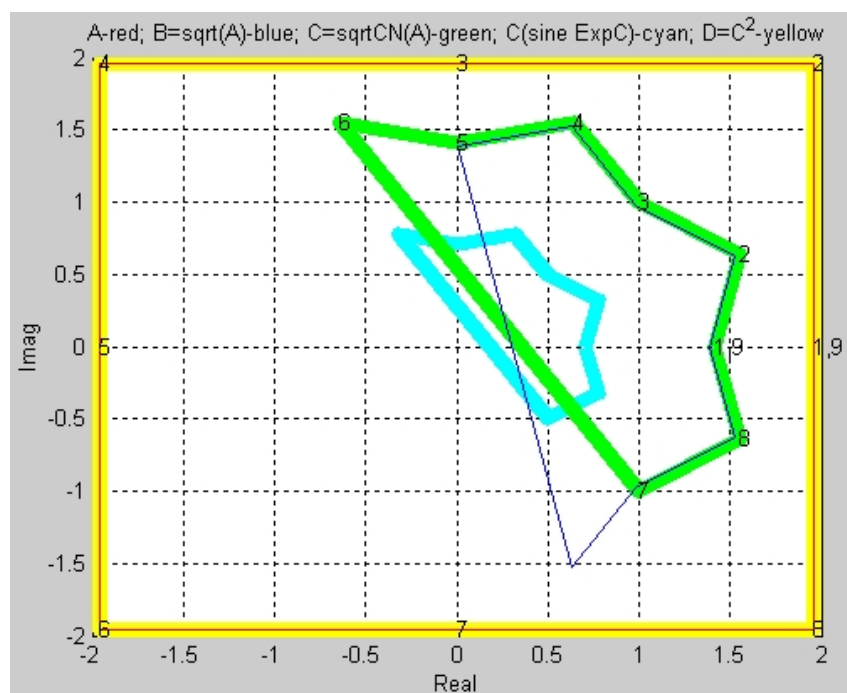
Таким образом, композицию результата необходимо выполнять по формуле (4.5a), где

$$q = \begin{cases} \sqrt{0.25s}, & \text{if } ExpZ - \text{even}, \\ \sqrt{0.5s}, & \text{if } ExpZ - \text{odd}. \end{cases} \quad (9)$$

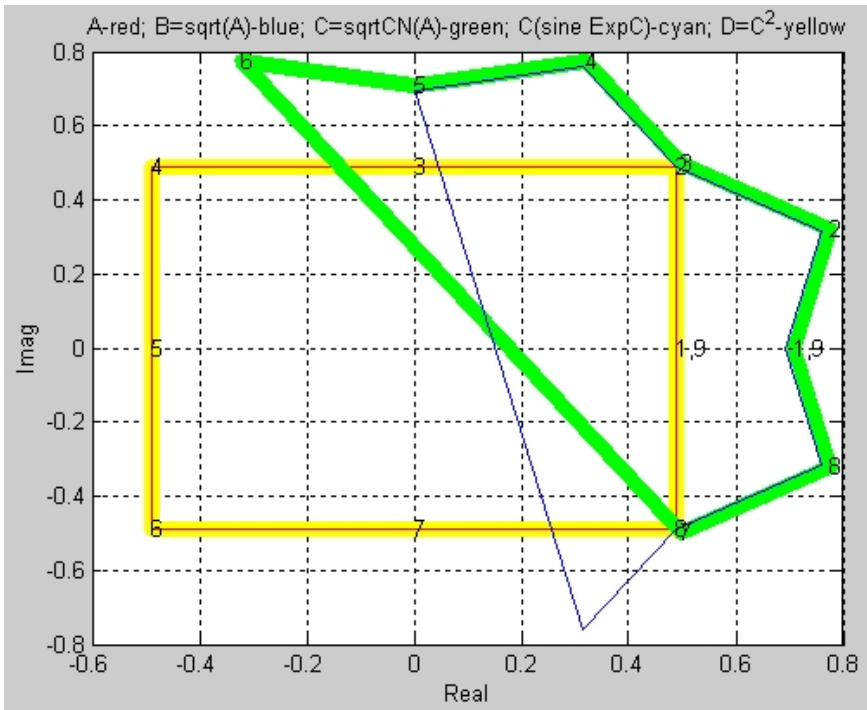
Функция `SqrtComp` выполняет вычисление по этим формулам.



testSquare2.fig, ExpZ=0



testSquare2.fig, ExpZ=1



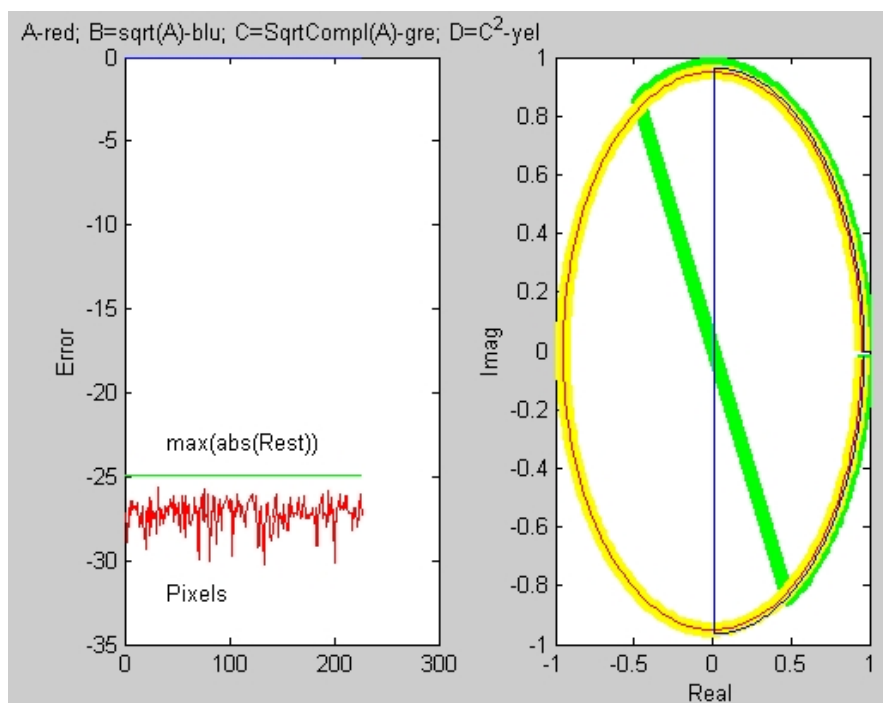
testSquare2.fig, ExpZ=-1

Функция `testSquare2` извлекает (при обращении к функции `SqrComp`) квадратный корень из комплексных чисел – точек, лежащих на определенном квадрате. Для контроля эта же операция выполняется традиционным способом. На фиг. `testSquare2.fig` показан результат вычислений при различных значениях экспоненты *ExpZ*. Здесь

- красная фигура – исходные числа A,
- синяя фигура – корень B из числа A, вычисленный традиционно,
- зеленая фигура – корень C из числа A, вычисленный предлагаемым методом,
- желтая фигура – квадрат D числа C,
- голубая фигура – мантисса числа C; здесь видно, что мантисса всегда находится в области нормализованных чисел (на последнем рисунке голубая фигура совпадает и полностью закрыта зеленой фигурой).

Функция `testOkrug2` извлекает (при обращении к функции `SqrComp`) квадратный корень из комплексных чисел – точек, лежащих на окружности. Для контроля эта же операция выполняется традиционным способом. На фиг. `testOkrug2.fig` показан результат вычислений. Функция строит

- зависимость относительной логарифмической ошибки декомпозиции и максимальную абсолютную ошибку остатка – см. первое окно на фиг. `testOkrug2.fig`,
- исходную окружность и полуокружность, полученную после извлечения квадратного корня – см. второе окно на фиг. `testOkrug2.fig`.



testOkrug2.fig

1.2.4. О сходимости

Рассмотрим вопрос *сходимости* вычислений. Предложенный метод можно трактовать как поиск минимума действительной функции M комплексного переменного $W=x+jy$, где M – модуль функции

$$f(W) = W^2 - Z,$$

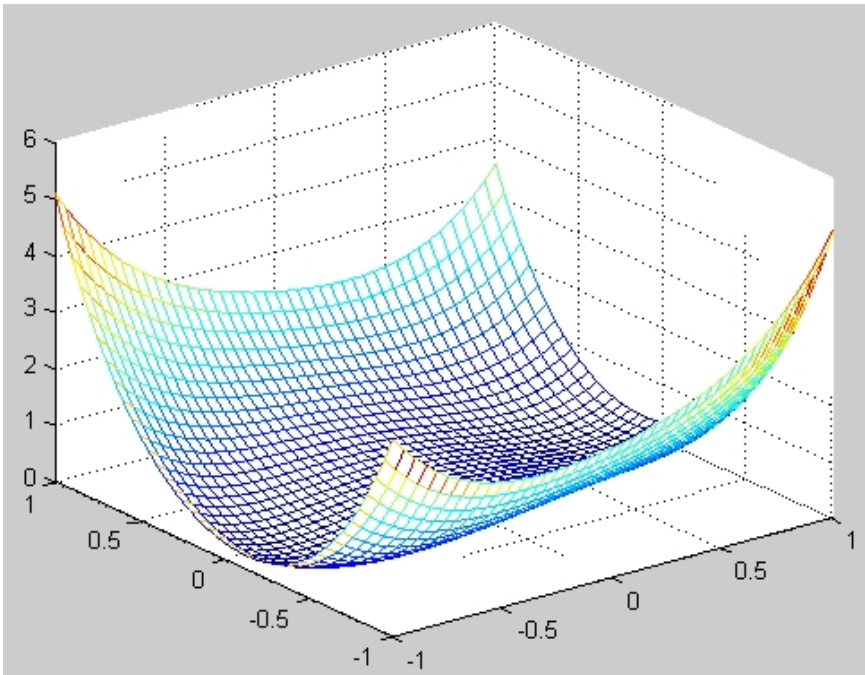
а Z – известное комплексное число, из которого извлекается квадратный корень.

Вместо модуля функции (1) можно анализировать квадрат ее модуля, который, очевидно, является функцией вида

$$\varphi(x, y) = \left| (x + jy)^2 - Z \right|^2$$

или

$$\varphi(x, y) = \left(x^2 - y^2 - \operatorname{Re} Z \right)^2 + (2xy - \operatorname{Im} Z)^2.$$



test3Dsqr.fig

На фиг. test3Dsqr.fig представлен вид функции (2). Программа test3Dsqr позволяет получить вид этой функции при любых

диапазонах изменения действительных чисел x , y и любых комплексных числах Z .

Очевидно, функция (2) имеет два локальных минимума, в которых эта функция принимает нулевое значение – искомый корень. Следовательно, спуск по этой функции в направлении минимума позволяет вычислить искомый корень. Важно при этом

- 1) начинать движение к оптимуму из окрестности главного локального минимума, который находится в правой полуплоскости,
- 2) следить за тем, чтобы любой шаг приводил к уменьшению значения функции (2).

Второе требование выполняется в предложенном алгоритме с очевидностью. Выполнение первого требования обеспечивается тем, что поиск начинается и заканчивается в первом полуквадранте. То, что он начинается в первом полуквадранте, определяется начальным значением итерационного процесса. То, что он заканчивается в первом полуквадранте, обеспечивается предварительным преобразованием исходного числа в *нормальную* (для извлечения корня) форму – см. выше.

Заметим, что алгоритм реализует, в сущности, покоординатный спуск с шагом ρ^k : этот шаг поочередно принимает значение в положительном и отрицательном направлении по каждой из координат.

1.5. Некоторые сравнения

Широко известны алгоритмы вычисления элементарных функций методом «цифра-за-цифрой». Рассмотрим некоторые программы, реализующие этот метод (что нужно будет для последующего сравнения). Эти программы находятся в каталоге CD/Trad/.

Функция **tOrt** вычисляет орт данного угла:

$$\text{ort} = (\cos \varphi + j \sin \varphi).$$

Можно заметить, что это вычисление выполняется за N итераций, где N – число разрядов в коде мантиссы, а каждая итерация выполняется за 5 тактов. Функция **tAtan** вычисляет при данных A и B одновременно аргумент и модуль комплексного числа $(B + jA)$:

$$\varphi = \arctg(A/B), \quad m = \sqrt{A^2 + B^2}.$$

Можно заметить, что это вычисление также выполняется за N итераций, где N – число разрядов в коде мантисс, а каждая итерация выполняется за 6 тактов.

А. Рассмотрим вычисление квадратного корня из комплексного числа на DSP-процессоре, в котором предусмотрены операции **tAtan** и **tOrt**. Применяя эти функции, можно вычислить корень квадратный из комплексного числа – см. функцию **tSqrt1**. Программа в этом случае имеет следующий вид: дано $z = a + jb$. Необходимо найти \sqrt{z} .

1. Определяем представление комплексного числа $z = a + jb$ в полярных координатах: $|z|, \varphi$ – см. функцию **tAtan**.
2. Вычисляем $\sqrt{|z|}$. Для этого в ДСП существует отдельная операция. Будем полагать, что корень квадратный из действительного числа также вычисляется за N итераций, а каждая итерация выполняется за 4 такта.
3. Вычисляем $\varphi/2$ и орт результата – см. функцию **tOrt**.

4. Определяем представление комплексного числа

$\sqrt{Z} = \sqrt{|Z|} \cdot e^{j\varphi/2}$ в прямоугольных координатах, что и требовалось.

Каждая из перечисленных в этом алгоритме операций, выполняется в процессоре ДСП одной инструкцией. Среди этих операций есть 3 сложных операций. При этом общее количество тактов при вычислении по функции `tSqrt1` равно

$$T_1 \approx 15N.$$

В. Рассмотрим вычисление квадратного корня из комплексного числа на процессоре INTEL. Программа, использующая трансцендентные функции, имеет следующий вид:

Дано $z = a + jb$. Необходимо найти

$$Z = A + jB = \sqrt{z} = (\cos\varphi + j\sin\varphi) \cdot \sqrt[4]{a^2 + b^2},$$

где

$$\varphi = \arctg(b/a)/2.$$

Программа при этом имеет следующий вид:

1. $c = (b/a)$
2. $\varphi = \arctg(c)$
3. $\varphi = \varphi/2$
4. $c = \cos\varphi$
5. $s = \sin\varphi$
6. $n = a^2$
7. $m = b^2$
8. $d = m + n$
9. $r = \sqrt{d}$
10. $r = \sqrt{r}$
11. $A = r \cdot c$
12. $B = r \cdot s$

Каждая из перечисленных в этом алгоритме операций, выполняется в процессоре INTEL одной инструкцией. Среди этих операций есть 5 сложных операций. При этом общее количество тактов при вычислении по функции равно

$$T_2 \approx 25N.$$

С. В предлагаемом устройстве вычисление состоит из единственной команды: $Z = \sqrt{z}$. Количество тактов при этом вычислении (как показано в разделе 5) равно

$$T \approx 6N.$$

|| Таким образом, время вычисления в на предлагаемом устройстве сокращается в **4.5** раза по сравнению с INTEL и в **2.5** раза по сравнению с ДСП (при той же технологии изготовления).

Глава 11. Моделирование устройства для потенцирования и логарифмирования комплексных чисел

2.1. Введение

В этой главе описывается устройство для потенцирования и логарифмирования комплексных чисел - см. также раздел 8 в части 1. На основе вышеописанного метода «цифра-за-цифрой» подробно рассматриваются алгоритмы этих операций и их моделирование в системе MATLAB. Программы для этого раздела находятся в каталоге CD/Expon. Показано, что по количеству тактов предлагаемый алгоритм сравним с алгоритмами вычисления элементарных функций действительного аргумента. Приведенной информации (вместе с некоторыми специальными схемами, описанными в части 3) достаточно для практической реализации устройства.

2.2. Композиции

Далее мы применим метод «цифра за цифрой» к вычислению экспоненты и логарифма комплексного числа - см. также разделы 8.1 и 8.2 в части 1. Рассмотрим разложения комплексного числа в следующем виде:

$$Z' = \prod (1 + \rho^{-h})^{a_h}, \quad (1)$$

$$Z'' = \sum a_h \cdot \ln(1 + \rho^{-h}), \quad (2)$$

где ρ – основание системы счисления 1 – см. раздел 1 и формулу (1.3), откуда следует, что

$$\rho^h = \left\{ \begin{array}{ll} (-2)^{h/2} & \text{if } h - \text{even,} \\ j \cdot (-2)^{(h-1)/2} & \text{if } h - \text{odd.} \end{array} \right\}. \quad (3)$$

Если числа α^h в разложениях (1) и (2) совпадают, то

$$Z'' = \ln Z', \quad (4)$$

и

$$Z' = e^{Z''}. \quad (5)$$

В дальнейшем будем применять разложения (1, 2) соответственно в следующем виде:

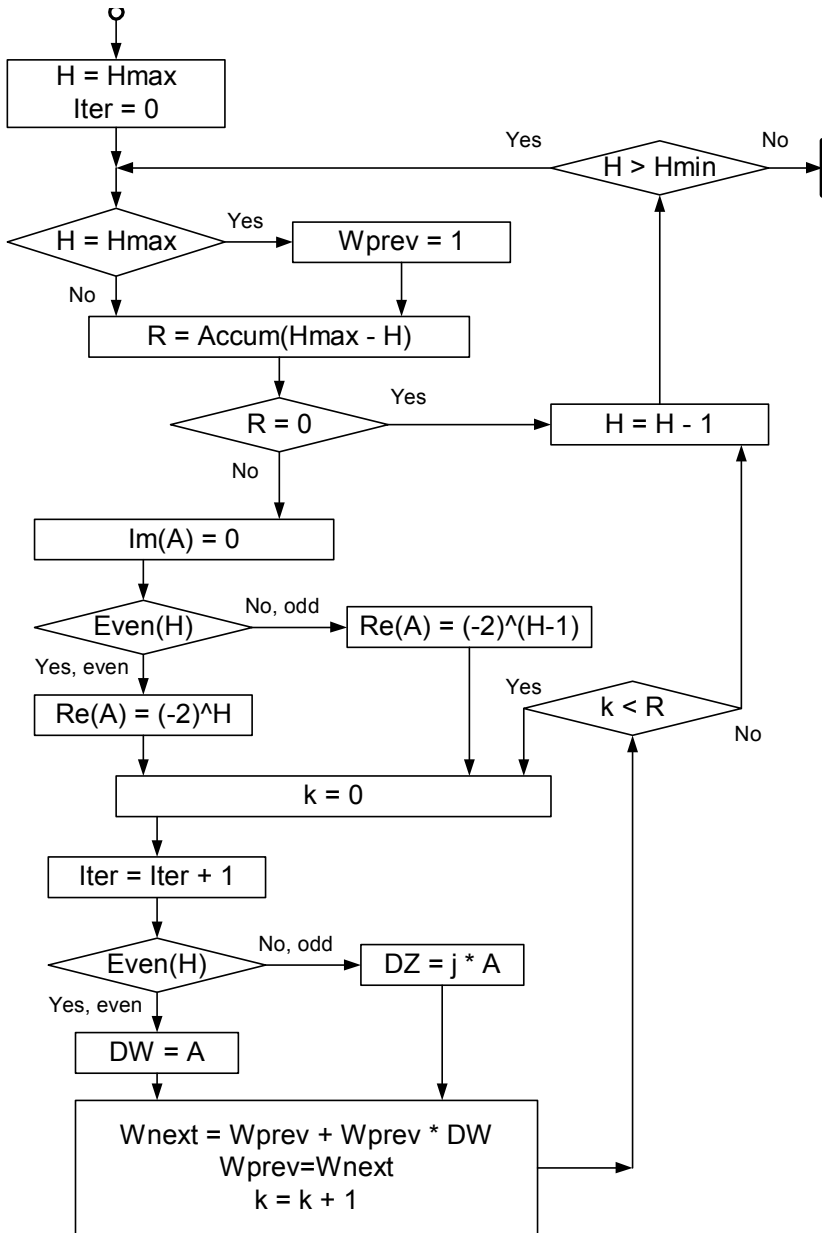
$$Z' = \prod_{H=H \min}^{H \max} (1 + \rho^{-H})^{a_H}, \quad (6)$$

$$Z'' = \sum_{H=H \min}^{H \max} a_H \cdot \ln(1 + \rho^{-H}). \quad (7)$$

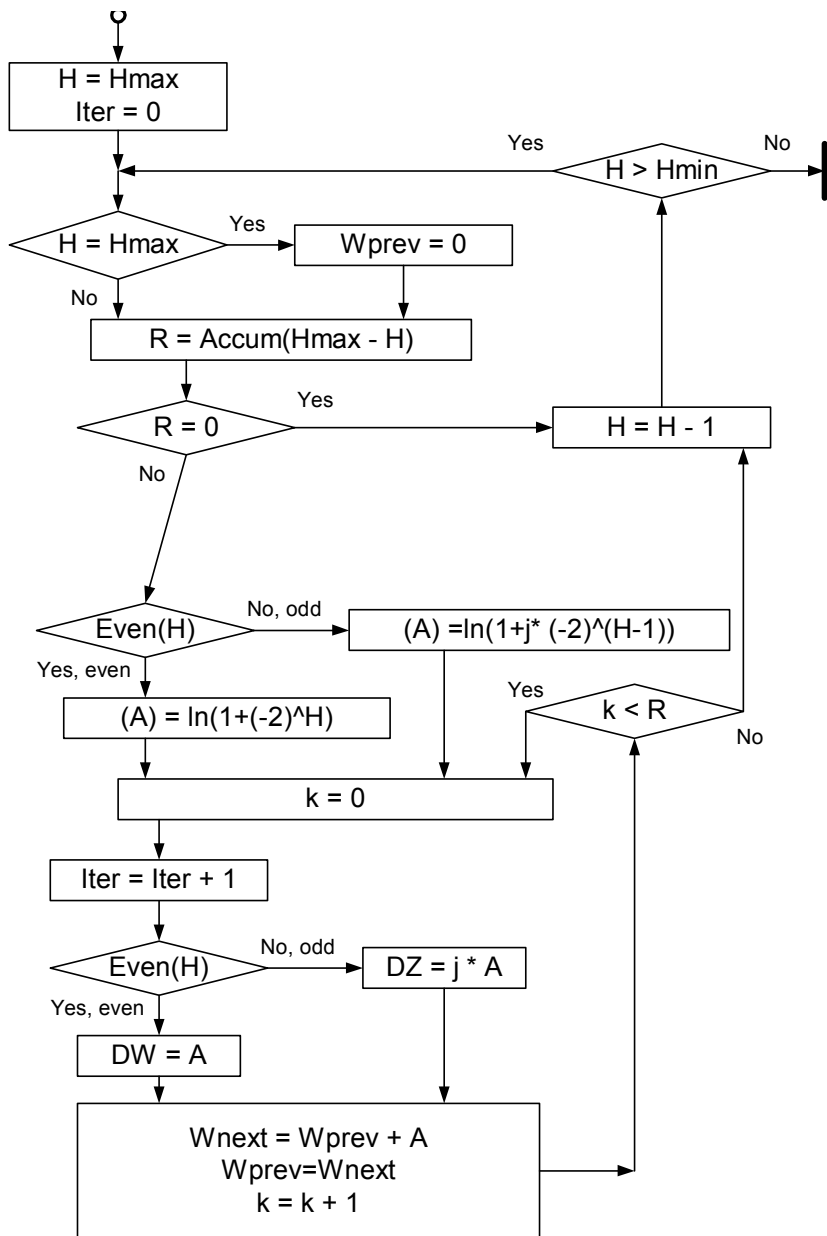
Результат разложения представляет собой множество чисел

$$Accum = \left\{ \alpha^{H \min}, \dots, \alpha^H, \dots, \alpha^{H \max} \right\}. \quad (8)$$

По известному множеству (8) может быть выполнена композиция числа Z .



Фиг. 1.



Фиг. 2.

На фиг. 1 и фиг. 2 представлены блок-схемы алгоритмов композиции по формулам (1) и (2) соответственно, где приняты следующие обозначения:

W_{prev} — предыдущее значение результата,

W_{next} — следующее значение результата,

DW – приращение,

k – счетчик значения разряда α^H массива Accum,

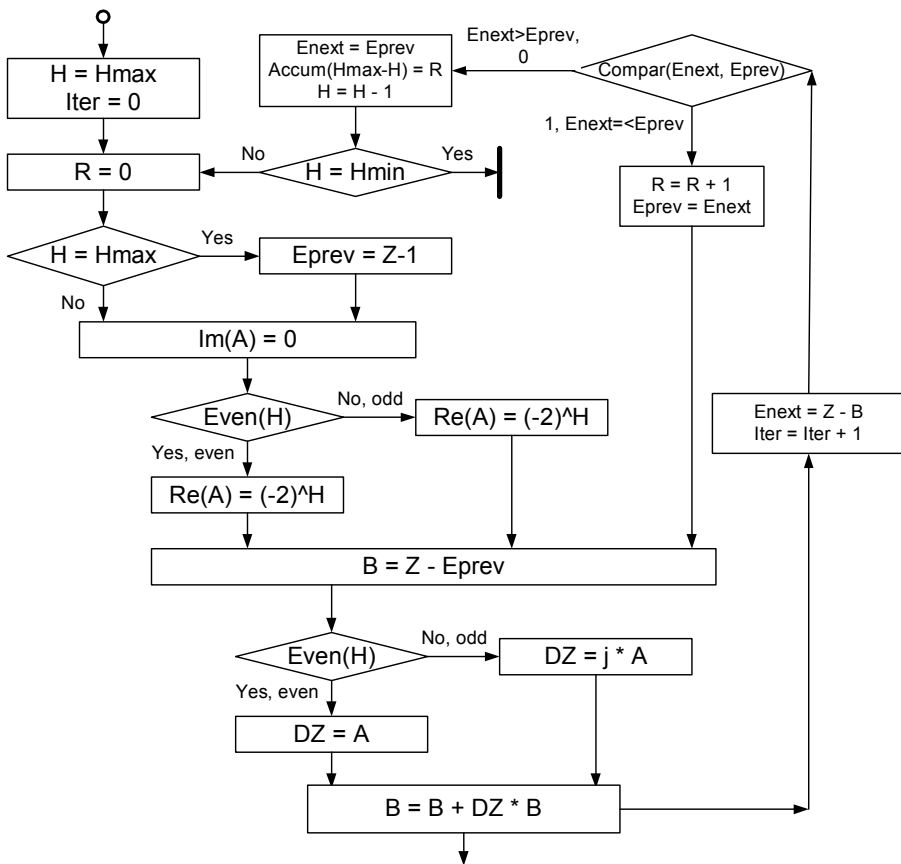
R – текущее значение разряда α^H ,

H – текущий номер разряда, $H_{\min} \leq H \leq H_{\max}$

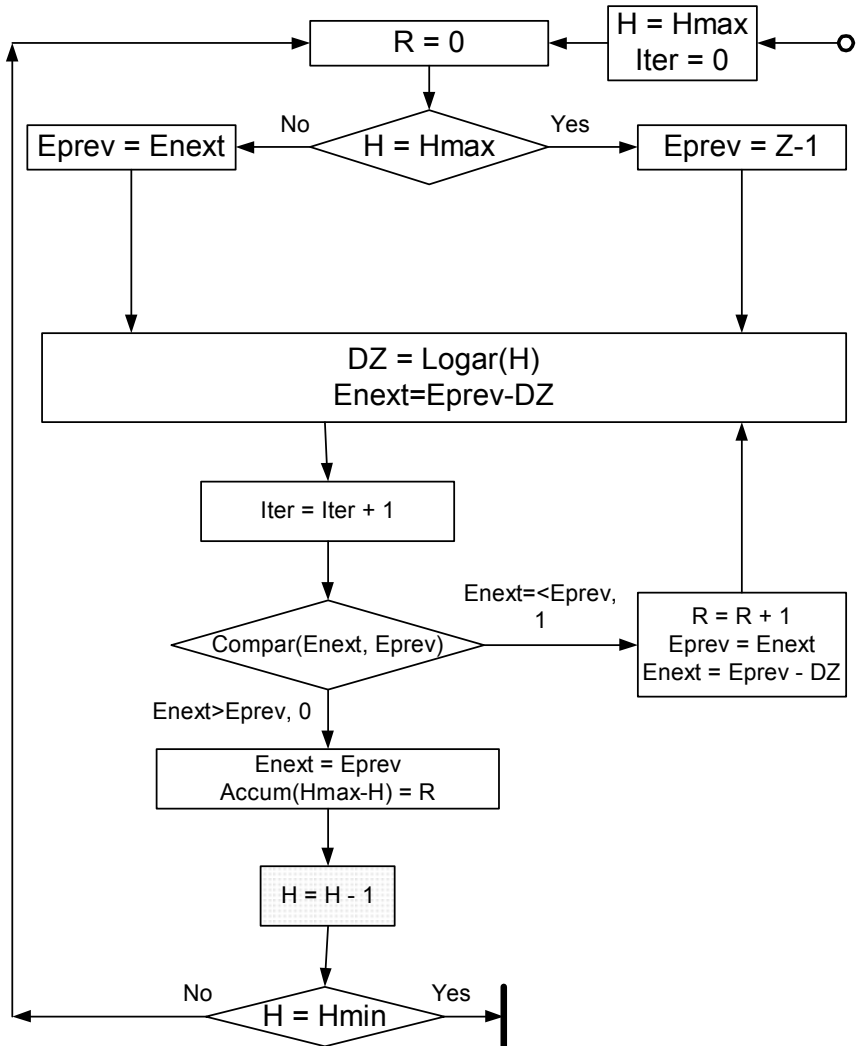
Even – функция проверки четности действительного целого положительного числа H,

Iter – счетчик итераций,

A – промежуточный результат (комплексное число).



Фиг. 3.



Фиг. 4.

2.2.3. Декомпозиции

Декомпозиция находит коэффициенты (8) разложений (1) или (2). Блок-схемы алгоритмов декомпозиции для разложений (1) или (2) представлены на фиг. 3 и фиг. 4 соответственно, где приняты следующие обозначения (в дополнение к обозначениям для алгоритма композиции):

E_{prev} — предыдущее значение остатка,

E_{next} — следующее значение остатка,

Compar – функция сравнения комплексных чисел по модулю, которая возвращает значение «1», если $E_{next} < E_{prev}$ и «0», если $E_{next} > E_{prev}$,

DZ – приращение,

Logar – функция выбора константы $\ln(1 + \rho^{-h})$,

A, B – промежуточные результаты (комплексные числа).

Функция DecBin1 реализует алгоритм декомпозиции комплексных чисел в разложение (1) и композиции по (2). В ней используются

- функция CNcompar, которая вычисляет модуль комплексного числа для последующего сравнения модулей остатков E_{next} и E_{prev} ,
- функция MultBinom0, которая умножает комплексное число на бином $(1 + \rho^{-h})$,
- функция LogBinom0, которая складывает комплексное число с константой $\ln(1 + \rho^{-h})$.

Именно простота аппаратной реализации этих функций предопределяет применение указанных разложений для комплексных чисел.

Используются принятые выше обозначения и следующие:

logZ – число, вычисляемое по (2) параллельно с декомпозицией,

Iter – номер итерации,

IterPlus – номер удачной итерации,

DM – разность модулей остатков E_{next} и E_{prev} после очередной итерации Iter,

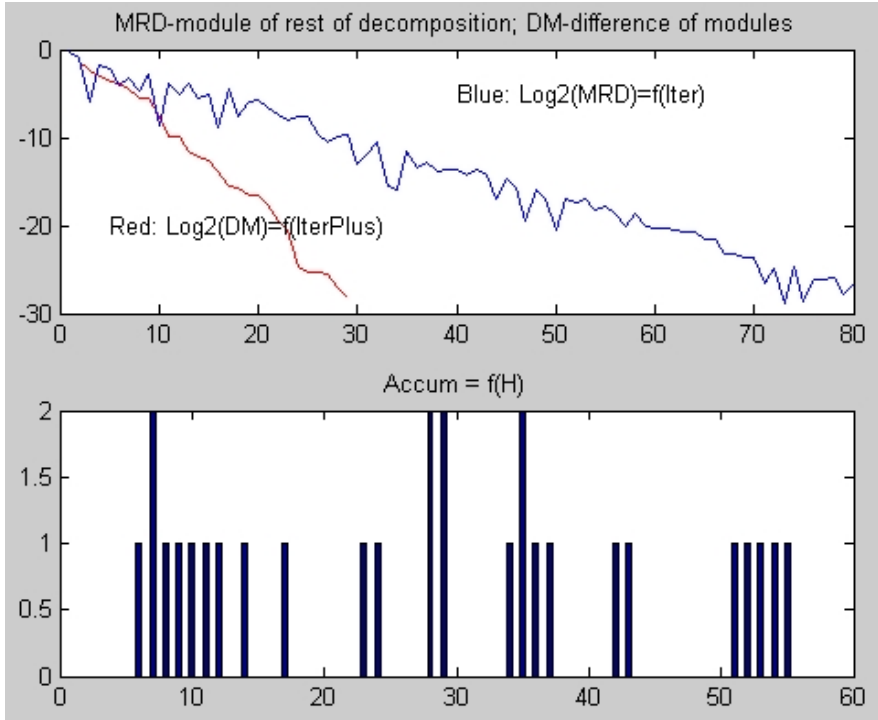
MRD – модуль следующего значения остатка E_{next} после очередной удачной итерации IterPlus.

Функция DecLog1 реализует алгоритм декомпозиции комплексных чисел в разложение (2) и композиции по (1). В ней используются принятые выше обозначения и следующие:

expZ – число, вычисляемое по (1) параллельно с декомпозицией.

Функция `testDecBin1` позволяет при обращении к функции `DecBin1` анализировать алгоритм декомпозиции. В этой функции

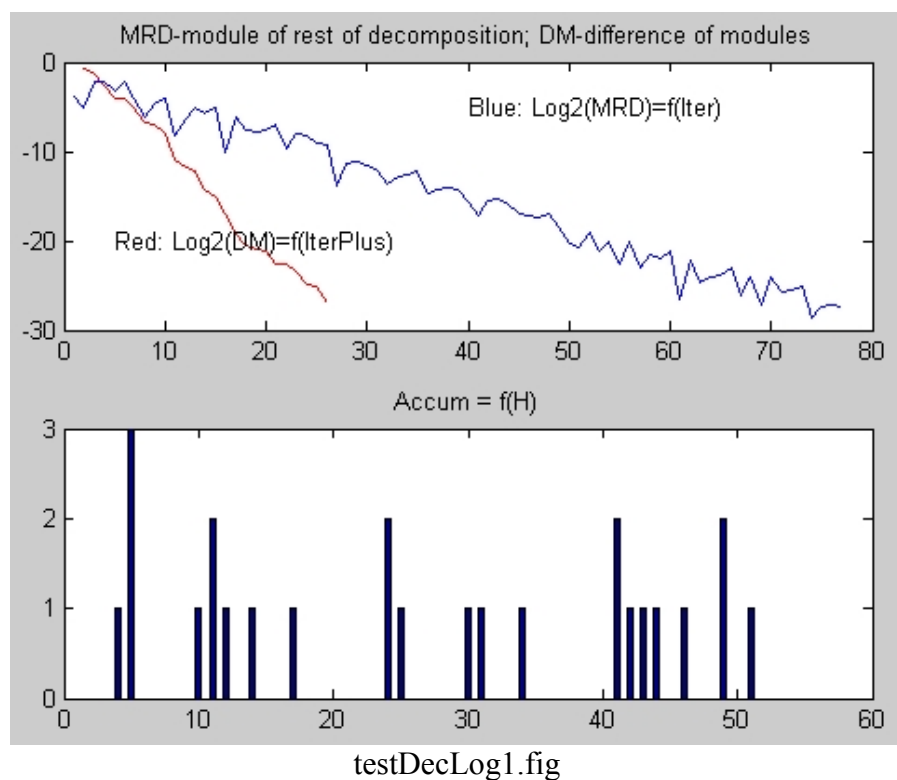
- вычисляется логарифмическая ошибка извлечения логарифмирования, т.е. количество верных разрядов `er`,
- строятся зависимости `DM(Iter)` и `MRD(IterPlus)` в первом окне - см. фиг. `testDecBin1.fig`,
- строится гистограмма значений разрядов массива (8) во втором окне - см. фиг. `testDecBin1.fig`.



testDecBin1.fig

Функция `testDecLog1` позволяет при обращении к функции `DecLog1` анализировать алгоритм декомпозиции. В этой функции

- вычисляется логарифмическая ошибка извлечения логарифмирования, т.е. количество верных разрядов `er`,
- строятся зависимости `DM(Iter)` и `MRD(IterPlus)` в первом окне - см. фиг. `testDecLog1.fig`,
- строится гистограмма значений разрядов массива (8) во втором окне - см. фиг. `testDecLog1.fig`.



2.2.4. Область представимых чисел

На фиг. 5 изображены числа – биномы вида $b_h = (1 + \rho^{-h})$, где ρ определено по (4.3). Пусть $h > 0$ и $h = 4k + m$, где $m = \{0, 1, 2, 3\}$.

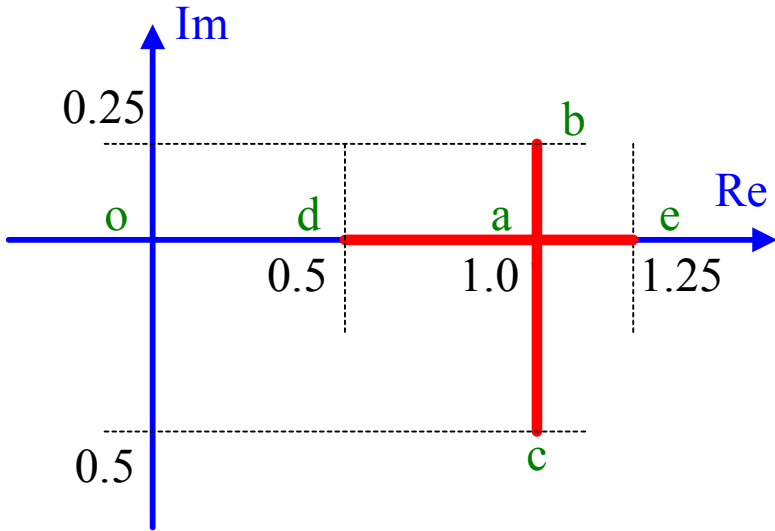
На фиг. 1

ae - область изменения биномов при $m=0$,

ас - область изменения биномов при $m=1$,

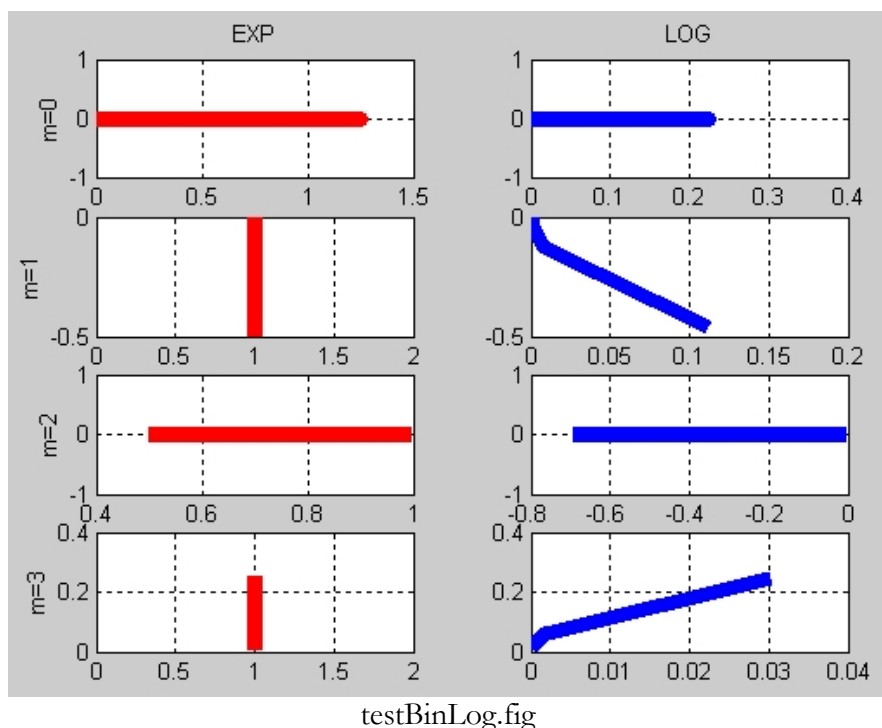
ad - область изменения биномов при $m=2$,

ab - область изменения биномов при $m=3$.



Фиг. 5

Функция `testBinLog` строит области изменения биномов $b_h = (1 + \rho^{-h})$ и логарифмов биномов $\log b_h = \ln(1 + \rho^{-h})$. Эти области изменения этих логарифмов представлены на фиг. `testBinLog.fig`. В левых окнах указаны области изменения биномов, а в правых - области изменения логарифмов биномов. Ряды окон пронумерованы значениями $m = \{0, 1, 2, 3\}$, которые определяются по условию $h = 4k + m$.



Отсюда следует, что любое комплексное число с реальной НЕотрицательной частью при $H_{min} > 0$ может быть представлено произведением биномов (1), если кратность α^H вхождения биномов в это произведение НЕ ограничена. Также любое комплексное число с реальной НЕотрицательной частью при $H_{min} > 0$ может быть представлено суммой логарифмов (2), если кратность α^H вхождения логарифмов в эту сумму НЕ ограничена. Вопрос заключается в том, насколько велика эта кратность, ибо для аппаратной реализации кратность должна быть мала. Для анализа кратности биномов можно воспользоваться функцией `testDecBinAn`.

На следующих фигурах представлены некоторые области чисел для декомпозиции по формуле (1). Для каждого из чисел определено максимальное значение кратности `maxAccum` вхождения квадратов биномов в разложение (1). Цвет точки указывается в зависимости от значения `maxAccum` так:

- синий, если `maxAccum < 2`,

- красный, если $\max \text{Accum}=2$,
- зеленый, если $\max \text{Accum}=3$,
- черный, если $\max \text{Accum}=4$,
- голубой, если $\max \text{Accum}=5$,
- розовый, если $\max \text{Accum}>5$.

Принимаются $N_{\min}=3$, $N_{\max}=55$. и вычисляются максимальная кратность для всей области $\max(\max \text{Accum})$ и точность декомпозиции er .

На фиг. f91fu в правом окне представлена область чисел вида

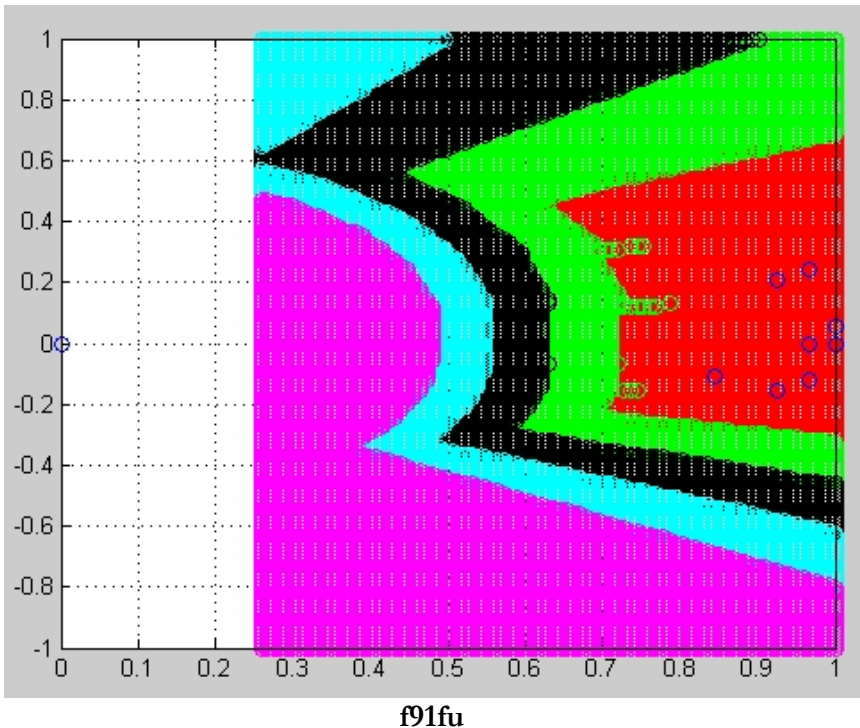
$$0.25 \leq x \leq 1, \quad |y| \leq 1.$$

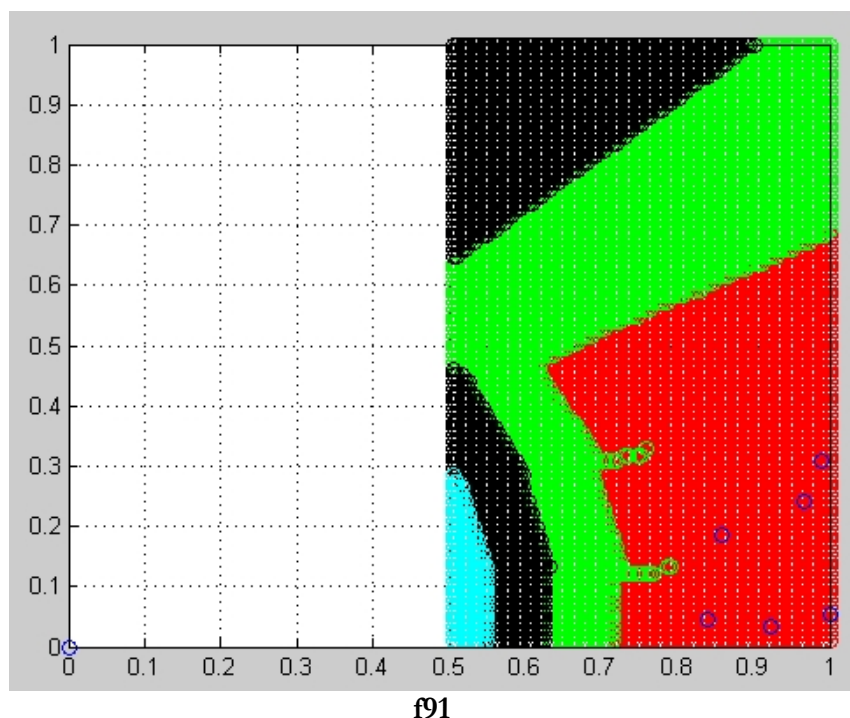
В результате получены $\max(\max \text{Accum})=11$ и точность декомпозиции $\text{er}=2.2\text{e-}8$.

На фиг. f91 в левом окне представлена область чисел вида

$$0.5 \leq x \leq 1, \quad 0 \leq y \leq 1. \quad (17)$$

В результате получены $\max(\max \text{Accum})=5$ и точность декомпозиции $\text{er}=2.1\text{e-}8$.





Для анализа кратности вхождения логарифмов биномов можно воспользоваться функцией `testDecLogAn`. В ней используются предыдущие обозначения.

На фиг. f101fulog в правом окне представлена область чисел вида

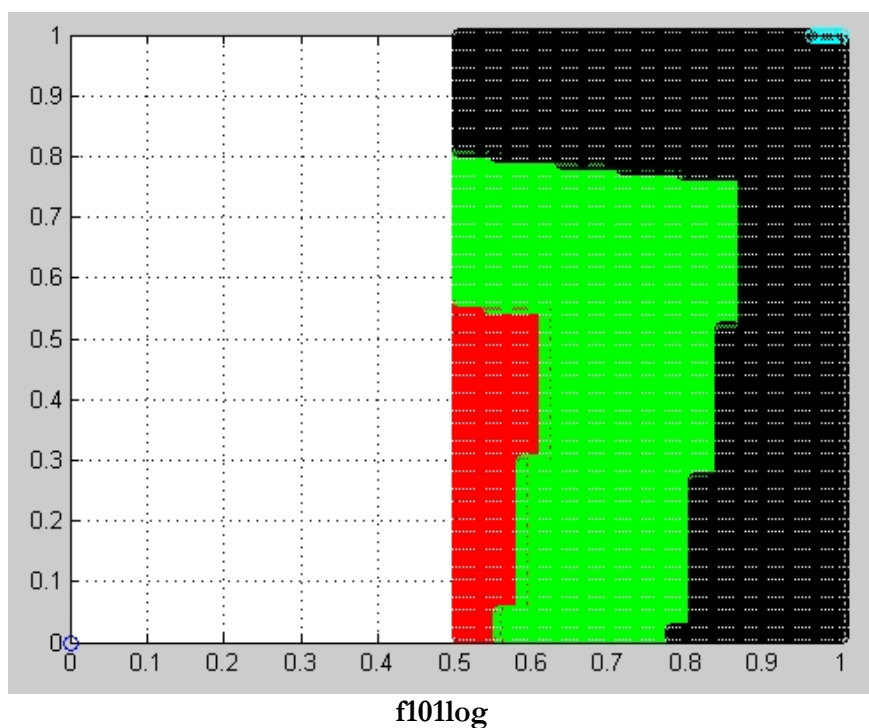
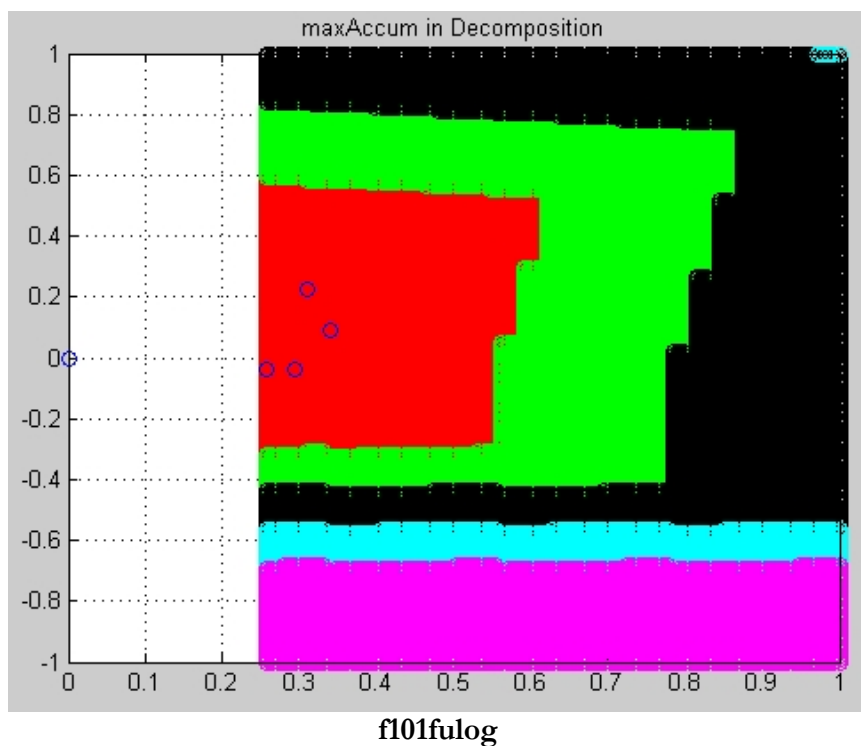
$$0.25 \leq x \leq 1, \quad |y| \leq 1.$$

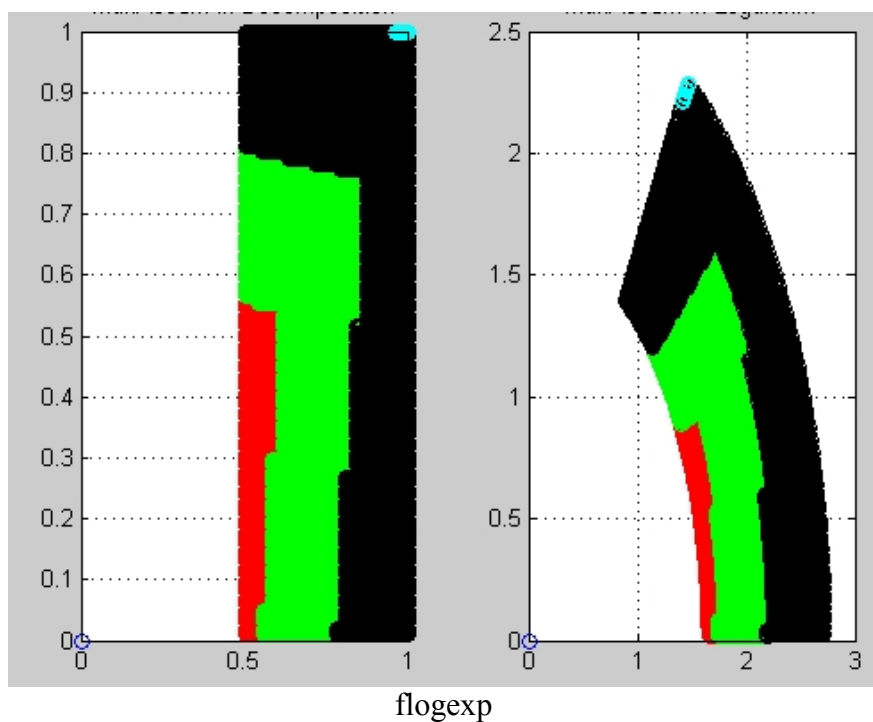
В результате получены $\max(\maxAccum)=8$ и точность декомпозиции $er=1.7e-8$.

На фиг. f101log в левом окне представлена область чисел вида

$$0.5 \leq x \leq 1, \quad 0 \leq y \leq 1. \quad (18)$$

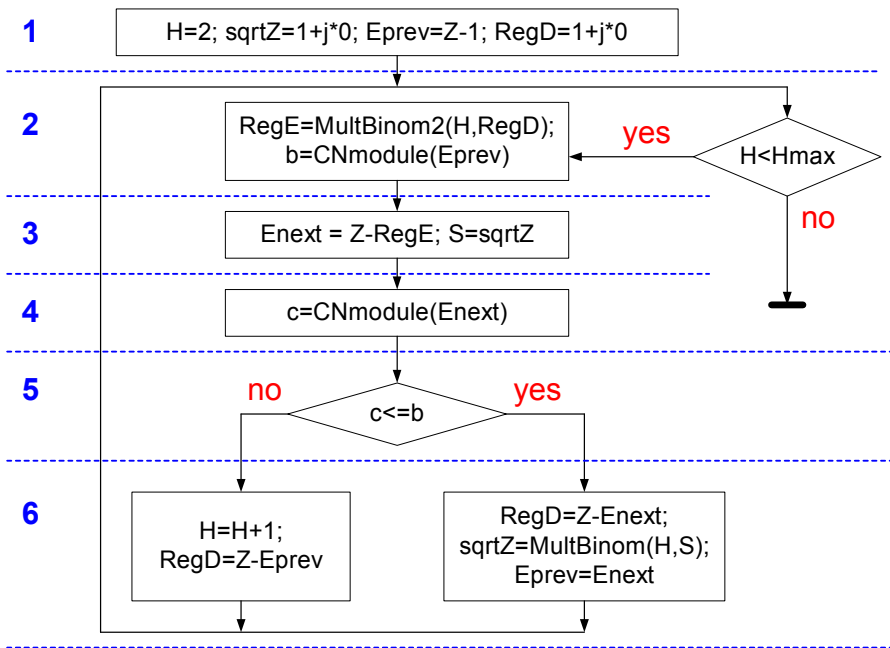
В результате получены $\max(\maxAccum)=5$ и точность декомпозиции $er=1.7e-8$.





2.2.5. Оценка быстродействия.

Рассмотрим функцию `dcb`, которая отличается от функции `DecBin2` отсутствием всех операторов и переменных, необходимых для отображения и анализа результатов. Функция `testDCB` тестирует функцию `dcb`.



Фиг. 3.

На фиг. 3 функция `dcb` изображена в виде блок-схемы, где операторы, которые при аппаратной реализации могут выполняться параллельно, представлены в одном блоке. Цифрами на этой схеме обозначены номера тактов. По этой схеме можно оценить время выполнения декомпозиции в тактах. Каждая итерация выполняется за 5 тактов. Количество итераций

$$Iter \approx \frac{3}{2} H_{\max}$$

и

$$N = H_{\max}/2,$$

где N – число разрядов в коде мантиссы каждой части комплексного числа. Таким образом, количество тактов декомпозиции

$$T \approx 15N.$$

Например, для кода с 23-разрядной мантиссой $T \approx 350$, для кода с 16-разрядной мантиссой $T \approx 240$.

Отметим еще, что этот же алгоритм может быть применен для вычисления корня квадратного из действительного положительного числа. Декомпозиция такого числа содержит только биномы с действительными ρ^H - см. формулу (2). Поэтому при декомпозиции достаточно анализировать только элементы с четными значениями H и время декомпозиции сокращается вдвое.

Итак, *время вычисления корня квадратного из комплексного числа вдвое больше времени вычисления корня квадратного из действительного положительного числа.*

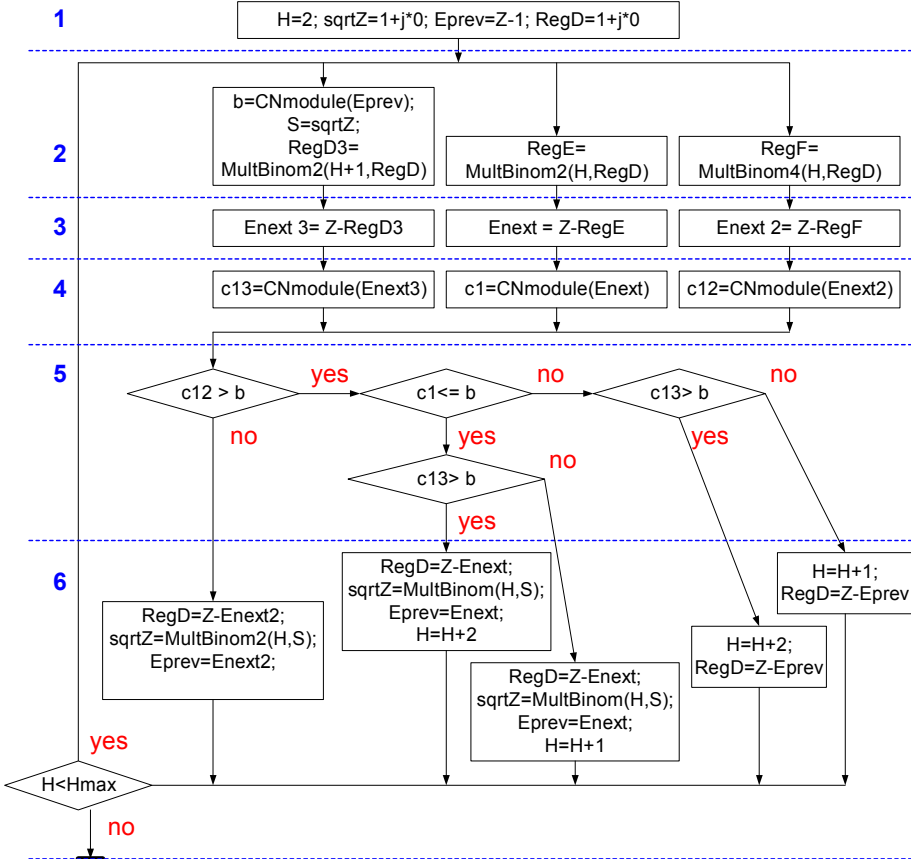
2.2.6. Оптимизация по быстродействию

Рассмотрим теперь возможные способы ускорения вычисления. В основном алгоритме делается пробное умножение на квадрат бинома при данном H . *Первый способ* ускорения состоит в том, что выполняется пробное умножение в двух вариантах: на квадрат бинома (что эквивалентно проверке условия $\alpha^H = 1$) и на четвертую степень бинома (что эквивалентно проверке условия $\alpha^H = 2$). *Второй способ* ускорения состоит в том, что выполняется пробное умножение на квадрат следующего бинома (что эквивалентно проверке условия $\alpha^{H+1} = 1$). Анализ всех этих вариантов позволяет *в более чем в два раза уменьшить количество итераций.*

Оба способа реализованы в функции `DecompBinom3`. В рассмотренных выше тестах она может заменить функцию `DecBin2`.

Рассмотрим функцию `dcb3`, которая отличается от функции `DecompBinom3` отсутствием всех операторов и переменных, необходимых для отображения и анализа результатов. Функция `testDCB3` тестирует функцию `dcb3`.

На фиг. 4 функция `dcb3` изображена в виде блок-схемы, где операторы, которые при аппаратной реализации могут выполняться параллельно, представлены в одном блоке или в параллельно включенных блоках.



Фиг. 4.

Каждая итерация выполняется за 5 тактов. Количество итераций

$$Iter \approx 0.7 \cdot H_{\max}.$$

Таким образом, количество тактов декомпозиции

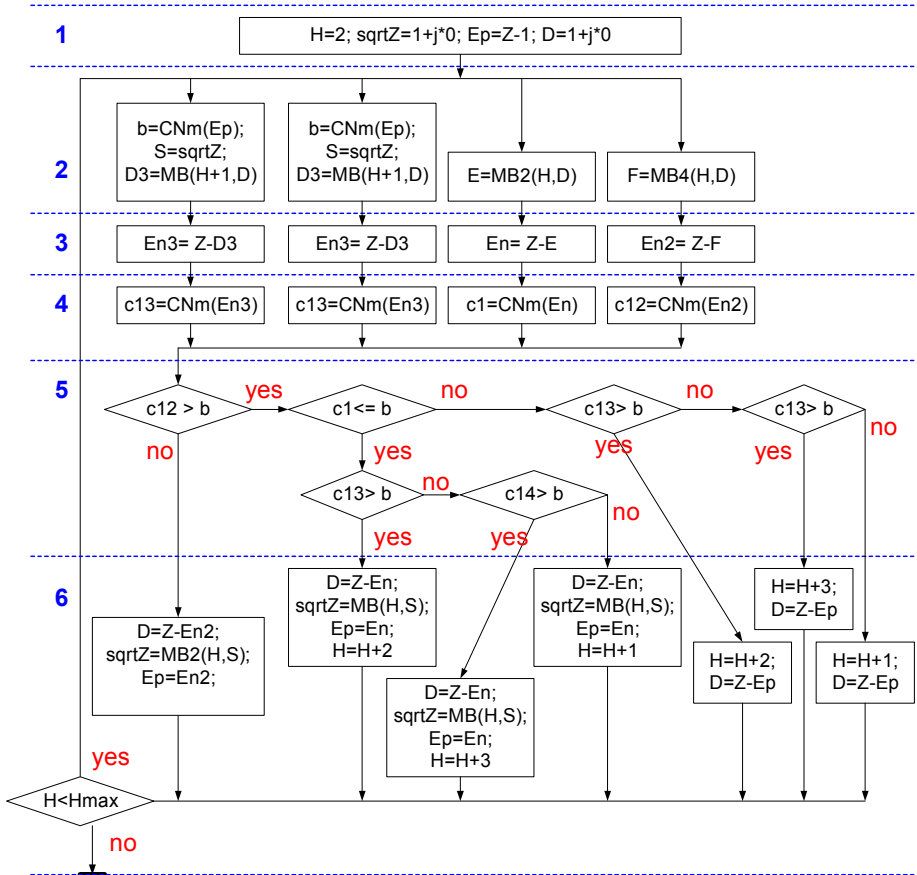
$$T \approx 7N.$$

Например, для кода с 23-разрядной мантиссой $T \approx 160$, для кода с 16-разрядной мантиссой $T \approx 110$.

Дальнейшего ускорения можно достигнуть, выполняя пробное умножение на квадрат бинома, эквивалентного проверке условия $\alpha^{H+2} = 1$ и т.д. Практический эффект от такого прогноза быстро падает с увеличением глубины прогноза. Так, при $H_{\max} = 55$ и

- при отсутствии всех способов ускорения математическое ожидание числа итераций $\bar{it} = 80$ - см. функцию DecBin2,
- при отсутствии прогноза $\bar{it} = 59$,

- при ускорении по способу 1 и прогнозе одного бинома $\bar{it} = 40$ - см. функцию **DecompBinom3**,
- при ускорении по способу 1 и прогнозе двух биномов $\bar{it} = 33$ - см. функцию **DecompBinom4**,
- при ускорении по способу 1 и прогнозе трех биномов $\bar{it} = 32$.



Фиг. 5.

Таким образом, целесообразно остановиться на прогнозе двух биномов. Поэтому рассмотрим функцию **dcb4**, которая отличается от функции **DecompBinom4** отсутствием всех операторов и переменных, необходимых для отображения и анализа результатов. Функция **testDCB4** тестирует функцию **dcb4**.

На фиг. 5 функция **dcb4** изображена в виде блок-схемы. В отличие от фиг. 4 на фиг. 5 переменные обозначены очевидными сокращенными именами.

В функции **dcb4** каждая итерация также выполняется за 5 тактов. Количество итераций

$$Iter \approx 0.6 \cdot H_{\max}.$$

Таким образом, количество тактов декомпозиции

$$T \approx 6N.$$

Например, для кода с 23-разрядной мантиссой $T \approx 140$, для кода с 16-разрядной мантиссой $T \approx 90$.

2.2.7. Дополнительные функции устройства

На данном устройстве могут быть вычислены и другие функции комплексного числа. Рассмотрим их.

2.2.7.1. Вычисление орта комплексного числа

Если комплексное число $Z = |Z|e^{j\varphi}$, то орт этого комплексного числа

$$e^{j\varphi} = \sqrt{\frac{Z^2}{|Z|^2}}. \quad (1)$$

Таким образом, если имеется разложение вида (4.3в)

$$Z^2 = |Z|^2 \cdot \prod (1 + \rho^{-h})^{2 \cdot ah}, \quad (2)$$

то орт может быть найден как разложение вида (3с)

$$e^{j\varphi} = \sqrt{\frac{Z^2}{|Z|^2}} = \prod (1 + \rho^{-h})^{ah}, \quad (3)$$

Для применения этих формул при данном $Z = a + jb$ необходимо вычислить $Z^2 = a^2 + 2jab - b^2$, $|Z|^2 = a^2 + b^2$ (для чего нужно 3 умножения и 3 сложения), а затем выполнить одновременно декомпозицию (2) и композицию (3).

2.2.7.2. Вычисление логарифма модуля и аргумента комплексного числа

Если комплексное число $Z = |Z|e^{j\varphi}$, то

$$\ln(Z) = \ln|Z| + j\varphi. \quad (4)$$

Таким образом, если имеется разложение вида (4.1)

$$Z = \prod (1 + \rho^{-h})^{2 \cdot a_h}, \quad (5)$$

то

$$(\ln|Z| + j\varphi) = 2 \cdot \sum a_h \cdot \ln(1 + \rho^{-h}). \quad (6)$$

В последнюю формулу входят константы вида

$$L_h = \ln(1 + \rho^{-h}), \quad (7)$$

которые могут быть вычислены заранее.

Для применения этих формул необходимо выполнить одновременно декомпозицию (5) и композицию (6) с известными константами (7). Реальная и мнимая части результата представляют $\ln|Z|$ и $j\varphi$ соответственно. Очевидно, можно использовать константы $\text{Re}(L_h)$ или $\text{Im}(L_h)$ и вычислять либо $\ln|Z|$, либо $j\varphi$.

2.2.7.3. Вычисление корня квадратного и логарифма комплексного числа

Вычисление корня квадратного и логарифма комплексного числа могут быть совмещены. Действительно, можно выполнить одновременно декомпозицию (5), композицию (6) и композицию

$$\sqrt{Z} = \prod (1 + \rho^{-h})^{a_h}, \quad (8)$$

Глава 12. Моделирование устройства для вычисления одного корня степенного полинома

3.1. Композиции

Далее мы применим метод «цифра за цифрой» для вычисления корня степенного полинома - см. также раздел 9.2 в части 1. Программы для этого раздела находятся в каталоге CD/Polynom/

Везде далее степень полинома будем обозначать через n . Рассмотрим разложение комплексного числа в следующем виде:

$$Z = \prod (1 + \rho^{-h})^{n \cdot a_h}, \quad (1)$$

где ρ – основание системы счисления 1 – см. раздел 1 и формулу (1.3), откуда следует, что

$$\rho^h = \begin{cases} (-2)^{h/2} & \text{if } h - \text{even}, \\ j \cdot (-2)^{(h-1)/2} & \text{if } h - \text{odd}. \end{cases} \quad (2)$$

Очевидно,

$$\sqrt[n]{Z} = \prod (1 + \rho^{-h})^{a_h}, \quad (3)$$

В дальнейшем будем применять разложение (1) в виде

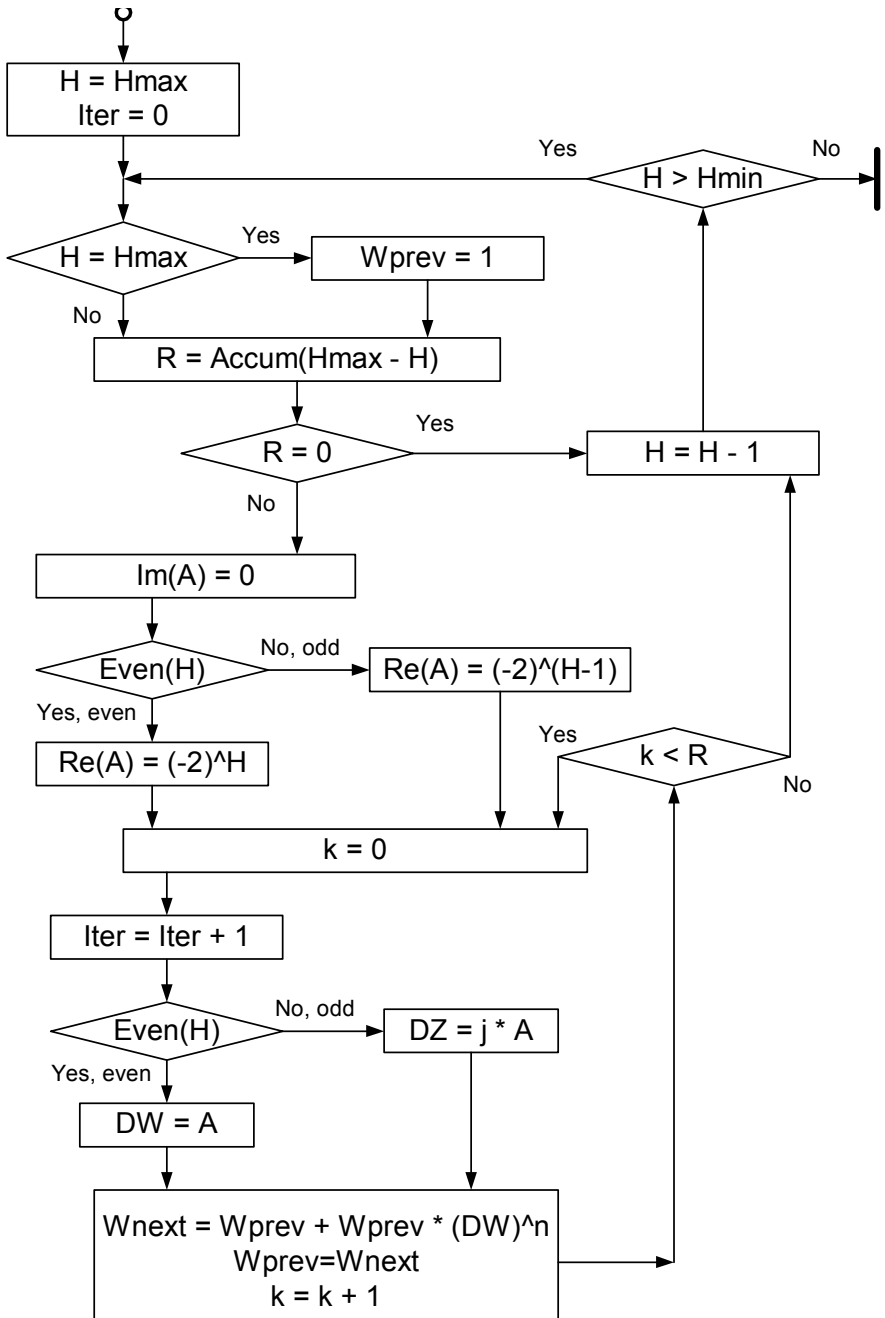
$$Z = \prod_{H=H \min}^{H \max} (1 + \rho^{-H})^{n \cdot a_H}, \quad (4)$$

и разложение (3) в виде

$$Z = \prod_{H=H \min}^{H \max} (1 + \rho^{-H})^{a_H}, \quad (5)$$

Рассмотрим область комплексных чисел, которые могут быть представлены разложением (4). Для этого рассмотрим алгоритм композиции числа X при известном множестве – массиве чисел

$$Accum = \{ \alpha^{H \min}, \dots, \alpha^H, \dots, \alpha^{H \max} \}. \quad (6)$$



Фиг. 1.

Блок-схема этого алгоритма представлена на фиг. 1, где приняты следующие обозначения:

Wprev– предыдущее значение результата,

Wnext– следующее значение результата,

DW– приращение,

k – счетчик значения разряда α^H массива Assum,

R – текущее значение разряда α^H ,

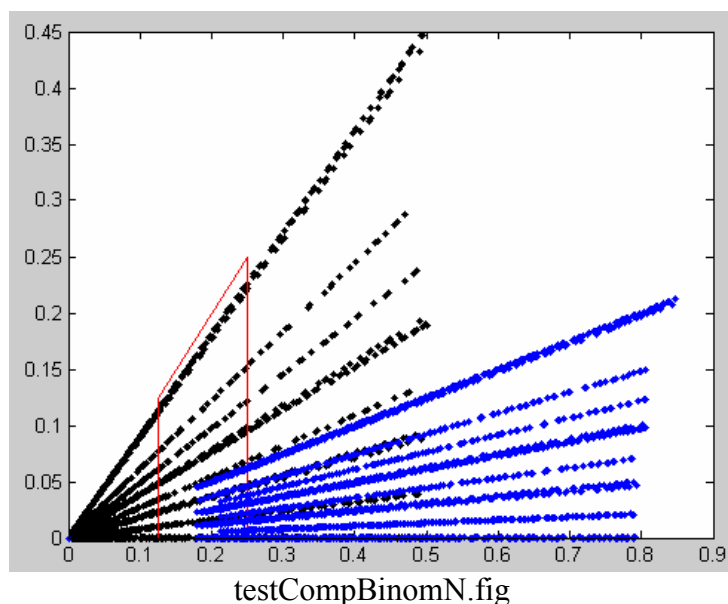
H – текущий номер разряда, Hmin =< H =< Hmax

Even– функция проверки четности действительного целого положительного числа H,

Iter– счетчик итераций,

A – промежуточный результат (комплексное число).

Этот же алгоритм реализован в функции CompBinomN. В ней используется функции MultBinomN и MultBinom. Последняя умножает комплексное число на бином $(1 + \rho^{-h})$. Именно простота аппаратной реализации этой функции предопределяет применение указанных разложений для комплексных чисел.



Функция testCompBinomN позволяет при обращении к функции CompBinomN построить область чисел, представимых разложением (4) при $n=power$, $Hmin=2$ и $\alpha^H = \{0, 1, 2, 3\}$. На

фиг. testCompBinomN.fig можно видеть эту область при $power=3$ (черные точки). При этом можно убедиться, что

- плотность расположения точек в области зависит от H_{max} ,
- эта область охватывает первый полуквadrant.

Функция **CompBinom1** реализует алгоритм композиции чисел, представленных разложением (5). Функция **testCompBinomN** позволяет при обращении к функции **CompBinom1** построить область чисел, представимых разложением (5). На фиг. testCompBinomN.fig можно видеть также и эту область (синие точки). Таким образом, на фиг. testCompBinomN.fig представлены области чисел Z (разложение (4)) и $\sqrt[n]{Z}$ (разложение (5)).

3.3.2. Декомпозиции для решения нормального квадратного трехчлена

Квадратный трехчлен вида

$$a \cdot x^2 + b \cdot x = c \quad (1)$$

с комплексными коэффициентам a, b, c будем называть нормальным, если

- 1) $c \neq 0, a \neq 0$,
- 2) известно, что по крайней мере один из корней $x = z$ таков, что комплексное число z^2 находится в первом полуквadrante.

Рассмотрим решение такого полинома методом «цифра-за-цифрой». Блок-схема алгоритма декомпозиции представлена на фиг. 2, где приняты следующие обозначения (в дополнение к обозначениям для алгоритма композиции):

Eprev— предыдущее значение остатка,

Enext— следующее значение остатка,

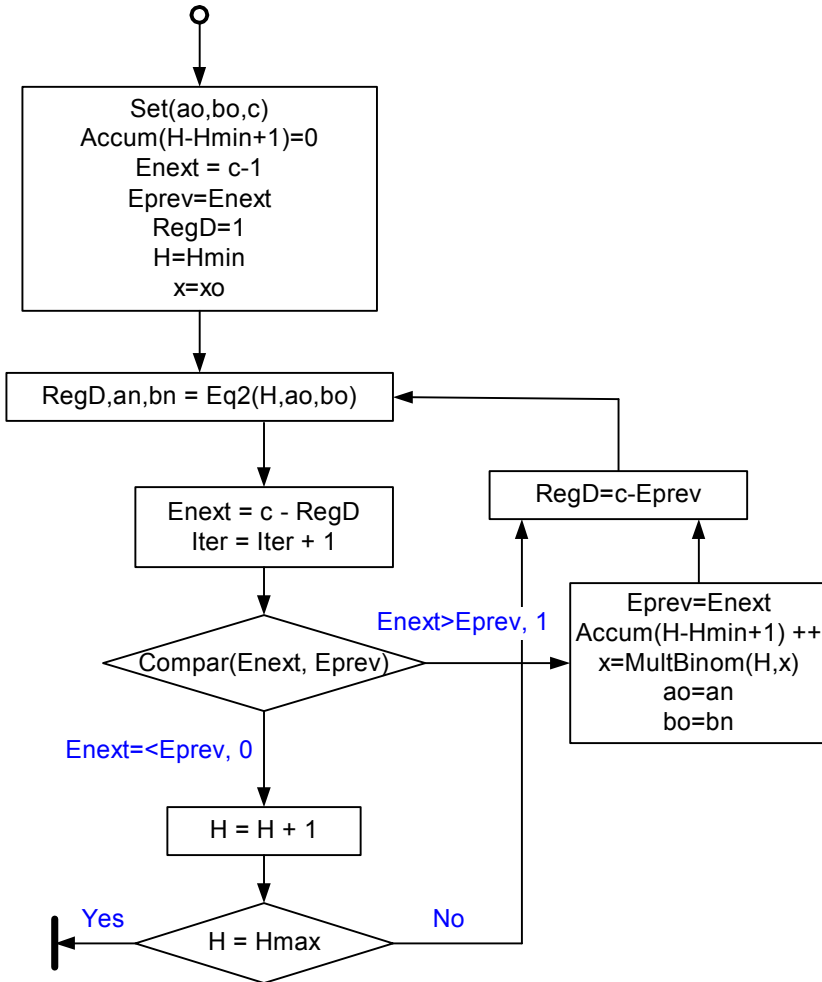
Compar— функция сравнения комплексных чисел по модулю, которая возвращает значение «1», если $Enext < Eprev$ и «0», если $Enext > Eprev$,

ao, an — текущее значение величины $a \cdot x^2$,

bo, bn — текущее значение величины $b \cdot x$,

RegD — текущее значение величины $D = a \cdot x^2 + b \cdot x$

$\text{MultBinom}(H, x)$ – функция умножения на бином (описанная выше),
 x_0 – начальное значение корня x ,
 $\text{Set}(a_0, b_0)$ – функция установки коэффициентов трехчлена,
 $[\text{RegD}, a_n, b_n] = \text{Eq2}(H, a_0, b_0)$ – функция вычисления текущих значений.



Фиг. 2.

Функция **DecompBinE2** реализует этот алгоритм для построения разложения (4). В ней используются также функция **CNcompar**, которая вычисляет модуль комплексного числа для последующего

сравнения модулей остатков E_{next} и E_{prev} . Используются принятые выше обозначения и следующие:

$Iter$ – номер итерации,

$IterPlus$ - номер удачной итерации,

DM – разность модулей остатков E_{next} и E_{prev} после очередной итерации $Iter$,

MRD - модуль следующего значения остатка E_{next} после очередной удачной итерации $IterPlus$.

Функция `testDecompBinE2` позволяет при обращении к функции `DecompBinE2` анализировать алгоритм декомпозиции. В этой функции используется

`Set(a, b)` – функция установки коэффициентов трехчлена,

`Make32(x)` – функция, вычисляющая выражение c при известных a и b , которые задаются функцией `Set`.

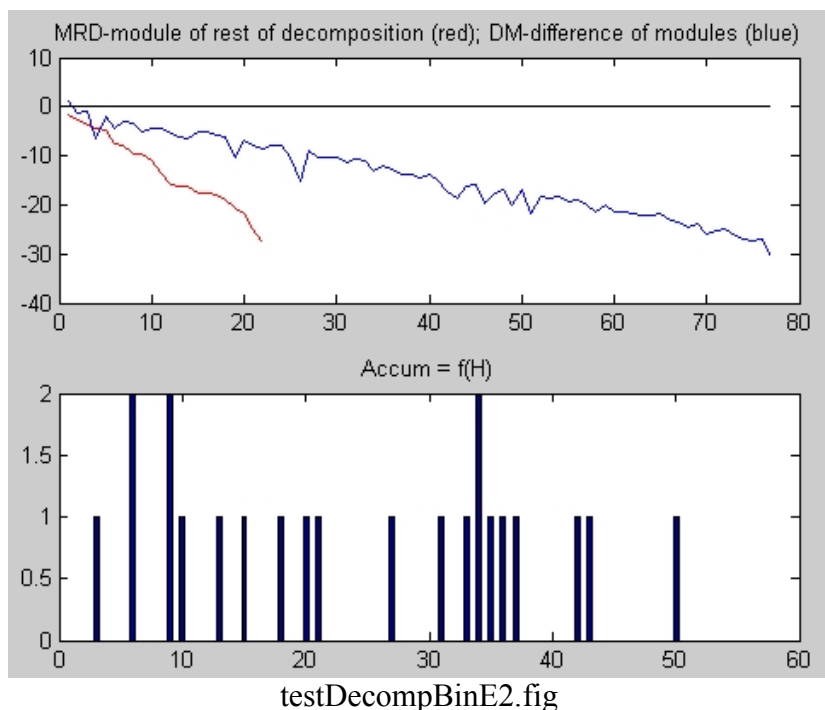
В функции `testDecompBinE2`

- вычисляется логарифмическая ошибка вычисления корня трехчлена, т.е. количество верных разрядов er ,
- вычисляется остаток $Rest$,
- строятся зависимости $DM(Iter)$ и $MRD(IterPlus)$ в первом окне - см. фиг. `testDecompBinE2.fig`,
- строится гистограмма значений разрядов массива (6) во втором окне - см. фиг. `testDecompBinE2.fig`.

Эта фигура построена для трехчлена

$$(0.7 + 0.25j)x^2 + (0.25 - 0.25j)x = 0.241 + 0.222j,$$

корнем которого является $x = 0.5153 + 0.319j$.



Функция `testOkrug4` вычисляет корень нормальных квадратных трехчленов (1), у которых числа x^2 лежат в первом полуквадранте и на окружности радиуса m . В этой функции дополнительно обозначено:

`pixels` - количество проверяемых точек,

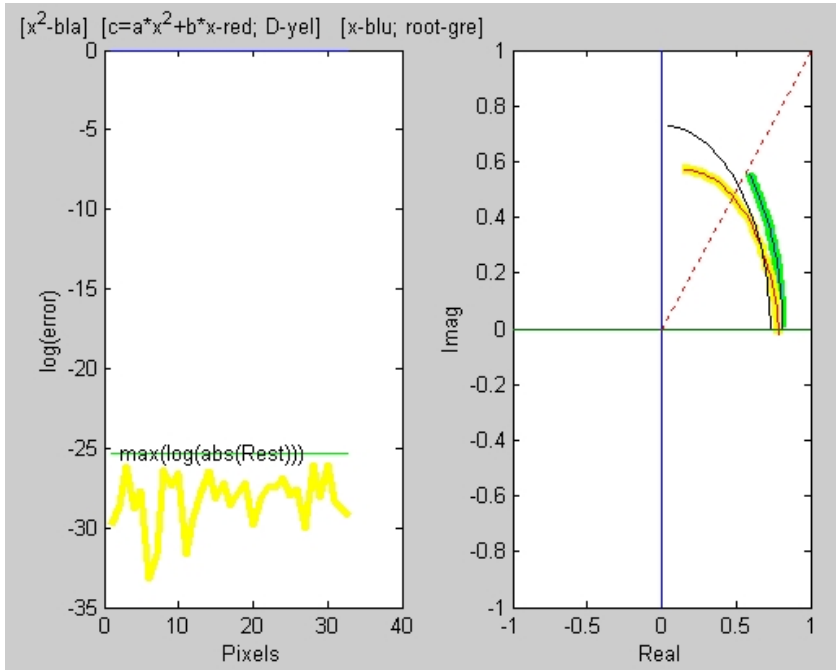
`x2` - число x^2 ,

`root` – вычисленный корень,

$$D = a \cdot \text{root}^2 + b \cdot \text{root}.$$

Функция `testOkrug4` строит

- зависимость относительной логарифмической ошибки декомпозиции и максимальную абсолютную ошибку остатка – см. первое окно на фиг. `testOkrug4.fig`,
- кривые, на которых расположены точки c , D , x , $x2$, `root` – см. второе окно на фиг. `testOkrug4.fig`.



testOkrug4.fig

Задача вычисления корня квадратного трехчлена усложняется, если местоположение корня на плоскости не известно – т.е. неизвестен квадрант корня и ограничение на его модуль. В этом случае для определения квадранта отыскивается минимум

$$Q = \min_q \left(\text{abs} \left(c - (aq^2 + bq) \right) \right), \quad q \in \{1, -1, j, -j\}$$

Величина q , соответствующая Q , определяет квадрант корня и является начальным значением для вычисления корня $x_0=q$. Функция `rotor3` выполняет эти вычисления.

Для приблизительного определения модуля корня отыскивается минимум

$$S = \min_n \left(\text{abs} \left(c - (as^2 + bs) \right) \right), \quad s = 2^n, \quad n = \overline{n_{\min}, n_{\max}}.$$

Величина n , соответствующая S , определяет приблизительное значение модуля корня и является начальным значением для вычисления корня $x_0=S$. Окончательное начальное значение для вычисления корня произвольном $x_0 = S*q$. Функция `shcalorN` выполняет эти вычисления.

Функция **DecompBinE205** реализует алгоритм для построения разложения (4) при произвольном начальном значении x_0 (в отличие от функции **DecompBinE2**, где $x_0=1$).

Функция **DecompBinE2a2** реализует алгоритм для построения разложения (4) в общем случае. В ней используются функции **DecompBinE205**, **rotor3** и **shcalorN**, описанные выше.

Функция **testDecBinAn5** обращается к функции **DecompBinE2a2** и вычисляет корень множества квадратных трехчленов вида (1), где

- 1) коэффициент $a = 1$,
- 2) коэффициент b принимает значения из множества
$$b \in d \cdot \{0, 1, 1-j, -j, -1-j, -1, -1+j, j\},$$
а множитель d принимает значения из множества
$$d = 2^m, \quad m \in \{-n, -n+1, \dots, 0, \dots, n-1, n\},$$
- 3) коэффициент c таков, что один из корней удовлетворяет условию
$$0.5 \leq q \cdot \operatorname{Re} X \leq 1, \quad 0.5 \leq q \cdot \operatorname{Im} X \leq 1,$$
$$q \in \{1, -1, j, -j\} \quad (2)$$

Эта функция строит восемь областей по числу значений коэффициента b , а в каждой области для каждой комбинации величин q и d проверяет (pixels^2) вариантов – см. также рис. 11. Заметим, что вариант $b=0$ соответствует извлечению квадратного корня из комплексного числа.

Итак, каждое из восьми окон содержит все корни, вычисленные при различных значениях величин q и d . Цвет точки указывается в зависимости от значения **maxAccum** так:

- синий, если **maxAccum**=1,
- красный, если **maxAccum**=2,
- зеленый, если **maxAccum**=3,
- черный, если **maxAccum**>3.

Кроме того, функция **testDecBinAn5** вычисляет максимальную кратность **maxAccum**, максимальную ошибку **MaxError**, количество экспериментов **Experiments**, разрядность кода результата **Bits**, среднее количество итераций на один разряд **AsumIter** и среднее количество успешных итераций на один разряд **AsumIterPlus**. В частности, при **Hmin**=-12, **Hmax**=55, **pixels**=13, **n**=2 имеем:

maxAccum=5, MaxError=7e-7, Experiments=27040, Bits=67,
AsumIter=2.3, AsumIterPlus=0.7.

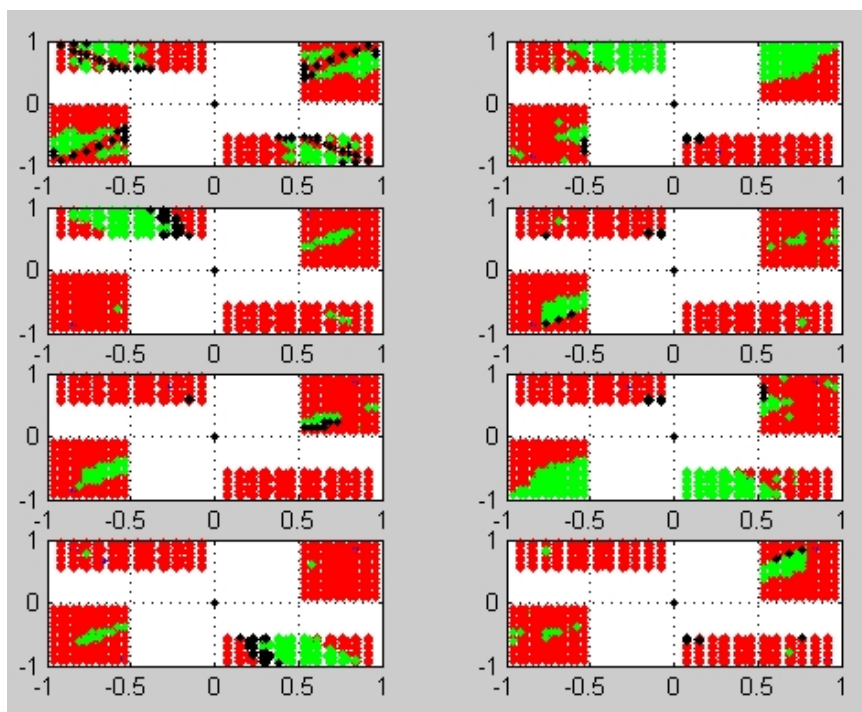
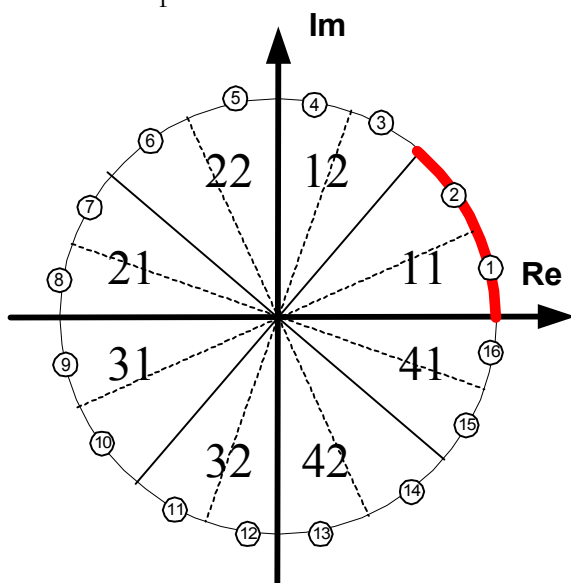


Рис. 11.

3.3. Root - нормализация при вычислении одного корня степенного полинома

3.3.1. Введение

Выше показано, что для декомпозиции (5.4) корень трехчлена - число x должно находиться в первом полуквадранте – см. также нумерацию полуквадрантов на фиг. 1. Кроме того, число x должно удовлетворять условию $|x| \approx 1$. При этом (как указывалось) кратность биномов не превышает 2.



Фиг. 1.

Преобразование, которое преобразует трехчлен так, что один из его корней x попадает в первый полуквадрант и $|x| \approx 1$, будем называть *root-нормализацией*. Она выполняется в два этапа: root-нормализация по углу и root-нормализация по модулю. В обоих случаях используется следующее соотношение. Рассмотрим уравнение

$$c = \frac{a}{s^2} \cdot z^2 + \frac{b}{s} \cdot z. \quad (a)$$

Очевидно, корни уравнений (4.1) и (a) связаны зависимостью

$$z = x \cdot s. \quad (b)$$

3.3.2. Root-нормализация по углу

Предположим, что окружность на фиг. 1 имеет единичный радиус. В центре дуги каждого *четвертьквадранта* выделяются точки, которые будем называть базовыми:

$$y_k = e^{j(\varphi_k + \pi/32)}, \quad \varphi_k = \frac{(k-1)\pi}{16}, \quad k = \overline{1, 16}. \quad (1)$$

Заметим, что квадрат комплексного числа, изображаемого *первой* базовой точкой, находится в первом полуквадранте. Рассмотрим уравнение (4.1) и выражения

$$D_k = a \cdot y_k^2 + b \cdot y_k, \quad (2)$$

$$m_k = |D_k - c|, \quad (3)$$

$$m_n = \min_k(m_k). \quad (4)$$

Очевидно, базовая точка y_n , где n определяется из условия (4), является ближайшей к одному из корней x_1 уравнения (4.1).

В уравнениях (а) и (в) может быть

$$s = e^{-j\varphi_n}. \quad (7)$$

Таким образом, решение уравнения (4.1) может быть заменено на решение *нормального* уравнения

$$c = a_n \cdot z^2 + b_n \cdot z, \quad (8)$$

один из корней z_1 которого находится в первом полуквадранте. При этом

$$a_n = a \cdot e^{2j\varphi_n}, \quad b_n = b \cdot e^{j\varphi_n}. \quad (9)$$

После определения корня z_1 нормального трехчлена (8) корень исходного трехчлена (4.1) определяется как

$$x = z \cdot e^{j\varphi_n}. \quad (10)$$

В целом нормализация трехчлена требует 11 умножений на константы вида $e^{j\varphi}$.

Приведение трехчлена к *первому четвертьквадранту* выполняет функция **rotor**, которая вычисляет коэффициенты уравнения (8).

3.3.3. Root-нормализация по модулю

Эта нормализация использует уравнения (а) и (в). Выбирается такое $s = 2^k$, где целое (положительное или отрицательное) число, при котором $|x| \approx 1$. Это условие выполняется, если

$$\left| 2^{2(k+1)}a + 2^{k+1}b - c \right| \geq \left| 2^{2k}a + 2^k b - c \right| \leq \left| 2^{2(k-1)}a + 2^{k-1}b - c \right|. \quad (11)$$

Приведение трехчлена к такому, у которого $|x| \approx 1$, *первому четвертьквадранту* выполняет функция `shcalor`, которая вычисляет соответствующее $s = 2^k$.

3.3.4. Полная root-нормализация

Функция `DecompBinE2a` вычисляет корень произвольного трехчлена. При этом она обращается к функции `DecompBinE2` и использует функции `rotor` и `shcalor` для выполнения root-нормализации по углу и модулю.

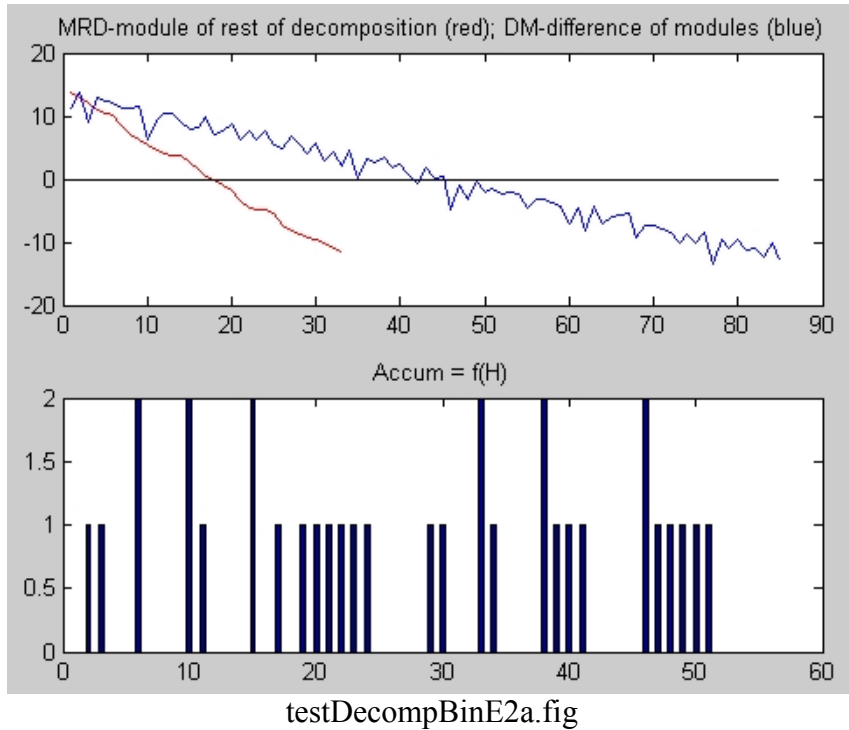
Функция `testDecompBinE2a` позволяет при обращении к функции `DecompBinE2` анализировать алгоритм декомпозиции. В этой функции

- вычисляется логарифмическая ошибка вычисления корня трехчлена, т.е. количество верных разрядов `er`,
- вычисляется остаток *Rest*,
- строятся зависимости `DM(Iter)` и `MRD(IterPlus)` в первом окне - см. фиг. `testDecompBinE2a.fig`,
- строится гистограмма значений разрядов массива (6) во втором окне - см. фиг. `testDecompBinE2a.fig`.

Эта фигура построена для трехчлена

$$(0.7 + 0.25j)x^2 + (0.25 - 0.25j)x = 8054.2 + 26195j,$$

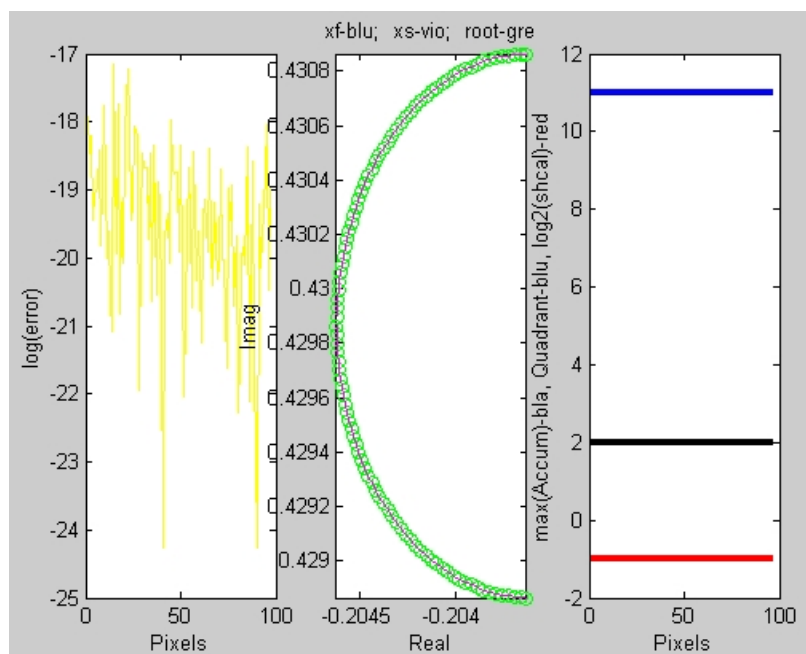
корнем которого является $x = -171.75 - 85.84j$. При этом номер четверть квадранта `semiqr=18` и нормализующий множитель $s = 2^7$.



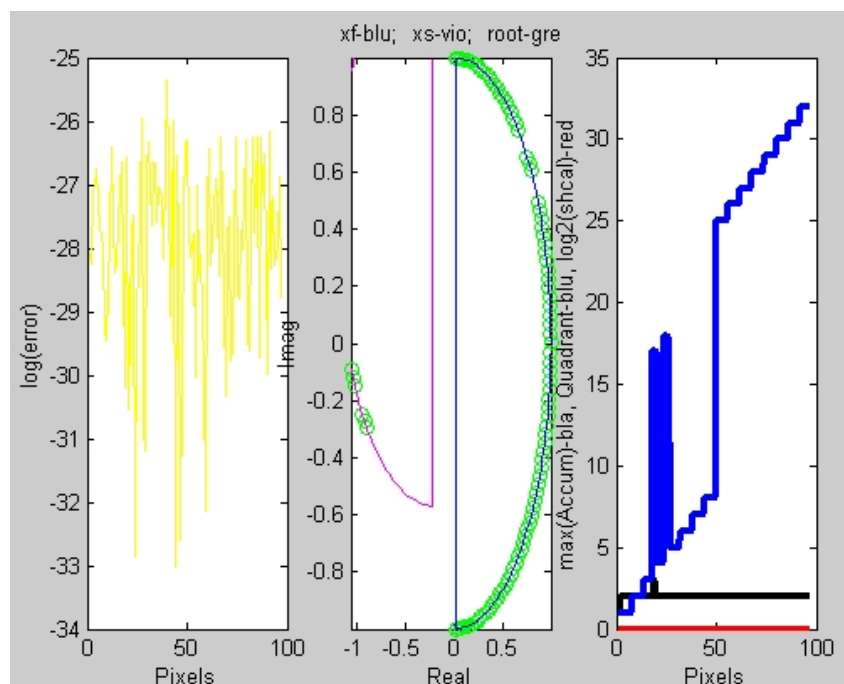
Функция `testOkrug5` вычисляет корень квадратных трехчленов, у которых числа x^2 лежат на окружности радиуса m . В этой функции обозначения совпадают с обозначениями в функции `testOkrug4`. Функция `testOkrug5` строит

- относительную логарифмическую ошибку декомпозиции – см. первое окно на фиг. `testOkrug5.fig`,
- кривые, на которых расположены точки xf , xs , $root$ – см. второе окно на фиг. `testOkrug5.fig`,
- номер сектора `Quadrant`, в который переносится точка при $root$ -нормализации, максимальное значение кратности бинорма $\max(\text{Accum})$ и нормализующий множитель $k = \log_2(s)$ точки - см. третье окно на фиг. `testOkrug5.fig`.

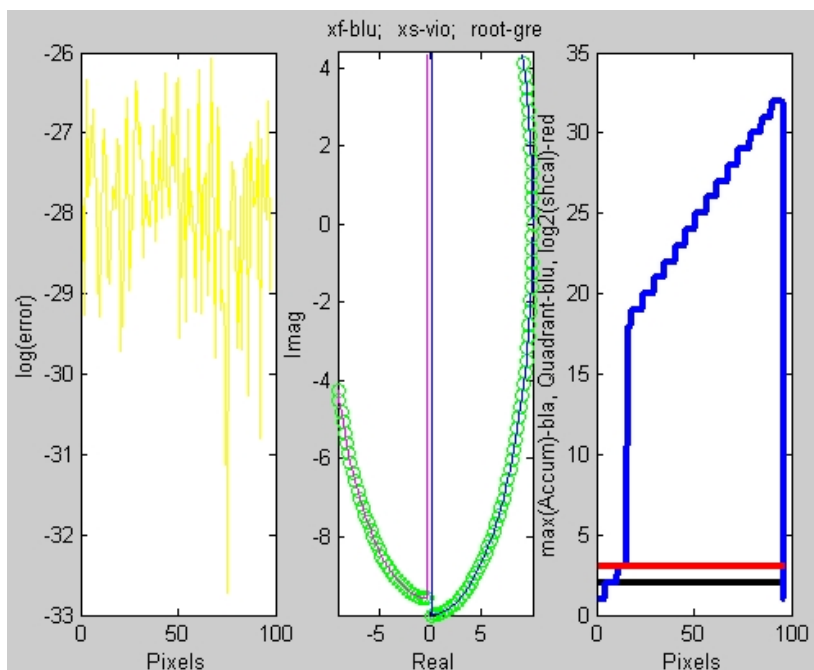
На следующих фигурах оказаны результаты работы функции `testOkrug5` при различных m .



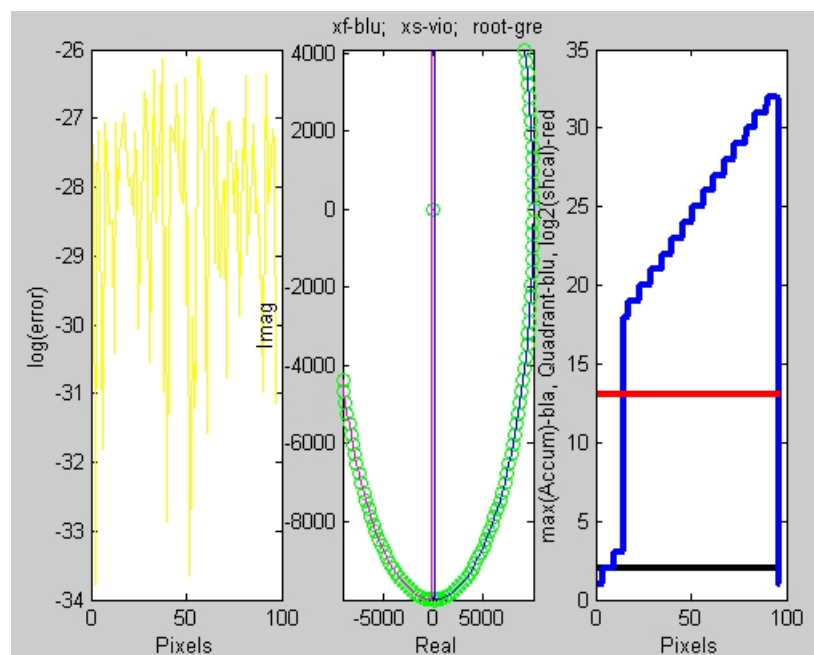
testOkrug5.fig (m=0.001)



testOkrug5.fig (m=1)



testOkrug5.fig (m=10)



testOkrug5.fig (m=10000)

Часть 3. Коды геометрических фигур

Аннотация

В этой части книги рассматриваются различные варианты процессоров, предназначенных для аффинных преобразований многомерных фигур - плоских и пространственных. Эти процессоры ориентированы на аффинное преобразование неструктурированных геометрических фигур с произвольным характером распределения точек. При этом используется нетрадиционная форма представления данных, использующая описанную выше теорию кодирования комплексных чисел и векторов. Задачи аффинного преобразования пространства широко используются в науке и технике.

Здесь описывается теория кодирования фигур – структура кодов, алгоритмы кодирования, декодирования плоских и пространственных фигур, арифметические операции с плоскими и пространственными фигурами. Теория дополняется многочисленными примерами. Рассматривается несколько вариантов геометрического процессора – представление данных, операционные блоки, техническая реализация алгоритмов кодирования, декодирования и арифметических операций. Оценивается быстродействие этих процессоров.

Содержание

Предисловие \ 3-4

Глава 1. Введение \ 3-6

Глава 2. Прототипы \ 10

Глава 3. Арифметика комплексных чисел и векторов \ 16

Глава 4. Векторный процессор \ 24

Глава 5. Теория кодирования фигур \ 31

Глава 6. Геометрический процессор \ 83

Глава 7. Сравнительный анализ \ 109

Обозначения \ 115

Список примеров \ 118

Список таблиц \ 119

Список рисунков \ 121

Предисловие

В этой части книги рассматривается полная малоизвестная теория и запатентованные инженерные решения для компьютерной арифметики геометрических фигур – плоских и пространственных. Эта теория ориентирована на **аффинное** преобразование неструктурированных геометрических фигур с произвольным характером распределения точек. Именно выявление структуры и является целью преобразования. Поэтому наблюдаемый объект может быть определен только как пространство, каждая точка которого имеет некоторые характеристики. Задачи аффинного преобразования пространства широко используются в науке и технике – в медицине, обработке и визуализация данных, астрономии, сейсмологии и т.д. Наиболее яркими и хорошо известными примерами применения аффинного преобразования могут служить компьютерная томография (см., например, [56]) и сжатие информации для телекоммуникационных систем (см., например, [57]).

Здесь рассматриваются аффинные преобразования (перемещения, повороты, масштабирование, сдвиги) n -мерных где $n=2, 3, 4$. Обычно указанные преобразования фигур выполняются путем вычисления координат точек преобразованной фигуры по известным координатам точек исходной фигуры. Однако такой метод требует много процессорного времени, так как вычисление координат выполняется последовательно для всех точек и требует нескольких операций для каждой точки (например, для вычисления новых координат при аффинном преобразовании плоской фигуры требуется по 4 операции сложения и умножения).

Указанные задачи включают операции с комплексными числами, так как точка на плоскости может быть представлена комплексным числом. При этом одноименные операции могут выполняются одновременно с множеством комплексных чисел. Для решения подобных задач используются процессоры с архитектурой SIMD (Single Instruction, Multiple Data). Однако эти процессоры оперируют с действительными числами. При этом каждая операция с комплексными числами требует нескольких операций с действительными числами - действительными и

мнимыми частями этих комплексных чисел. Аналогично, геометрические преобразования в трехмерном пространстве используют операции с трехмерными векторами – тройками действительных чисел. При этом каждая операция с векторами требует еще больше операций с действительными числами. Все это значительно увеличивает время вычислений. Кроме того, множество комплексных чисел и векторов, описывающих фигуру, занимает большой объем памяти. Имеется, таким образом, потребность в методе и системе для эффективных SIMD вычислений с множеством комплексных чисел и векторов, описывающих фигуру. Эти вычисления должны быть эффективными по времени вычислений и требуемой памяти.

Эта часть содержит 7 глав. В **первой главе** рассматривается постановка задачи. Во **второй главе** рассматриваются известные устройства для преобразования фигур – с одним и несколькими вычислителями. Эта информация используется далее для аналогий и сравнения. В **третьей главе** приводятся основы компьютерной арифметики комплексных чисел и векторов – теория и аппаратные решения. Эта глава необходима, поскольку при кодировании и декодировании геометрического кода фигуры используются коды комплексных чисел и векторов. В **четвертой главе** описывается векторное арифметическое устройство, основанное на векторной компьютерной арифметике, изложенной в предыдущей главе. В **пятой главе** описывается теория кодирования фигур – структура кодов, алгоритмы кодирования, декодирования плоских и пространственных фигур, арифметические операции с плоскими и пространственными фигурами. Теория дополняется многочисленными примерами. В **шестой главе** рассматривается устройство растрового геометрического процессора – представление данных, операционные блоки, техническая реализация алгоритмов кодирования, декодирования и арифметических операций, а также оценивается быстродействие этого процессора. В **седьмой главе** приводится сравнение характеристик арифметических устройств и блоков оперативной памяти, предназначенных для операций с фигурами. Сравниваются рассмотренные в предыдущих главах традиционные устройства, устройства векторной арифметики и устройства арифметики геометрических фигур.

Глава 1. Введение

Решение задач аффинного преобразования геометрических фигур может быть значительно ускорено при специальном кодировании множества комплексных чисел и векторов. В связи с этим далее рассматривается метод представления множества комплексных чисел и векторов так называемым геометрическим кодом, а затем описываются различные операции с ним, а также аппаратная реализация этих операций. Геометрические коды предложены в [22, 26] и рассмотрены также в [28-32, 39]. При построении геометрического кода используется метод представления комплексных чисел и векторов единым двоичным кодом [21, 32, 34].

Этот метод заключается в том, что координаты двоичных кодов комплексных чисел и векторов изображается единым двоичным кодом. Его объем существенно меньше суммарного объема массива исходных двоичных кодов. Относительное сокращение объема зависит от количества кодируемых чисел и растет с увеличением этого количества. Кодируемое множество комплексных чисел НЕ структурировано. Можно говорить о их случайном множестве. Кодируемые комплексные числа и векторы являются множеством координат (что существенно), с которыми надо выполнять вычисления. Дополнительная информация о точках (например, цвет), если она не участвует в вычислениях, кодированию не подлежит и должна храниться в отдельном массиве — массиве атрибутов. Геометрический код хранит (помимо координат) также информацию о связях каждой точки с ее атрибутами.

С геометрическим кодом выполнимы арифметические операции (алгебраическое сложение, умножение комплексных чисел и векторов, аффинное преобразование). Эти операции эквивалентны групповым операциям с координатами всех точек одновременно.

Важно отметить, что время выполнения операции с геометрическим кодом равно времени выполнения одноименной операции с парой чисел, *если* весь геометрический код помещается в оперативном регистре арифметического устройства. Предполагается, что исходные коды являются кодами с фиксированной точкой (например, координатами точки на экране).

Предлагается также метод фрагментации геометрического кода, который позволяет оперировать с отдельными фрагментами геометрического кода, если разрядность регистра арифметического устройства не достаточна для хранения полного кода.

Важно отметить, что геометрический код позволяет оперировать с фигурой, как с целым, единственным объектом. При этом объем данных (кодов координат) сокращается. Однако (и это важно подчеркнуть для исключения заблуждений) геометрический код не сжимает сами геометрические фигуры. Предполагается, что кодируемая геометрическая фигура описывается случайным набором точек и не имеет какой-либо структуры, что характерно для растровых изображений.

В общем, применение ГС сокращает объем данных в n раз, где n - разрядность линейных кодов. Скорость выполнения групповых операций с геометрическими кодами во много раз превышает скорость таких же операций с массивом чисел. Общее время выполнения вычислений сокращается еще и за счет уменьшения времени доступа к данным.

Далее рассматриваются три вида арифметических устройств –

- традиционное, оперирующее с действительными числами и содержащее несколько вычислителей, работающих параллельно,
- векторное, оперирующее с предложенными кодами векторов и также содержащее несколько вычислителей, работающих параллельно и
- геометрическое, оперирующее с геометрическими кодами фигур.

Рассматривается также устройство специализированной оперативной памяти, выполненное на основе метода кодирования геометрических фигур.

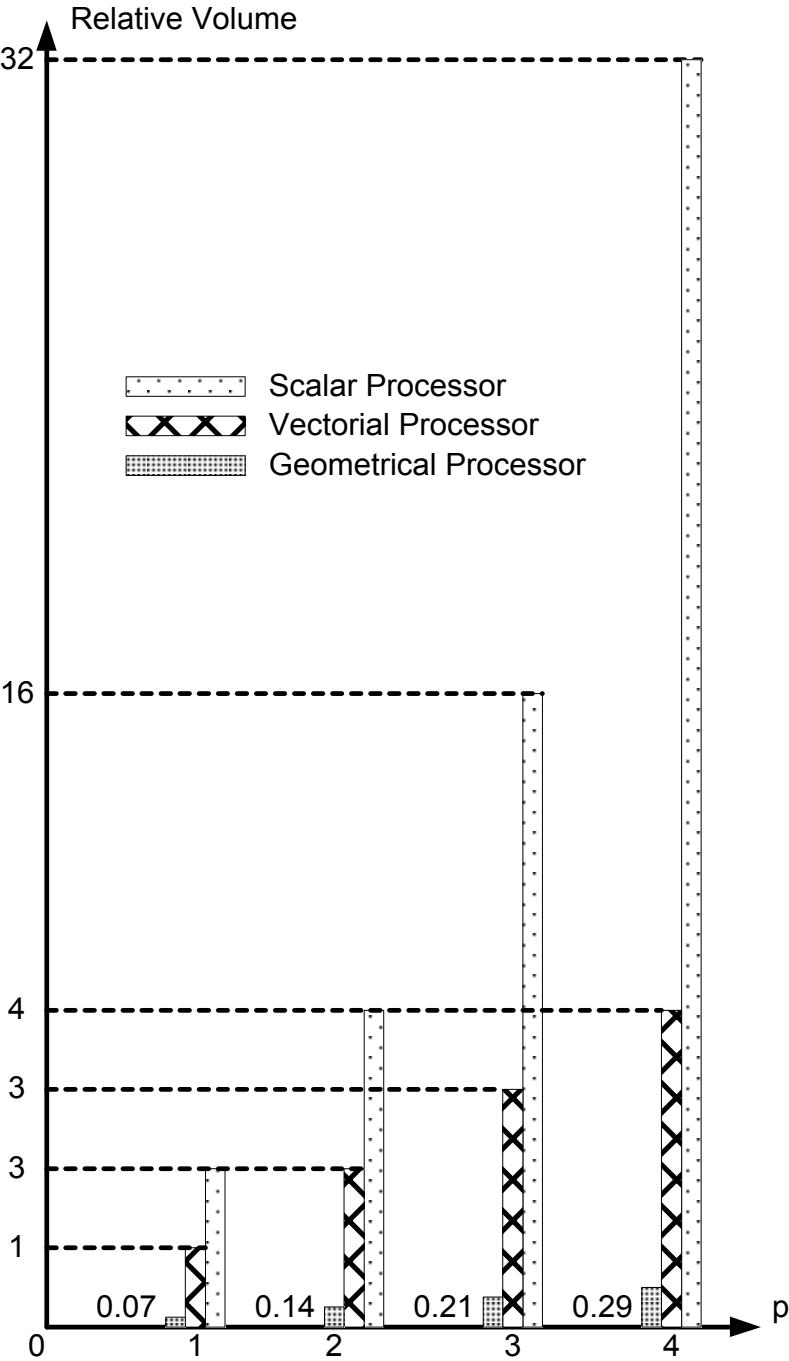


Рис. 1.1. Гистограмма относительного объема арифметических устройств.

Приводится сравнение качества этих устройств. Естественно характеризовать *качество* устройства отношением объема устройства к количеству определенных процедур, выполняемых устройством в единицу времени. Будем называть это отношение *относительным объемом* устройства. Для арифметических устройств типовой процедурой является аффинное преобразование. Для устройств оперативной памяти типовой процедурой является либо поиск точки с данными координатами в неупорядоченном массиве, либо обычный доступ, либо, наконец, заданная смесь этих процедур.

На рис. 1.1 приведена гистограмма относительного объема перечисленных арифметических устройств в зависимости от размерности p кодируемого пространства. Единицей измерения на этом рисунке является величина 14^*M , где M - количество точек кодируемого пространства. Например, при $p=3$ величины относительного объема рассмотренных арифметических устройств относятся как (84:14:1).

Для сравнения вариантов исполнения оперативной памяти предположим, что в данной задаче операции чтения\записи встречаются в H раз чаще, чем операции поиска по данным координатам. Показано, что относительный объем специализированного запоминающего устройства меньше относительного объема традиционного запоминающего устройства в $(\sim M/10H)$ раз.

Глава 2. Прототипы

2.1. Представление данных

Задача, которую должны решать предлагаемые ниже процессоры состоит в следующем. Дано множество (M) точек в p -мерном пространстве. Точки образуют область определения, которая представляет собой p -мерный куб, и распределены в этой области по узлам равномерной сетки. Каждая координата представляется n -разрядным кодом с фиксированной точкой. Шаг сетки равен значению младшего разряда этого кода. Каждая точка характеризуется координатами и атрибутом (некоторыми величинами, поставленными в соответствие каждой точке). Будем говорить, что таким образом определена фигура в p -мерном пространстве или, просто, p -мерная фигура F . Необходимо найти

другую фигуру F_{ω} полученную аффинным преобразованием фигуры F . Аффинное преобразование описывается p -мерной *матрицей преобразования* (для центроаффинного преобразования фигуры) и p -мерным *вектором переноса* (для переноса фигуры в каком-либо направлении). Каждый элемент матрицы преобразования представлен r -разрядным кодом с фиксированной точкой. Каждый элемент вектора переноса представлен n -разрядным кодом с фиксированной точкой. Общая разрядность параметров аффинного преобразования равна

$$a = p \cdot n + p^2 \cdot r. \quad (2.1.1)$$

В памяти процессора для каждой точки хранятся пары «*координаты-атрибут*». Очевидно, максимальное количество кодируемых точек

$$M = 2^{pn}. \quad (2.1.2)$$

Адреса пар остаются неизменными во время решения задачи для того, чтобы по измененным в процессе преобразования координатам можно было бы найти атрибут точки. Кроме того, при некоторых преобразованиях координаты точек могут совместиться. Таким образом, каждая точка определяется триадой «*адрес-координаты-атрибут*».

Будем в дальнейшем называть процессор для решения указанной задачи, растровым геометрическим процессором - **RGP**. Далее рассматриваются различные варианты арифметических устройств для этого процессора.

2.2. Простейшее арифметическое устройство

Предварительно рассмотрим простейшую конструкцию скалярного арифметического устройства **SAU** (см. рис. 2.2.1), которая будет использоваться для аналогий и сравнения более сложных конструкций. В этом устройстве применен простейший умножитель последовательного типа, содержащий только сдвигатель и сумматор.

В этом SAU достаточно предусмотреть $(n+r)$ -разрядный сумматор, $(n+r)$ -разрядный умножитель, $(n+r)$ -разрядный регистр координаты, (r) -разрядный регистр выбранного параметра и a -разрядный регистр всех параметров преобразования - компонент матрицы преобразования и вектора переноса - см. (2.1.1). Кроме того, это SAU содержит мультиплексор для выбора параметра и управляющее устройство. Аффинное преобразование каждой точки содержит p^2 умножений и

$$D = p(p-1) = 2, 6, 12, \dots \text{ if } p = 2, 3, 4, \dots (2.2.1)$$

сложений.

Сумматор в этом АУ служит для сложения координаты с одной из компонент вектора переноса и для сложения частичного произведения со множимым при умножении. Регистр суммы в нем содержит триггеры со счетным входом и поэтому может быть совмещен с регистром одного из слагаемых - регистром первоначального значения координаты.

Умножитель в этом АУ реализует следующий алгоритм:

0. Задан r -разрядный множитель A , представляющий одну компоненту матрицы преобразования, и n -разрядное множимое B , представляющее одну координату точки.
1. Вначале частичное произведение равно 0, а множимое B расположено так, что его младший 0-разряд совмещен со

- старшим $(n-1)$ -разрядом α_{n-1} множителя A . Номер текущего разряда множителя $t=n-1$, т.е. $\alpha_t = \alpha_{n-1}$.
2. Сложение частичного произведения со множимым B , если $\alpha_t = 1$.
 3. Сдвиг множимого B на 1 разряд вправо и уменьшение текущего номера $t := t - 1$.
 4. Прекращение вычисления, если $t < 0$, или переход к п. 2. В результате образуется $(n+r)$ -разрядное произведение C .

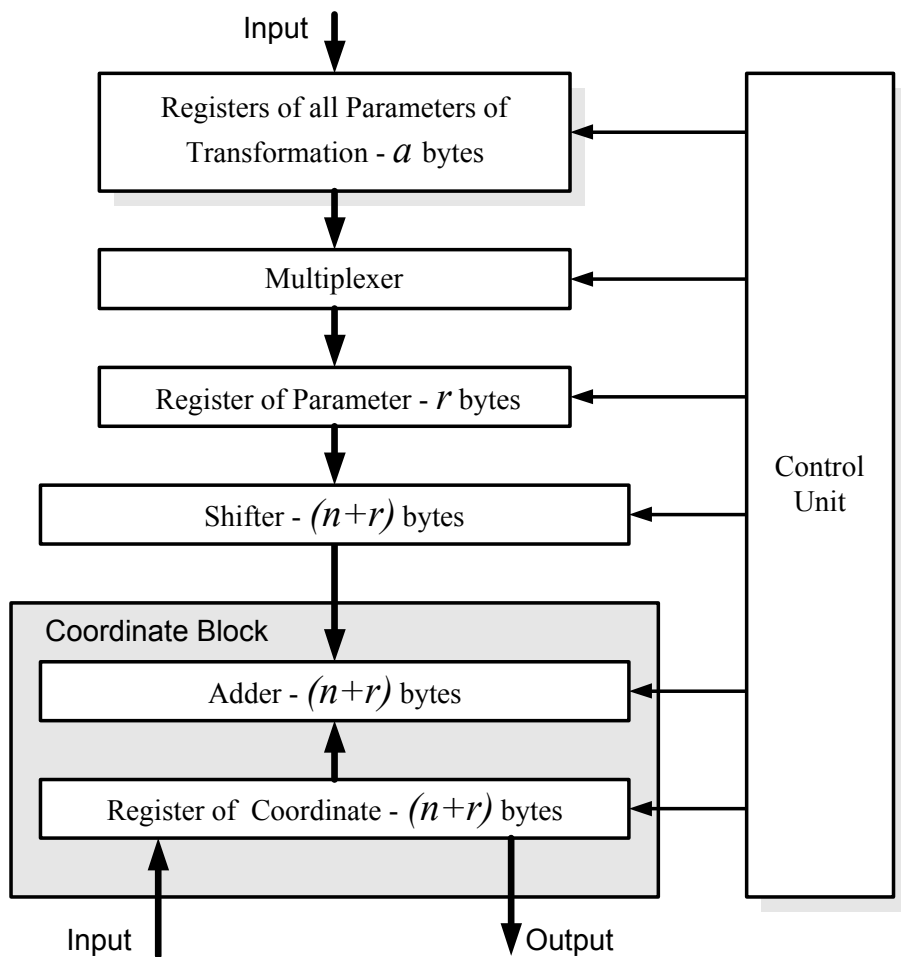


Рис. 2.2.1. Простейшее арифметическое устройство.

Таким образом, умножитель содержит только сдвигатель на 1 разряд вправо и сумматор.

Разрядность результатов аффинного преобразования может достигать величины $(n+r)$. Это приводит к тому, что часть точек может выйти за пределы первоначального p -куба. При этом можно

1. либо исключить эти точки из заданного множества (поставив соответствующий признак в атрибут этой точки),
2. либо округлить все коды координат (отбросив младших разрядов) и изменить значение шага сетки (который является параметром всей фигуры).

И в том, и в другом случае может быть, что

1. В некотором узле сетки присутствует несколько точек. Атрибут узла определяется как функция атрибутов всех точек, оказавшихся в этом узле. Такая функция известна, например, для вычисления суммарного цвета совпавших точек. Если, например, атрибут является интенсивностью монохромного цвета, то эта функция является средним арифметическим интенсивности объединяемых векторов.
2. В некотором узле сетки отсутствует какая-либо точка. Атрибут узла определяется как функция атрибутов всех соседних узлов.

Рассмотрим перечень команд процессора, реализуемых в SAU:

- Прием параметров преобразования
- Прием координаты
- Сложение с компонентой вектора переноса (D модификаций – см. (2.2.1))
- Умножение на компоненту матрицы преобразования (p^2 модификаций)
- Выдача координаты
- Округление

2.3. Арифметическое устройство с прямоугольными кодами.

Не стремясь к экономии аппаратуры, можно предложить арифметическое устройство **MSAU**, содержащее множество (M) элементарных арифметических устройств SAU, работающих

параллельно. В этом устройстве коды одной координаты всех точек фигуры образуют массив, который будем называть *прямоугольным кодом чисел* - **RCS**. RCS содержит M регистров разрядностью $(n+r)$. MSAU в целом содержит M сумматоров разрядностью $(n+r)$, M умножителей разрядностью $(n+r)$, RCS и один a -разрядный регистр параметров. Это MSAU выполняет групповые операции - сложение RCS с кодом переноса и умножение RCS на код элемента матрицы центроаффинного преобразования. Аффинное преобразование фигуры содержит p^2 групповых умножений и D групповых сложений.

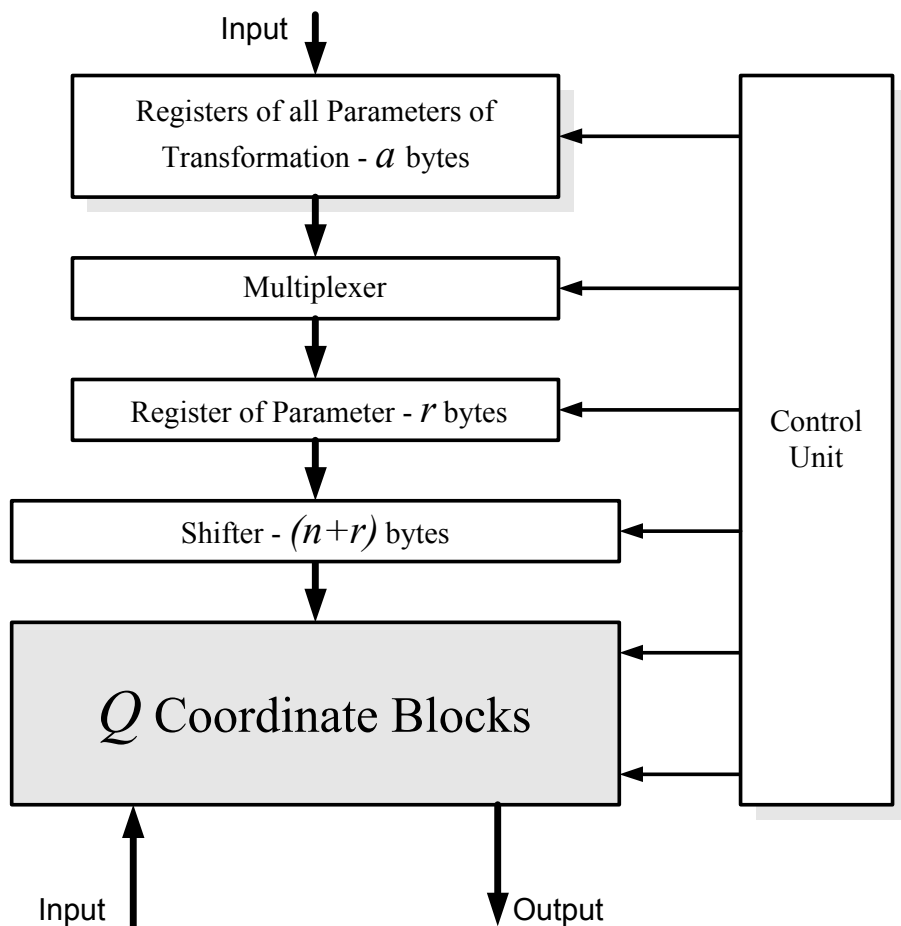


Рис. 2.2.2. Арифметическое устройство с фрагментарными прямоугольными кодами.

Вариант MSAU имеет очень большой объем и его реализация находится за пределами возможностей сегодняшней технологии. Поэтому рассмотрим еще один вариант, занимающий промежуточное место между АУ с одиночными и групповыми операциями. Для этого разделим RCS на несколько фрагментов

RCS_q каждый из которых содержит (Q) регистров разрядностью $(n+r)$. Арифметическое устройство **FSAU** содержит в целом Q сумматоров разрядностью $(n+r)$, Q умножителей разрядностью $(n+r)$, RCS_q и один a -разрядный регистр параметров. Схема FSAU представлена на рис. 2.2.2. Эта схема идентична схеме рис. 2.2.1, за исключением того, что в ней используются Q координатных блоков, выделенных на рис. 2.2.1.

FSAU выполняет групповые операции с координатами точек фрагмента фигуры. На нем аффинное преобразование фигуры содержит Qb^2 групповых умножений и QD групповых сложений.

Глава 3. Арифметика комплексных чисел и векторов

3.1. Вступление

В качестве линейных кодов могут использоваться рассмотренные в части 2 коды комплексных чисел. - см. табл. 1.1.6.5. Для построения геометрических кодов наилучшими являются системы кодирования комплексных чисел 1, 2. В этих системах комплексный код представляется некоторой композицией кодов действительного числа по основанию «-2». Аналогично, коды многомерных векторов также представляются некоторой композицией кодов действительного числа по основанию «-2» - см. раздел 1.1.7.

Арифметика линейных кодов комплексных чисел и векторов рассмотрена в части 1. Далее подробнее (с ориентацией на применение в геометрическом процессоре) рассматриваются умножение, кодирование и декодирование комплексных чисел и векторов

3.2. Умножение многомерных векторов

3.2.1. Метод умножения многомерных и векторов

Необходимо найти произведение векторов $C=A*B$, где множитель и множимое имеют соответственно разложения

$$A = \sum_h \alpha_h f(\rho, h), \quad B = \sum_k \beta_k f(\rho, k).$$

Код произведения определяется как $C = \sum_h [B\alpha_h f(\rho, h)]$.

Поскольку $\alpha_h = \{0,1\}$, то умножение состоит только из операций умножения на базовую функцию $f(\rho, h)$ и суммирования. Рассмотрим умножение на базовую функцию для двух важных для нашего приложения случаев.

3.2.2. Умножение на базовую функцию для векторов по основанию (3.3.10).

В этом случае умножение множимого на базовую функцию равносильно сдвигу на h разрядов. Таким образом, умножение кодов в этой системе сводится к последовательно выполняемым операциям сдвига и сложения.

3.2.3. Умножение на базовую функцию для векторов по основанию (3.3.7).

Разрядность кода множителя $N = n \cdot m$, где n – размерность вектора. Код множителя можно разбить на m групп, а в каждой t -группе рассматривать первый (младший) разряд, второй разряд, ... i -разряд, ... n -разряд. Группы будем нумеровать также, как разряды кода, справа налево от 0 до $(m-1)$. При этом умножение множимого на базовую функцию (если соответствующий множителя равен 1) состоит из двух действий (которые совмещаются во времени):

1. Сдвиг множимого на $h = n \cdot t$ разрядов, если рассматривается t -группа разрядов множителя.
2. Умножение множимого B на орт в зависимости от номера i разряда в группе:

$$B_i = E_i B, \quad i = \overline{1, n} \quad (3.5.1)$$

- см. также (3.3.3). Например, при $n=2$, имеем:

$$B_1 = B, \quad B_2 = jB;$$

при $n=3$, имеем:

$$B_1 = iB, \quad B_2 = jB, \quad B_3 = kB;$$

при $n=3$, имеем:

$$B_1 = iB, \quad B_2 = jB, \quad B_3 = kB, \quad B_4 = mB$$

и т.д. Заметим, что преобразованные множимые B_i могут быть заготовлены перед умножением.

Вычисление по формуле (3.5.1) выполняется в соответствии с таблицей умножения вектора или формулой (3.2.2). Пусть в соответствии с (3.3.3)

$$B = E_1 b_1 + E_2 b_2 + \dots + E_n b_n, \quad (3.5.2)$$

В частности, для комплексных чисел используется табл. 3.2.2. Имеем:

$$B_1 = B, \quad B_2 = j(b_1 + jb_2) = -b_2 + jb_1.$$

Для трехмерных векторов используется табл. 3.2.1. Имеем:

$$B_1 = B,$$

$$B_2 = j(b_1 + jb_2 + kb_3) = -b_3 + jb_1 + kb_2,$$

$$B_{32} = k(b_1 + jb_2 + kb_3) = -b_2 - jb_3 + kb_1.$$

Для четырехмерных векторов используется табл. 3.2.3. Имеем:

$$B_1 = B,$$

$$B_2 = j(b_1 + jb_2 + kb_3 + mb_4) = -b_4 + jb_1 + kb_2 + mb_3,$$

$$B_3 = k(b_1 + jb_2 + kb_3 + mb_4) = -b_3 - jb_4 + kb_1 + mb_2,$$

$$B_4 = m(b_1 + jb_2 + kb_3 + mb_4) = -b_2 - jb_3 - kb_4 + mb_1.$$

Отсюда следует, что умножение кода вектора на орт состоит из инвертирования некоторых компонент и перестановки компонент кода вектора.

3.2.4. Умножение целых кодов векторов по основанию (3.3.10).

Рассмотрим систему кодирования n -мерных векторов по основанию (3.3.10). Для нашего приложения следует анализировать разряды множителя, начиная со старшего, и сдвигать множимое вправо. В соответствии с этим алгоритм умножения имеет вид:

1. Вначале частичное произведение равно 0, а множимое B расположено так, что его младший 0-разряд совмещен со старшим $(N-1)$ -разрядом α_{N-1} множителя A . Номер текущего разряда множителя $t=N-1$, т.е. $\alpha_t = \alpha_{N-1}$.

2. Сложение частичного произведения со множимым B , если $\alpha_t = 1$.
3. Сдвиг множимого B на 1 разряд вправо и уменьшение текущего номера $t := t - 1$.
4. Прекращение вычисления, если $t < 0$, или переход к п. 2.

3.2.5. Умножение целых кодов векторов по основанию (3.3.7).

Рассмотрим систему кодирования n -мерных векторов по основанию (3.3.7). Разрядность кода множителя $N = n \cdot m$. В этом случае алгоритм умножения имеет вид:

1. Подготавливаются n вариантов множимого B по формуле (3.5.1).
2. Будем рассматривать группы по n разрядов множителя. Вначале частичное произведение равно 0, а множимые B_i расположены так, что их младшие 0-разряды совмещены с разрядом множителя A , имеющего номер $N - n = n \cdot (m - 1)$. Номер текущей группы разрядов множителя $t = m$.
3. Рассматривается t -группа разрядов множителя A . В ней
 - 3.1. Рассматривается первый (младший) разряд $\alpha_{n(t-1)}$. Выполняется сложение частичного произведения со множимым B_1 , если $\alpha_{n(t-1)} = 1$.
 - 3.2. Рассматривается второй разряд $\alpha_{n(t-1)+1}$. Выполняется сложение частичного произведения со множимым B_2 , если $\alpha_{n(t-1)+1} = 1$.
 - ...
 - 3.i. Рассматривается i -разряд $\alpha_{n(t-1)+i}$. Выполняется сложение частичного произведения со множимым B_i , если $\alpha_{n(t-1)+i} = 1$.
 - ...

- 3.п. Рассматривается n -разряд α_{nt} . Выполняется сложение частичного произведения со множимым B_n , если $\alpha_{nt} = 1$.
4. Сдвиг множимого на n разряд вправо (напомним, что здесь n – размерность вектора) и уменьшение текущего номера $t := t - 1$.
5. Прекращение вычисления, если $t < 0$, или переход к п. 3.

3.2.6. Покомпонентное умножение многомерных векторов.

В отличие от простого умножения, в каждом цикле покомпонентного умножения значение множимого, с которым производится сложение, зависит от номера m разряда множителя (т.е. от того, к какой компоненте вектора множителя принадлежит этот разряд). Пусть

m - номер разряда множителя A ,
 k - целое число.

Если выполняется покомпонентное умножение *комплексного числа*

$$C = A * (X, Y),$$

то множимое B определяется следующим образом:

$$B = X, \text{ если } m = 3k,$$

$$B = Y, \text{ если } m = 3k+1.$$

Если выполняется покомпонентное умножение *трехмерного вектора*

$$C = A * (X, Y, V),$$

то множимое B определяется следующим образом:

$$B = X, \text{ если } m = 3k,$$

$$B = Y, \text{ если } m = 3k+1,$$

$$B = V, \text{ если } m = 3k+2.$$

Если выполняется покомпонентное умножение *четырёхмерного вектора*

$$C = A * (X, Y, V, W),$$

то множимое B определяется следующим образом:

$$B = X, \text{ если } m = 3k,$$

$$B = Y, \text{ если } m = 3k+1,$$

$$B = V, \text{ если } m = 3k+2,$$

$$B = W, \text{ если } m = 3k+3.$$

Как показано выше, покомпонентное умножение на заранее определенные тройки векторов эквивалентно скалярному и векторному умножению, умножению на число, центроаффинному преобразованию и т.д.

3.2.7. Скалярное и векторное умножения описаны в разделе 2.5.3

3.3. Алгоритмы и устройства для кодирования и декодирования комплексных чисел и векторов

Для построения геометрических кодов наилучшими являются системы кодирования 1, 2. Потому далее рассматриваются алгоритмы и устройства для кодирования и декодирования только в системах 1 и 2.

Кодируемое комплексное число представляется в виде $Z = X_\alpha + jX_\beta$, где X_α , X_β - действительная и мнимая части комплексного числ, являющиеся действительными (положительными или отрицательными) числами.

3.3.1. Кодирование комплексного числа в системе 1.

1. Кодирование действительных (положительных или отрицательных) чисел X_α , X_β из Р-кода в М-код. Вначале целесообразно предварительно кодировать только положительные числа и сохранить знаки $sign(X_\alpha)$, $sign(X_\beta)$ и М-коды чисел $|X_\alpha|$, $|X_\beta|$. Для кодирования действительных положительных чисел используется кодер положительного Р-кода в М-код. см. раздел 1.5.1.2. Затем следует на инверторе М-кодов см. раздел 1.2.3 вычислить числа $|X_\alpha| \cdot sign(X_\alpha)$, $|X_\beta| \cdot sign(X_\beta)$.
2. Формирование С-кода

$K(Z) = \dots \beta_m \alpha_m \dots \beta_1 \alpha_1 \beta_0 \alpha_0, \beta_{-1} \alpha_{-1} \beta_{-2} \alpha_{-2} \dots$ комплексного числа $Z = X_\alpha + jX_\beta$, который в дальнейшем представляется

в виде $K(Z) = \dots \gamma_m \dots$, где $\left\{ \begin{array}{l} \gamma_{2m} = \alpha_m \text{ if } m - \text{even} \\ \gamma_{2m+1} = \beta_m \text{ if } m - \text{odd} \end{array} \right\}$. Для

этого используется распределитель см. раздел 1.5.1.6.

3.3.2. Декодирование комплексного числа в системе 1.

1. Выделение из С-кода комплексного числа $Z = X_\alpha + jX_\beta$ четных и нечетных разрядов по правилу $\left\{ \begin{array}{l} \alpha_{m/2} = \gamma_m \text{ if } m - \text{even} \\ \beta_{(m-1)/2} = \gamma_m \text{ if } m - \text{odd} \end{array} \right\}$ и формирование из разрядов α_m и β_m М-кодов действительных чисел X_α , X_β соответственно. Для этого используется прекодер см. раздел 1.5.1.1
2. Декодирование действительных чисел X_α , X_β (положительных или отрицательных) чисел X_α , X_β из М-кода в Р-код. Для этого используется полный декодер М-кода в Р-код см. раздел 1.5.1.5.

3.3.3. Кодирование комплексного числа в системе 2.

1. Вычисление $\bar{X}_\beta = \mu \cdot X_\beta$ при $\mu = 1/\sqrt{2}$. Это вычисление выполняется в традиционной системе двоичного кодирования.
2. Кодирование действительных (положительных или отрицательных) чисел X_α , \bar{X}_β из Р-кода в М-код аналогично п.1 алгоритма 3.7.1.
3. Формирование С-кода комплексного числа $Z = X_\alpha + j\bar{X}_\beta$ аналогично п.2 алгоритма 3.7.1.

3.3.4. Декодирование комплексного числа в системе 2.

1. Выделение из С-кода комплексного числа $Z = X_\alpha + jX_\beta$ четных и нечетных разрядов по правилу $\left\{ \begin{array}{l} \alpha_{m/2} = \gamma_m \text{ if } m - \text{even} \\ \beta_{(m-1)/2} = \gamma_m \text{ if } m - \text{odd} \end{array} \right\}$ и формирование из разрядов α_m и β_m М-кодов действительных чисел X_α , \bar{X}_β соответственно, где $\bar{X}_\beta = \mu \cdot X_\beta$ при $\mu = 1/\sqrt{2}$. Для этого используется прекодер. см. раздел 1.5.1.1
2. Декодирование действительных (положительных или отрицательных) чисел X_α , \bar{X}_β из М-кода в Р-код аналогично п.2 алгоритма 3.7.2.
3. Вычисление $X_\beta = \bar{X}_\beta \sqrt{2}$. Это вычисление выполняется в традиционной системе двоичного кодирования.

Глава 4. Векторный процессор

4.1. Представление данных и векторное арифметическое устройство

В отличие от обычного представления данных, описанного в разделе 2.1, координаты точки представляются кодом точки-вектора. Простое векторное арифметическое устройство **VAU** оперирует с p -мерными векторами. Такое устройство должно содержать $p(n+r)$ -разрядный умножитель, $p(n+r)$ -разрядный сумматор, $p(n+r)$ -разрядный регистр координат, и a -разрядный регистр параметров. На нем аффинное преобразование каждой точки содержит только одну операцию. VAU представлено на рис. 4.1.1. Оно аналогично устройству SAU, но, в отличие от последнего, содержит сумматор векторов. Все параметры преобразования в нем подаются одновременно в координатный блок в кодах векторов преобразования – см. описание операций с кодами векторов. Кроме того, в нем предусмотрены блоки кодирования и декодирования векторов.

Рассмотрим перечень команд процессора, реализуемых в VAU:

- Прием и кодирование параметров преобразования
- Прием и кодирование всех координат точки
- Сложение с вектором переноса
- Умножение на матрицу преобразования
- Декодирование и выдача всех координат
- Округление

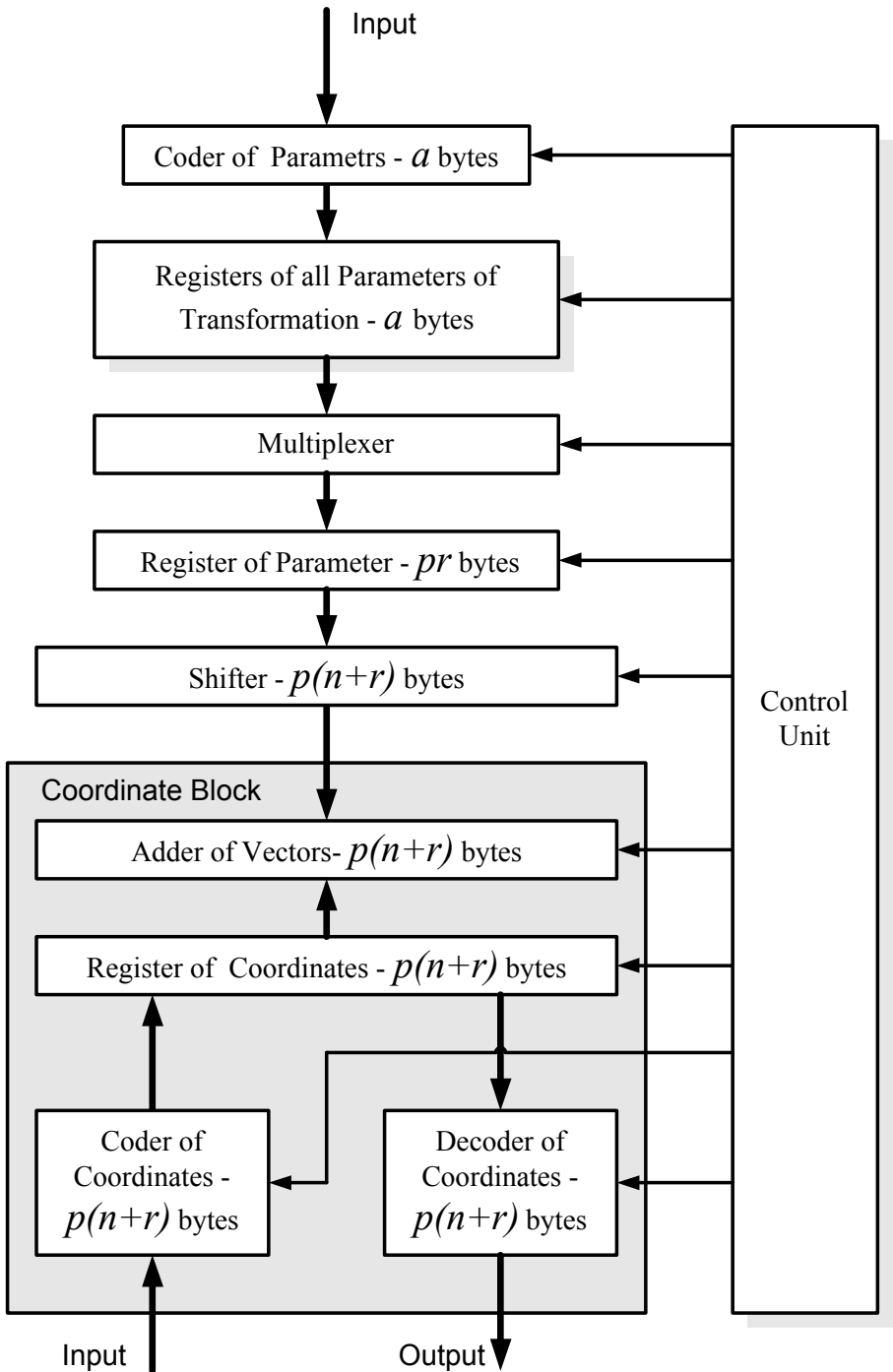


Рис. 4.1.1. Векторное арифметическое устройство

Можно предложить также (по аналогии с MSAU) арифметическое устройство **MVAU**, содержащее множество (M) элементарных устройств VAU, работающих параллельно. В этом устройстве коды всех точек-векторов фигуры образуют массив, который будем называть прямоугольным кодом векторов - **RCV**. RCV содержит M регистров разрядностью $p(n+r)$. MVAU в целом содержит M сумматоров разрядностью $p(n+r)$, M умножителей разрядностью $p(n+r)$, RCV и один a -разрядный регистр параметров. Это MVAU выполняет групповые операции - сложение RCV с вектором переноса и умножение RCV на матрицу преобразования.

Далее, по аналогии с FSAU, можно рассмотреть еще один вариант, занимающий промежуточное место между АУ с одиночными и групповыми операциями. Для этого разделим RCV на несколько фрагментов RCS_q каждый из которых содержит (Q) регистров разрядностью $p(n+r)$. Арифметическое устройство **FVAU** содержит в целом Q сумматоров разрядностью $p(n+r)$, Q умножителей разрядностью $p(n+r)$, RCS_q и a -разрядный регистр параметров. Схема FVAU представлена на рис. 4.1.2. Эта схема идентична схеме рис. 4.1.1, за исключением того, что в ней используется операционный блок, содержащий Q координатных блоков, выделенных на рис. 4.1.1.

FVAU выполняет групповые операции с координатами точек фрагмента фигуры. На нем аффинное преобразование фигуры содержит Q групповых умножений и Q групповых сложений.

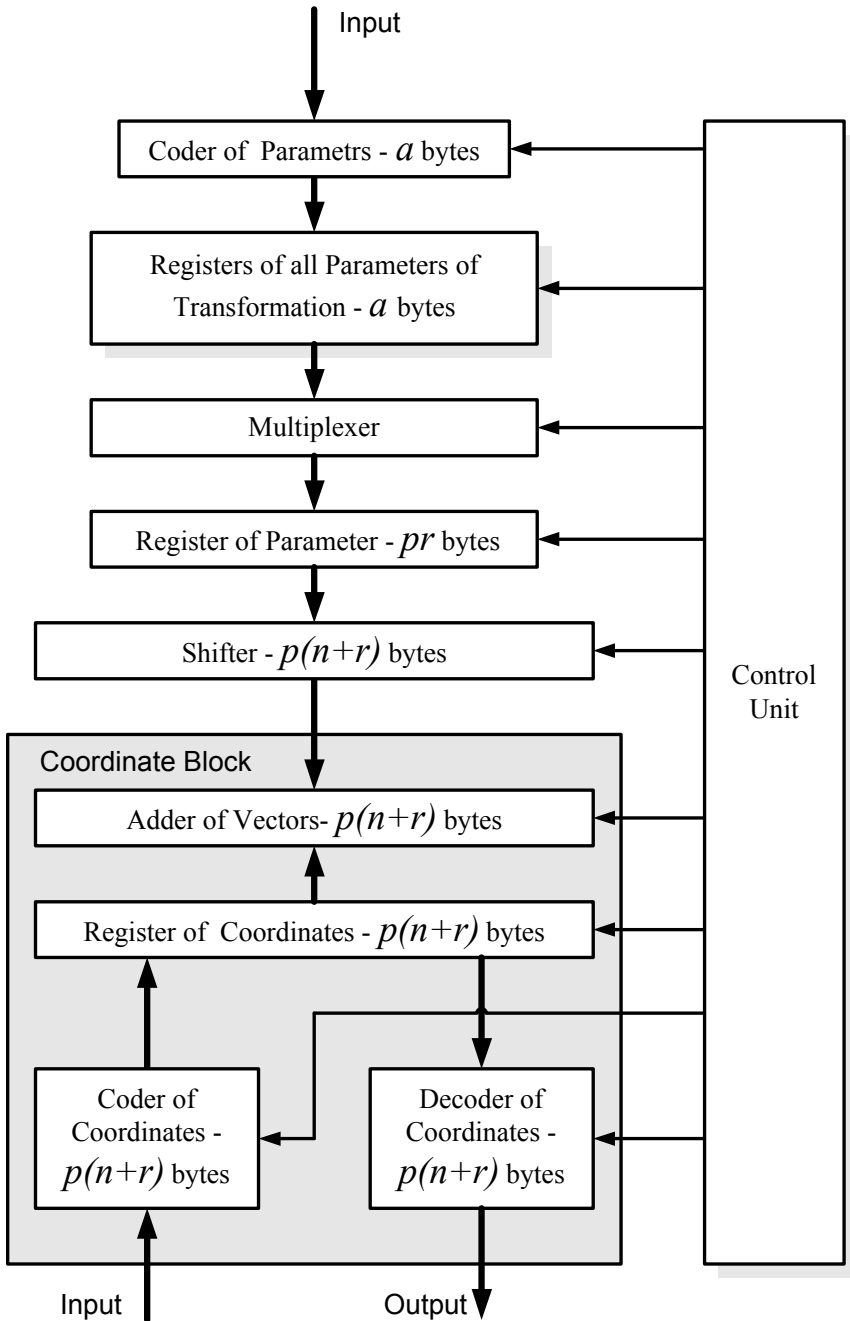


Рис. 4.1.2. Векторное арифметическое устройство с прямоугольными кодами.

4.2. Сравнения

В табл. 4.2.1а и 4.2.1б приведены сравнительные характеристики перечисленных устройств. В этой таблице

- R - разрядность всех регистров;
- U - количество умножителей;
- A - объем сумматоров, измеренный в разрядах регистра; предполагается, что объем сумматора в *три раза* превышает объем регистра;
- M - объем сдвигателя в умножителе, измеренный в разрядах регистра; предполагается, что объем сдвигателя в *два раза* превышает объем регистра;
- $W = (R + A + M)$ - объем АУ, измеренный в разрядах регистра;
- S - количество элементарных операций данного АУ для аффинного преобразования фигуры в целом.

Таблица 4.2.1а. Сравнительные характеристики АУ.

#	AU	U	R	A	M
1	SAU	1	$2(n+r)+a$	$2(n+r)$	$3(n+r)$
2	VAU	1	$2p(n+r)+a$	$2p(n+r)$	$3p(n+r)$
3	MSAU	M	$2M(n+r)+a$	$2M(n+r)$	$3(n+r)M$
4	MVAU	Mp^2	$2Mp(n+r)+a$	$2Mp(n+r)$	$3p(n+r)M$
5	FSAU	Q	$2Q(n+r)+a$	$2Q(n+r)$	$3(n+r)Q$
6	FVAU	Qp^2	$2Qp(n+r)+a$	$2Qp(n+r)$	$3p(n+r)Q$

Таблица 4.2.1б. Сравнительные характеристики АУ.

#	AU	U	W	S
1	SAU	1	$7(n+r)+a$	$M(D+p^2)$
2	VAU	1	$7p(n+r)+a$	M
3	MSAU	M	$7M(n+r)+a$	$D+p^2$
4	MVAU	Mp^2	$7Mp(n+r)+a$	1
5	FSAU	Q	$7Q(n+r)+a$	$(D+p^2)M/Q$
6	FVAU	Qp^2	$7Qp(n+r)+a$	M/Q

В табл. 4.2.2, 4.2.3, 4.2.4 приведены числовые сравнительные характеристики перечисленных устройств при различных значениях n , r , p , M , Q . Эта таблица построена на основе табл. 4.2.1. Кроме

того, в этой таблице указано качество АУ, измеренное как $H=W*S/M$, и величина

$$h_k = \frac{H_k}{H_{k+1}},$$

которая определяет относительное качество АУ, оперирующим с числами, по сравнению с АУ, оперирующим с векторами.

Таблица 4.2.2. Числовые характеристики АУ при $p=2$, $r=6$, $M=10^6$, $n=12$, $Q=256$, $a=72$.

АУ	R	U	W	S	H	h
1	126	1	198	$6*10^6$	1188	3.7
2	180	1	324	10^6	324	
3	$54*10^6$	10^6	$126*10^6$	6	756	3
4	$108*10^6$	10^6	$254*10^6$	1	254	
5	14000	256	32000	24000	768	3
6	28000	256	64000	4000	256	

Таблица 4.2.3. Числовые характеристики АУ при $p=3$, $r=6$, $M=10^6$, $n=12$, $Q=256$, $a=90$.

АУ	R	U	W	S	H	h
1	144	1	216	$15*10^6$	3240	6.9
2	252	1	468	10^6	468	
3	$54*10^6$	10^6	$126*10^6$	15	1890	5
4	$162*10^6$	10^6	$378*10^6$	1	378	
5	14000	256	32000	60000	1920	5
6	42000	256	96000	4000	384	

Таблица 4.2.4. Числовые характеристики АУ при $p=4$, $r=6$, $M=10^6$, $n=12$, $Q=256$, $a=240$.

АУ	R	U	W	S	H	h
1	292	1	366	$28*10^6$	10248	13.8
2	384	1	744	10^6	744	
3	$54*10^6$	10^6	$126*10^6$	28	3528	7
4	$216*10^6$	10^6	$504*10^6$	1	504	
5	14000	256	32000	112000	3584	7
6	56000	256	128000	4000	512	

Из приведенных таблиц следует, что качество АУ, оперирующих с векторами, превышает качество АУ, оперирующих с числами.

Относительное качество возрастает в $h = 3, 5, 7$ раз при $p = 2, 3, 4$ и при $Q \gg 1$. Относительное качество h возрастает при $Q \rightarrow 1$. Это означает, что при данном объеме АУ производительность VAU, MVAU, FVAU возрастает в h раз по сравнению с производительностью SAU, MSAU, FSAU. Это означает также, что при данной производительности АУ объем VAU, MVAU, FVAU уменьшается в h раз по сравнению с объемом SAU, MSAU, FSAU. Таким образом, для разработки геометрических процессоров целесообразно использовать арифметику векторов.

Для выбора оптимального значения Q можно минимизировать критерий $\lambda = kW + S$, где k – определенный весовой коэффициент. При этом для FVAU оптимальное значение

$$Q = \sqrt{\frac{M}{7ap(n+r)}}$$

В частности, при $a=0.05$, $M=10^6$, $r=6$, $n=12$, $p=(2, 3, 4)$ оптимальное значение $Q = (282, 230, 199)$.

Основное внимание далее уделяется геометрическим процессорам, основанным на арифметике геометрических кодов. При этом для сравнения используется рассмотренное выше устройство FVAU, основанные на арифметике векторов.

Глава 5. Теория кодирования фигур

5.1. Первичные геометрические коды

5.1.1. Структура данных

Рассмотрим бинарное дерево, изображенное на рис. 5.1.1, и присвоим каждой его вершине двухзначный номер (i, k) , где k - номер яруса, а i - номер вершины в k -ярусе. При этом будем считать, что нумерация ярусов идет справа налево, а нумерация вершин - сверху вниз.

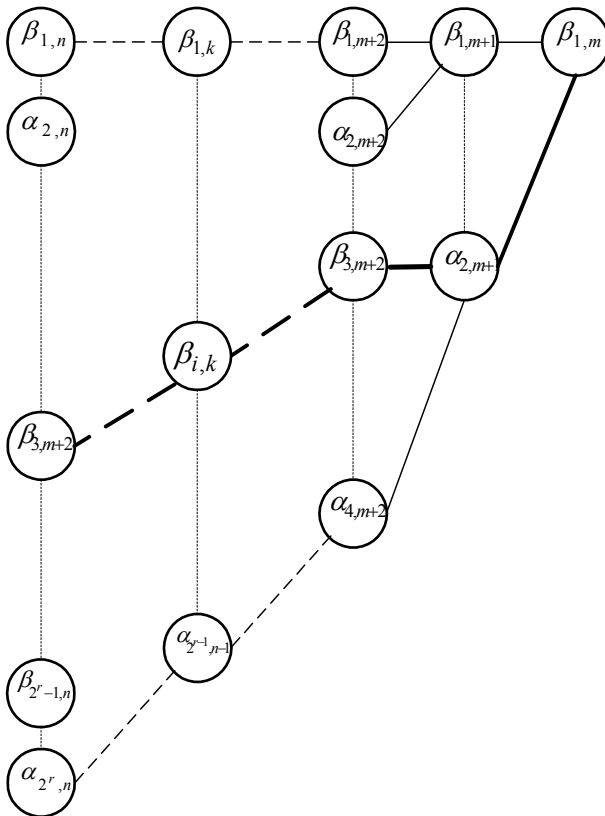


Рис. 5.1.1. Дерево геометрического кода.

Пусть m и n - номера крайнего правого и левого ярусов соответственно. Тогда $k=(n, n-1, \dots, m+1, m)$; $i=(1, 2, \dots, 2^{k-m})$; число ярусов $r=(n-m+1)$; число узлов дерева $u=(2^r-1)$; число вершин в n -ярусе $N=2^{r-1}$. Обозначим через $\alpha_{i,k}$ вершину с номером i - четным и через $\beta_{i,k}$ - вершину с номером i - нечетным.

Путь в дереве, соединяющий вершины $\beta_{1,m}$ и $\beta_{p,n}$, назовем **p-путем**. Очевидно, каждый p -путь можно изобразить последовательностью символов α и β . Например, на рис. 5.1.1 выделен p -путь, которому соответствует последовательность

$$\beta_{p,n} \cdots \beta_{i,k} \cdots \beta_{3,m+2} \alpha_{2,m+1} \beta_{1,m}.$$

Каждый символ $\alpha_{i,k}$ или $\beta_{i,k}$ последовательности, изображающей некоторый p -путь на дереве, назовем **k-разрядом p-пути** или **(i, k)-разрядом дерева**. Если каждому разряду p -пути поставить в соответствие 1 для α -разряда или 0 для β -разряда, то p -путь может быть изображен двоичным кодом $K[p]$. В частности, для рис. 5.1.1 $K[p] = 0\dots 0\dots 010$. Номер p -пути равен номеру того разряда в n -ярусе, которым заканчивается этот путь. Условимся теперь, что α и β - двоичные величины, то-есть $\alpha = (0,1)$ и $\beta = (0,1)$. Назовем p -путь **открытым**, если величина всех его разрядов равна 1, и - **закрытым**, если величина хотя бы одного его разряда равна 0.

На рис. 5.1.2 для иллюстрации изображено дерево двоичных разрядов, в котором открыты пути (в скобках указан двоичный код, соответствующий данному пути)

$$\alpha_{43} \alpha_{22} \beta_{11} \beta_{10} \quad (K[4]=1100),$$

$$\beta_{53} \beta_{32} \alpha_{21} \beta_{10} \quad (K[5]=0010),$$

$$\alpha_{63} \beta_{32} \alpha_{21} \beta_{10} \quad (K[6]=1010),$$

$$\beta_{73} \alpha_{42} \alpha_{21} \beta_{10} \quad (K[7]=0110),$$

то-есть это дерево представляет 4 двоичных кода. Следует обратить внимание на то, что открытому пути, изображаемому в дереве только единичными разрядами, соответствует двоичный код, содержащий в общем случае и нулевые разряды.

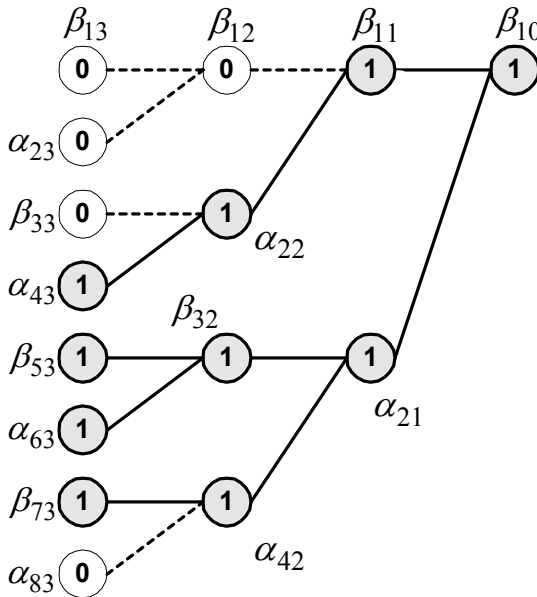


Рис. 5.1.2. Пример. Дерево бинарных разрядов

Построенное так бинарное дерево двоичных разрядов, изображающее множество двоичных кодов, назовем первичным геометрическим кодом **PGC** (в данном разделе прилагательное «первичный» будет опускаться и будет говориться о геометрическом коде **GC**), а составляющие его двоичные коды - линейными кодами. Номер разряда в старшем ярусе геометрического кода назовем адресом соответствующего линейного кода. Сокращение числа двоичных разрядов при изображении a двоичных кодов в виде геометрического кода $g = ra / (2^{r+1} - 2)$.

В частности, если все пути дерева открыты, то оно изображает все r -разрядные двоичные коды. Из приведенной формулы следует, что экономичность геометрического кода возрастает пропорционально величине a . Однако достоинства геометрического кода состоят, главным образом, в том, что с ним довольно просто выполняются различные операции. Поэтому геометрический код имеет смысл применять тогда, когда имеется достаточно большая группа двоичных кодов, с которыми необходимо выполнять одинаковые - групповые операции, например, умножать все коды на одно и то же число. Кроме того, геометрическим кодом удастся (как будет показано ниже) изображать произвольные фигуры и трактовать различные преобразования этих фигур как операции с геометрическим кодом.

5.1.2. Арифметические операции с геометрическими кодами по действительному основанию

5.1.2.1. Общие положения

Операции с геометрическими кодами, которые рассмотрены ниже, как правило, эквивалентны некоторой логической или арифметической операции между известным - **базисным** двоичным кодом и каждым из линейных кодов, входящих в множество, представленное геометрическим кодом. Кроме того, эти операции связаны с распространением **переносов** из правых - младших ярусов в левые - старшие ярусы дерева. Обозначим

- $\beta_{i,k}$ - (i, k) - разряд геометрического кода при i - нечетном;
- $\alpha_{i,k}$ - (i, k) - разряд геометрического кода при i - четном;
- $\pi_{i,k}$ - общий перенос в разряды $\beta_{2i-1,k+1}$ и $\alpha_{2i,k+1}$ (i - нечетное);
- $\mu_{i,k}$ - перенос π из разряда $\beta_{i,k}$;
- $\eta_{i,k}$ - перенос π из разряда $\alpha_{i,k}$;
- δ_k - k - разряд базисного кода;
- $\tau_{i,k}$ - сигнал транспонирования кода, у которого угловым является (i, k) - разряд.

Перенос $\eta_{i,k}$ из разряда $\alpha_{i,k}$ или перенос $\mu_{i,k}$ из разряда $\beta_{i,k}$ поступают в разряды $\beta_{2i-1,k+1}$ и $\alpha_{2i,k+1}$ в качестве переноса $\pi_{i,k}$ по схеме, представленной на рис. 5.1.2а.

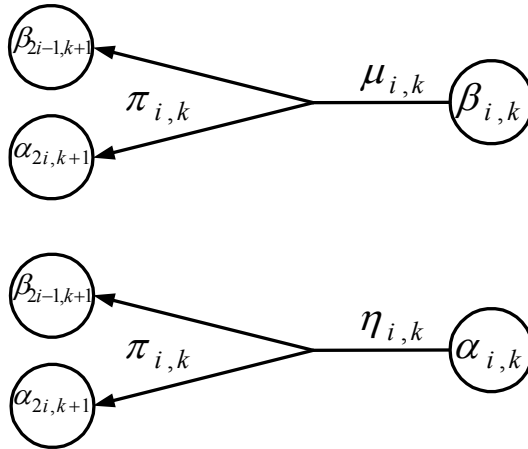


Рис. 5.1.2а. Схема распространения переносов в GC

Сигнал транспонирования $\tau_{i,k}$ **предшествует** сигналам $\mu_{i,k}$ и $\eta_{i,k}$, которые являются логическими функциями значений разрядов $\beta_{2i-1,k+1}$ и $\alpha_{2i,k+1}$, полученных **после** транспонирования.

Базисный код и линейные коды, представленные геометрическим кодом, могут рассматриваться как двоичные коды по основанию p некоторого числа или вектора. При этом всем разрядам геометрического кода, входящим в k -ярус, должен быть присвоен вес k -разряда линейного кода.

Число линейных кодов в составе геометрического кода при арифметических операциях не изменяется.

5.1.2.2. Запись базисного кода.

Процесс распространения переноса при формировании в GC пути, имеющего линейный код, равный базисному коду δ определяется следующими формулами:

$$\mu = \pi \wedge \overline{\delta}, \quad \eta = \pi \wedge \delta.$$

При $\mu=1$ разряд β принимает значение “1” вне зависимости от его прежнего значения. Аналогично, при $\eta=1$ разряд α принимает значение “1”.

5.1.2.3. Транспонирование

Транспонированием геометрического кода будем называть такое преобразование, при котором нижняя и верхняя половины геометрического кода меняются местами. Точнее говоря, разряды исходного кода связаны с разрядами транспонированного кода (помеченные верхней чертой) следующим образом:

$$\alpha_{i,k} = \bar{\alpha}_{j,k}, \quad \beta_{i,k} = \bar{\beta}_{j,k}, \quad j = \text{rest}(1 + 2^{k-i}) \bmod 2^{k-i+1}.$$

Например, код на рис. 5.1.2 транспонируется в код на рис. 5.1.3.

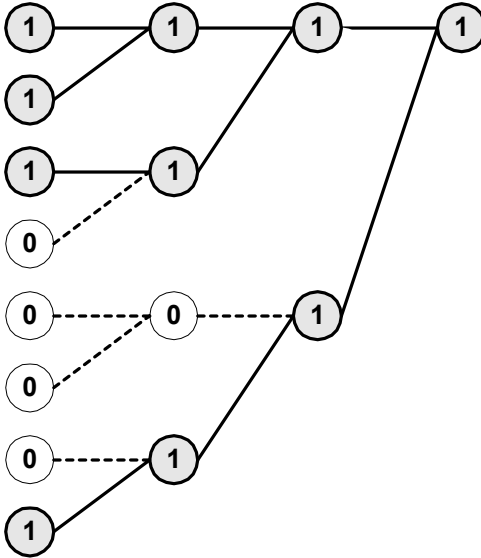


Рис. 5.1.3. Пример. Транспонированный код.

5.1.2.4. Сложение геометрического и базисного

кодов при $p=2$ описывается табл. 5.1.1а, откуда следует, что

$$\tau = \delta \oplus \pi, \quad \eta = \alpha \wedge \delta \wedge \pi, \quad \mu = (\delta \vee \pi) \wedge \beta,$$

Расставляя индексы, получаем следующие формулы:

$$\tau_{i,k} = \delta_{k+1} \oplus \pi_{i,k}, \quad (5.1.1)$$

$$\eta_{i,k} = \alpha_{i,k} \wedge \delta_k \wedge \pi_{i-1,k-1} \quad \text{при } i - \text{четном}, \quad (5.1.2)$$

$$\mu_{i,k} = (\delta_k \vee \pi_{i,k-1}) \wedge \beta_{i,k} \quad \text{при } i - \text{нечетном}. \quad (5.1.3)$$

Таблица 5.1.1а. Сложение геометрического и базисного кодов при $\rho=2$.

α	β	δ	π	τ	η	μ
0	0	0	0	0	0	0
0	1	0	0	0	0	0
1	0	0	0	0	0	0
1	1	0	0	0	0	0
0	0	0	1	1	0	0
0	1	0	1	1	0	1
1	0	0	1	1	0	0
1	1	0	1	1	0	1
0	0	1	0	1	0	0
0	1	1	0	1	0	1
1	0	1	0	1	0	0
1	1	1	0	1	0	1
0	0	1	1	0	0	0
0	1	1	1	0	0	1
1	0	1	1	0	1	0
1	1	1	1	0	1	1

Пример 5.1.1 сложения при $\rho=2$. Пусть базисный код $\mathbf{K}=\langle 2 \rangle$ или $\mathbf{K}=10$, а геометрический код \mathbf{G} изображает множество линейных кодов $\{ 1100, 0010, 1010, 0110 \}$ или, что одно и то же, множество чисел $\{ 12, 2, 10, 6 \}$. Найдем геометрический код $\mathbf{R} = \mathbf{G} + \mathbf{K}$ – см. рис. 5.1.4. Процесс распространения переносов прекращается. Полученный код $\mathbf{R} = \mathbf{G}_4$ изображает множество кодов $\{ 1110, 0010, 1100, 1000 \}$, то есть множество чисел $\{ 14, 4, 12, 8 \}$, что и требовалось получить. Таким образом, сложение геометрического и базисного кодов при $r = 2$ сводится к многократному транспонированию.

	$m =$	3 2 1 0	номер разряда
	$K =$	0 0 1 0	базисный код
1)	$G_1 =$	0 0 1 1	$\pi_{10} = 0$
		0	
		0 1	$\tau_{10} = \delta_1 = 1$
		1	
		1 1 1	
		1	
		1 1	
		0	
2)	$G_2 =$	1 1 1 1	$\pi_{11} = (\delta_1 \vee \pi_{10}) \wedge \beta_{11} = 1$
		1	
		1 1	$\pi_{21} = \alpha_{21} \wedge \delta_1 \wedge \pi_{10} = 0$
		0	
		0 0 1	$\tau_{11} = \delta_2 \oplus \pi_{11} = 1$
		0	
		0 1	$\tau_{21} = \delta_2 \oplus \pi_{21} = 0$
		1	
3)	$G_3 =$	1 1 1 1	$\pi_{12} = 1$
		0	
		1 1	$\pi_{22} = \pi_{32} = \pi_{42} = 0$
		1	
		0 0 1	$\tau_{12} = 1$
		0	
		0 1	$\tau_{22} = \tau_{32} = \tau_{42} = 0$
		1	
4)	$G_4 =$	0 1 1 1	$\pi_{i,3} = 0$
		1	
		1 1	$\tau_{i,3} = 0$
		1	
		0 0 1	
		0	
		0 1	
		1	

Рис. 5.1.4. К примеру 5.1.1.

5.1.2.5. Алгебраическое сложение геометрического и базисного кодов при $\rho=2$ возможно только в том случае, если исходные числа представлены в виде дополнительных кодов. В этом случае алгебраическое сложение описывается теми же уравнениями. Применение обратных кодов невозможно, так как в геометрическом коде не удастся организовать цепи циклического переноса.

5.1.2.6. Алгебраическое сложение геометрического и базисного кодов при $\rho=-2$ состоит из последовательно выполняемых операций инвертирования (умножения на ‘-1’) и обратного сложения (вычисления по формуле $c=-a-b$). Операция обратного сложения описывается табл. 5.1.1b, из которой следует, что

$$\tau = \delta \oplus \pi, \quad \eta = \alpha \wedge (\delta \vee \bar{\pi}), \quad \mu = \beta \wedge \delta \wedge \bar{\pi}.$$

Таблица 5.1.1b. Обратное сложение GC с базисным кодом при $\rho=-2$.

α	β	δ	π	τ	η	μ
0	0	0	0	0	0	0
0	1	0	0	0	0	1
1	0	0	0	0	0	0
1	1	0	0	0	0	1
0	0	0	1	1	0	0
0	1	0	1	1	0	0
1	0	0	1	1	0	0
1	1	0	1	1	0	0
0	0	1	0	1	0	0
0	1	1	0	1	0	1
1	0	1	0	1	1	0
1	1	1	0	1	1	1
0	0	1	1	0	0	0
0	1	1	1	0	0	1
1	0	1	1	0	0	0
1	1	1	1	0	0	1

Эти же формулы при $\delta=0$ описывают инвертирование геометрического кода. Здесь также можно воспользоваться формулами (5.1.1) и

$$\eta_{i,k} = \beta_{i,k} \wedge \delta_k \wedge \bar{\pi}_{i-1,k-1} \quad \text{при } i - \text{ четном,} \quad (5.1.4)$$

$$\mu_{i,k} = (\delta_k \vee \bar{\pi}_{i,k-1}) \wedge \alpha_{i,k} \quad \text{при } i - \text{ нечетном.} \quad (5.1.5)$$

Пример 5.1.2 обратного сложения при $\rho = -2$. Пусть базисный код $\mathbf{K} = \langle 2 \rangle$ или $\mathbf{K} = 110$, а геометрический код \mathbf{G} изображает множество линейных кодов $\{0000, 0100, 0010, 0110\}$ или, что одно и то же, множество чисел $\{0, 4, -2, 2\}$. Найдем геометрический код $\mathbf{R} = -\mathbf{G} - \mathbf{K}$ - см. рис. 5.1.5.

Процесс распространения переносов прекращается. Полученный код $\mathbf{R} = \mathbf{G}_4$ изображает множество кодов $\{0000, 1100, 0010, 1110\}$, то есть множество чисел $\{0, -4, -2, -6\}$, что и требовалось получить. Таким образом, сложение геометрического и базисного кодов при $\rho = -2$ сводится к многократному транспонированию.

$m =$	3 2 1 0	номер разряда
$K =$	0 0 1 0	базисный код
1)	$G_1 =$	$\pi_{10} = 0$
	1 1 1 1	
	0	
	1 1	$\tau_{10} = \delta_1 = 1$
	0	
	1 1 1	
	0	
	1 1	
	0	
2)	$G_2 =$	$\pi_{11} = \beta_{11} \wedge \delta_1 \wedge \bar{\pi}_{10} = 1$
	1 1 1 1	
	0	
	1 1	$\pi_{21} = \alpha_{21} \wedge (\delta_1 \vee \bar{\pi}_{10}) = 1$
	0	
	1 1 1	$\tau_{11} = \delta_2 \oplus \pi_{11} = 0$
	0	
	1 1	$\tau_{21} = \delta_2 \oplus \pi_{21} = 0$
	0	
3)	$G_3 =$	$\pi_{12} = \pi_{32} = 0$
	1 1 1 1	
	0	
	1 1	$\pi_{22} = \pi_{42} = 1$
	0	
	1 1 1	$\tau_{12} = \tau_{32} = 0$
	0	
	1 1	$\tau_{22} = \tau_{42} = 1$
	0	
4)	$G_4 =$	$\pi_{i,3} = 0$
	1 1 1 1	
	0	
	0 1	$\tau_{i,3} = 0$
	1	
	1 1 1	
	0	
	0 1	
	1	

Рис. 5.1.5. К примеру 5.1.2.

5.1.2.7. Умножение геометрического и базисного кодов

При описании этой операции мы ограничимся случаем, когда базисный код является целым, поскольку другой случай легко сводится к этому сдвигом произведения. Итак, предположим, что базисный код является множимым, а геометрический - множителем. Смысл умножения заключается в том, чтобы заменить все разряды $\alpha_{i,k} = 1$ множителя базисным кодом. Для такой замены необходимо

- выделить в геометрическом коде G геометрический код $G_{i,k}$, в младшем разряде которого находится разряд $\alpha_{i,k} = 1$, и код остатка G_0 ;
- сложить код $G_{i,k}$ с базисным кодом, полагая, что вершина кода $G_{i,k}$ лежит в нулевом ярусе - в результате этой операции образуется некоторый код $G'_{i/2,k-1}$;
- наложить код $G'_{i/2,k-1}$, полученный в предыдущем пункте, на код остатка G_0 .

Умножение в целом состоит в последовательной замене разрядов $\alpha_{i,k} = 1$ множителя, которая начинается с разрядов старшего яруса и распространяется слева направо. При этом переносы при сложении распространяются в противоположную сторону и не искажают тех разрядов множителя, которые еще не претерпели процесса замены. Заметим, что код $G_{i,k}$ состоит из (r, s) -разрядов кода, где $s > k$, $i2^{r-k} \geq r \geq i(2^{r-k-1} + 1)$. Например, если $\alpha_{i,k} = \alpha_{21}$, то

$$G_{i,k} = G_{21} = \dots \begin{matrix} \beta & & \beta & & 0 \\ & 53 & & 32 & \\ \alpha & & & & \\ & & 63 & & \\ \beta & & & \alpha & \\ & 73 & & & 42 \\ \alpha & & & & \\ & & 83 & & \end{matrix}$$

Для ликвидации переполнения разрядной сетки, которое может возникнуть при умножении, следует воспользоваться операцией округления, описанной ниже.

Данный способ умножения не применим при $p=2$, если среди чисел, представленных линейными кодами, имеются отрицательные числа.

Пример 5.1.3 умножения при $p=-2$. Найдем произведение базисного кода $K = \langle -2 \rangle$ или $K = 10$ и геометрического кода — см. рис. 5.1.6.

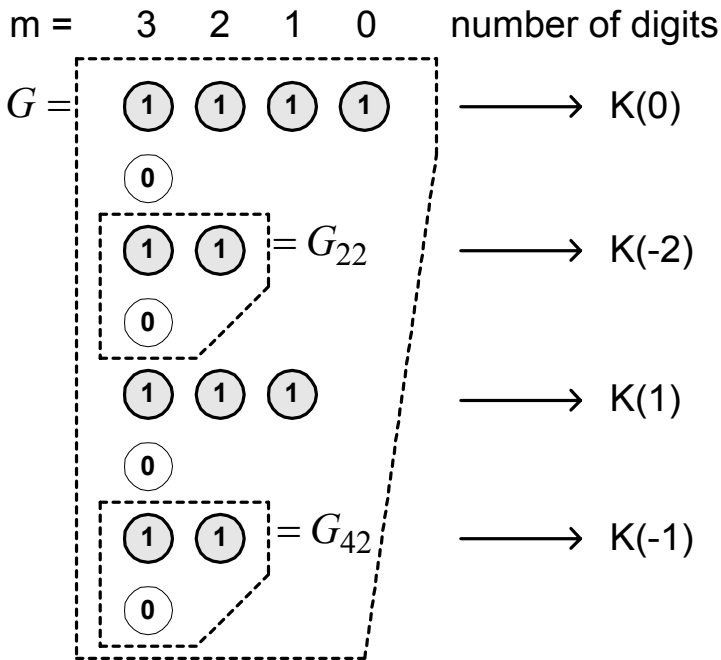


Рис. 5.1.6. К примеру 5.1.3.

В третьем ярусе кода G нет разрядов $\alpha = 1$. Поэтому переходим к анализу второго яруса, где $\alpha_{22} = \alpha_{42} = 1$. Выделяем коды G_{22} , G_{42} , G'_0 . Производим сложение кодов G_{22} , G_{42} и 10 , в результате чего получаем коды G'_{11} и G'_{21} . Налагаем затем эти коды на код G'_0 и получаем код G' — см. рис. 5.1.7.

$G'_0 =$	$G'_{21} =$	$G'_{11} =$	$G' =$
1 1 1 1	0 1 1	0 1 1	1 1 1 1
0	1	1	1
0 0	0 0	0 0	0 0
0	0	0	0
1 1 1			1 1 1
0			1
0 0			0 0
0			0

Рис. 5.1.7. К примеру 5.1.3.

Рассматриваем первый ярус полученного кода G' и выделяем из него коды – см. рис. 5.1.8.

$G''_{21} =$	$G''_0 =$
1 1 1	1 1 1 1
1	1
0 0	0 0
0	0
	0 0 0
	0
	0 0
	0

Рис. 5.1.8. К примеру 5.1.3.

Производим сложение кода G''_{21} с кодом 10 и в результате получаем код G''_{11} . Налагаем затем код G''_{11} на код G''_0 и получаем окончательно код G'' – см. рис. 5.1.9.

$G''_{11} =$	$G'' =$	$\rightarrow K(\dots)$
0 0 1 1	1 1 1 1	$\rightarrow K(0)$
0	1	$\rightarrow K(4)$
1 1	1 1	$\rightarrow K(-2)$
1	1	$\rightarrow K(2)$
0 0 0	0 0 0	
0	0	
0 0	0 0	
0	0	

Рис. 5.1.9. К примеру 5.1.3.

Таким образом, $G'' = -2G$. Правильность умножения легко проверить. Действительно, код G изображает множество чисел $\{-2, -1, 0, 1\}$, а код G'' - множество чисел $\{-2, 0, 2, 4\}$, получаемое из первого умножением на '-2'.

Рассмотрим частный случай, когда базисный код содержит «1» в младшем разряде. При этом алгоритм умножения упрощается и состоит в следующем:

- выделить в геометрическом коде G геометрический код $G_{i,k}$, в младшем разряде которого находится разряд $\alpha_{i,k} = 1$;
- сложить код $G_{i,k}$ с базисным кодом, у которого младший разряд обнулен, полагая, что вершина кода $G_{i,k}$ лежит в нулевом ярусе – в результате этой операции образуется некоторый код $G'_{i,k}$;
- наложить код $G'_{i,k}$, полученный в предыдущем пункте, на код G .

Такой алгоритм эквивалентен тому, что все коды $\{G_{i,k}, i - \text{var}\}$ k -яруса складываются с базисным кодом, у которого младший разряд обнулен. При этом переносы исходят из разрядов $\alpha_{i,k}$, а сам этот разряд не меняет своего значения, т.к. складывается с нулевым разрядом базисного кода.

Пример 5.1.3а умножения при $\rho=-2$. Найдем произведение базисного кода $\mathbf{K}=\langle -1 \rangle$ или $\mathbf{K}=11$ и геометрического кода – см. рис. 5.1.9а.

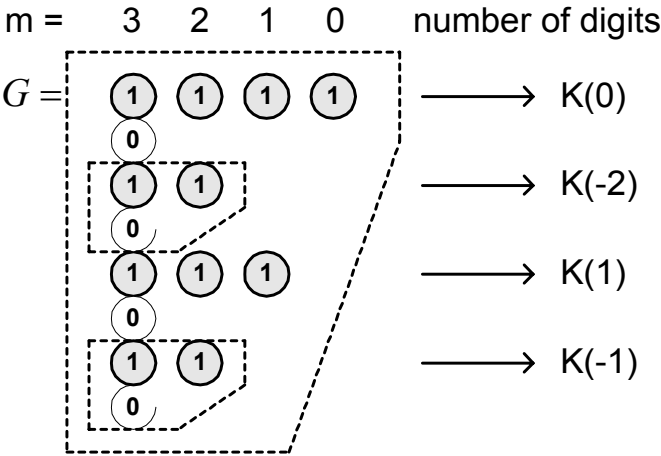


Рис. 5.1.9а. К примеру 5.1.3а.

В третьем ярусе G нет разрядов $\alpha=1$. Поэтому переходим к анализу второго яруса, где $\alpha_{22} = \alpha_{42} = 1$. После сложения кода G с измененным базисным кодом 10 получаем код G' – см. рис. 5.1.9б.

$G' =$	$\rightarrow K(\dots)$
1 1 1 1	$\rightarrow K(0)$
0	
0 1	
1	$\rightarrow K(2)$
1 1 1	$\rightarrow K(1)$
0	
0 1	
1	$\rightarrow K(3)$

$G'' =$	$\rightarrow K(\dots)$
1 1 1 1	$\rightarrow K(0)$
0	
0 1	
1	$\rightarrow K(2)$
1 1 1	$\rightarrow K(1)$
0	
1 1	$\rightarrow K(-1)$
0	

Рис. 5.1.9б. К примеру 5.1.3а.

В первом ярусе этого кода $\alpha_{12} = 1$. После сложения кода G' с измененным базисным кодом 10 получаем код G'' – см. рис. 5.1.9б. Таким образом, $G'' = -G$. Правильность умножения легко проверить. Действительно, код G изображает множество чисел $\{-2, -1, 0, 1\}$, а код G'' – множество чисел $\{-1, 0, 1, 2\}$, получаемое из первого умножением на -1 .

5.1.2.8. Деление геометрического на базисный код некоторого числа заменяется умножением геометрического кода на базисный код обратного числа.

5.1.2.9. Округление геометрического кода, содержащего r ярусов, состоит в отбрасывании младшего яруса. В результате образуются два кода, содержащих по $(r-1)$ ярусов. Операция заканчивается наложением полученных кодов. Таким образом, в результате наложения остается меньшее (точнее, не большее) число линейных кодов меньшей разрядности. Это связано с тем, что при отбрасывании младших разрядов могут образоваться равные линейные коды, которые в результирующем геометрическом коде фиксируются как один код.

5.1.3. Геометрические коды по комплексному основанию.

В качестве основания кодирования линейных кодов могут использоваться комплексные числа. Аналогично этому могут быть построены атрибутные геометрические коды по комплексному основанию. В отличие от предыдущего в таких кодах значением пути является линейный двоичный код по комплексному основанию. Такие коды описаны в разделе 3.1 – см. табл. 3.1.1, где перечислены существующие системы кодирования комплексных чисел. Ниже будут рассмотрены арифметические операции с геометрическими кодами в системах кодирования 1, 2 и 3. Поэтому в дальнейшем изложении кодов можно воспользоваться результатами предыдущего раздела. Некоторые операции вовсе не зависят от основания кодирования и они здесь не рассматриваются.

5.1.3.1. Алгебраическое сложение геометрического и базисного кодов.

В указанных системах двоичный код комплексного числа может рассматриваться (при выполнении алгебраического сложения) как два кода частей Im и Re по основанию $\rho = -2$, разряды которых чередуются. В связи с этим операции обратного сложения при описываются такими же, как при $\rho = -2$, уравнениями, но переносы $\mu_{i,k}$ и $\eta_{i,k}$ из k -яруса поступают не в два разряда $(k+1)$ -яруса, а в четыре разряда $(k+2)$ -яруса. Формулы сложения в этом случае принимают следующий вид:

$$\tau_{i,k} = \delta_{k+1} \oplus \bar{\pi}_{(i+1)/2,k-1} \quad \text{при } i - \text{четном}, \quad (5.1.6)$$

$$\tau_{i,k} = \delta_{k+1} \oplus \bar{\pi}_{i/2,k-1} \quad \text{при } i - \text{нечетном}, \quad (5.1.7)$$

$$\eta_{i,k} = \beta_{i,k} \wedge \delta_k \wedge \bar{\pi}_{j,k-2} \quad \text{при } i - \text{четном}, \quad (5.1.8)$$

$$\mu_{i,k} = (\delta_k \vee \bar{\pi}_{j,k-2}) \wedge \alpha_{i,k} \quad \text{при } i - \text{нечетном}, \quad (5.1.9)$$

где $j=1+z[(i-1)/4]$, а функция $z[x]$ - целая часть от аргумента x .

Пример 5.1.4 обратного сложения при $\rho = j\sqrt{2}$. Пусть базисный код $K = \langle j\sqrt{2} \rangle = 10$, а коды чисел 0 и $j\sqrt{2}$ изображены геометрическим кодом G . Найдем геометрический код $R = -G - K$ – см. рис. 5.1.10.

m =	3 2 1 0	номер разряда
K =	0 0 1 0	базисный код
1) $G = G_1 =$	1 1 1 1	$\pi_{10} = 0$
	0	
	0 0	$\tau_{10} = \delta_1 = 1$
	0	
	1 1 1	
	0	
	0 0	
	0	

2)	$G_2 =$	1 1 1 1 0 0 0 0 1 1 1 0 0 0 0	$\pi_{11} = \beta_{11} \wedge \delta_{11} = 1$ $\pi_{21} = \alpha_{21} \wedge \delta_{11} = 1$ $\tau_{11} = \delta_{21} = 0$ $\tau_{21} = \delta_{21} = 0$
3)	$G_3 =$	1 1 1 1 0 0 0 0 1 1 1 0 0 0 0	$\pi_{12} = \pi_{22} = \pi_{32} = \pi_{42} = 0$ $\tau_{12} = \tau_{22} = \tau_{32} = \tau_{42} = 1$
4)	$G_4 =$	0 1 1 1 1 0 0 0 0 1 1 1 0 0 0	$\pi_{i,3} = 0$ $\tau_{i,3} = 0$

Рис. 5.1.10. К примеру 5.1.4.

Процесс распространения переносов прекращается. Полученный код $\mathbf{R} = G_4$ изображает коды 1010 и 1000 чисел $(-j\sqrt{2})$ и $(-2j\sqrt{2})$ соответственно. Таким образом, и при $\rho = j\sqrt{2}$ процесс сложения сводится к многократному транспонированию.

Итак, умножение GC на базисный код состоит из сложений фрагментов GC с базисным кодом. При основании (-2) такое сложение состоит из обратного сложения и инвертирования полученного фрагмента.

5.1.3.2. Умножение геометрического и базисного кодов производится так, как описано выше для произвольного

основания. Но, кроме того, в этом случае возможны еще некоторые модификации данной операции:

- умножение действительной части геометрического кода (ярусы с четным номером) на базисный код, заключающееся в замене только тех разрядов $\alpha_{i,k} = 1$, которые принадлежат действительным частям линейных кодов;
- умножение мнимой части геометрического кода (ярусы с нечетным номером) на базисный код, выполняемое аналогично предыдущему;
- умножение действительной и мнимой части геометрического кода одновременно на различные базисные коды.

Еще одно отличие относится только к системе кодирования 1 и состоит в следующем. В ярусах с четным номером разряды $\alpha_{i,k} = 1$ заменяются на линейные коды комплексного числа Z так, как описано выше для общего случая. В ярусах с нечетным номером разряды $\alpha_{i,k} = 1$ заменяются на линейные коды комплексного числа jZ так, как описано выше для общего случая.

Пример 5.1.5 умножения мнимой части геометрического кода при $\rho = j\sqrt{2}$. Известен базисный код $\mathbf{K} = \langle 1 + j\sqrt{2} \rangle$ или $\mathbf{K} = 11$ и геометрический код \mathbf{G} - см. рис. 5.1.11.

m = 3 2 1 0 -1	m = 3 2 1 0 -1
$\mathbf{G} =$ 1 1 1 1 1	$\mathbf{G}'_0 =$ 1 1 1 1 1
0	0
0 0	0 0
0	0
1 1 1	1 1 1
0	0
0 1	0 1
1	0
0 0 1 1	0 0 1 1
0	0
1 1	1 1
0	0
1 1 1	1 1 1
0	0
0 1	0 1
1	0

Рис. 5.1.11. К примеру 5.1.5.

Найдем геометрический код $R=K(JmG)$. Вначале анализируем старший нечетный (третий) ярус кода G и обнаруживаем, что $\alpha_{83} = \alpha_{163} = 1$. Выделяем коды $G_{83} = G_{163} = 1$ и G'_0 . Затем производим сложение кодов G_{83} и K , в результате чего получаем коды

$$G'_{73} = G'_{153} \begin{pmatrix} 0 & 0 & 1 \\ 0 \\ 0 & 1 \\ 1 \end{pmatrix}$$

Налагаем коды G'_0 , G'_{73} , G'_{153} и получаем код G' – см. рис.

5.1.12. Пропуская второй ярус (поскольку производится умножение только мнимой части), анализируем первый ярус кода G' и замечаем, что $\alpha_{21} = \alpha_{41} = 1$. Выделяем коды G'_{21} , G'_{41} , G''_0 . Затем производим сложение кодов G'_{21} и G'_{41} с кодом K и в результате получаем коды G''_{10} и G''_{20} . Налагаем, далее, эти коды на код G''_0 и получаем окончательно код R – см. рис.

5.1.13. Далее будет дана геометрическая интерпретация этого примера.

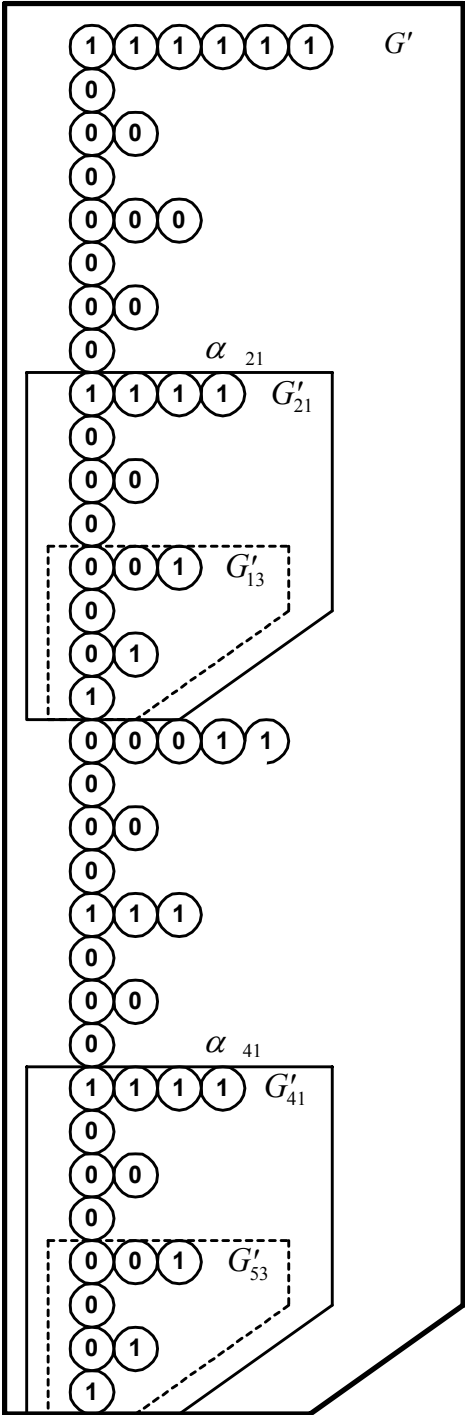


Рис. 5.1.12. К примеру 5.1.5.

$G''_0 =$	$G''_{10} =$	$G''_{20} =$	$R =$
1 1 1 1 1	0 0 0 1	0 0 0 1	1 1 1 1 1
0	0	0	0
0 0	0 0	0 0	0 0
0	0	0	0
0 0 0	0 1 1	0 1 1	0 1 1
0	1	1	1
0 0	1 1	1 1	1 1
0	0	0	0
0 0 1 1			0 0 1 1
0			0
1 1			1 1
0			0
0 0 0			0 1 1
0			1
0 0			1 1
0			0

Рис. 5.1.13. К примеру 5.1.5.

5.1.4. Кодирование и преобразование плоских фигур

5.1.4.1. Метод кодирования

При кодировании плоских фигур будем полагать, что

- на плоскости выделено N точек, распределенных равномерно с шагом Δx по x оси и Δy по оси y ;
- каждой точке может приписываться одно из двух значений - 0 или 1;
- фигура определяется подмножеством a точек, которым приписано единичное значение.

Тривиальный способ кодирования фигуры мог бы заключаться в задании пар координат x и y всех точек или кодов комплексных чисел $x+jy$, соответствующих этим точкам. Тогда различные преобразования фигур заключались бы в вычислениях с комплексными числами по некоторой программе. Однако множество a двоичных кодов комплексных чисел можно представить геометрическим кодом. Такое кодирование, во-первых,

требует меньше памяти, а, во-вторых, геометрические преобразования фигур легко интерпретируются как операции с геометрическими кодами.

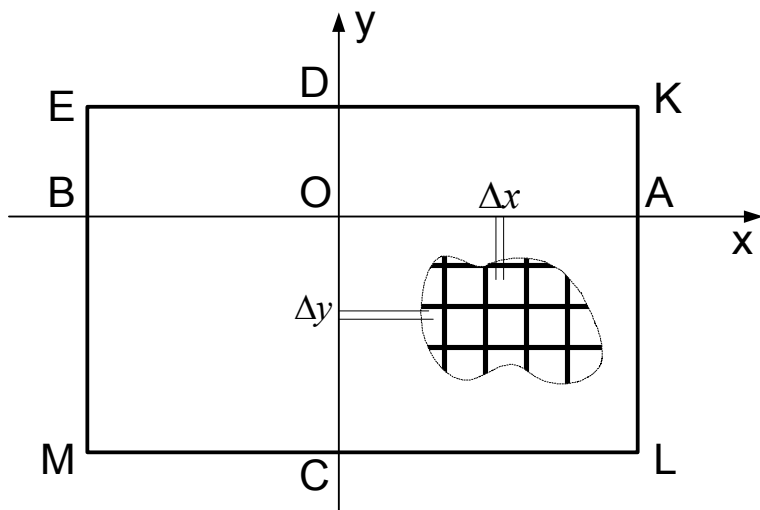


Рис. 5.1.14. Кодирование плоской фигуры.

Кодируемый геометрическим кодом участок плоскости имеет вид прямоугольника ЕКLM, стороны которого проходят через точки А, В, С, D перпендикулярно осям - см. рис. 5.1.14. В области ЕКLM выделяется $N = 2^r$ точек, каждая из которых соответствует одному из линейных кодов комплексных чисел, объединенных в геометрический код. Расстояния между этими точками определяются величинами Δx и Δy , которые зависят от m и ρ : если $(m+1)$ - четное число или 0, то $\Delta x = |\rho|^{m+1}$ и $\Delta y = \Delta x |\rho|$; в противном случае $\Delta y = |\rho|^{m+1}$ и $\Delta x = \Delta y |\rho|$. Размер и расположение кодируемой области, в свою очередь, зависит от Δx , Δy , n .

Пример 5.1.6 кодирования плоскости при $\rho = j\sqrt{2}$. Пусть $m=-1$, $n=3$, $r=n-m+1=5$. Геометрический код для этого случая изображен на табл. 5.1.2, где перечислены также линейные коды, соответствующие путям в дереве геометрического кода (предполагается, что все пути открыты), и значения комплексных чисел, представленных этими кодами. В этой таблице (а также в следующей табл. 5.1.2а) обозначено:

N - номер точки,
 Z - значение – комплексное число этой точки,
 L - код этого комплексного числа – линейный код,
 G - геометрический код.

На рис. 5.1.15 изображены точки комплексной плоскости, соответствующие этим комплексным числам. Тем самым построен участок плоскости, кодируемый данным геометрическим кодом.

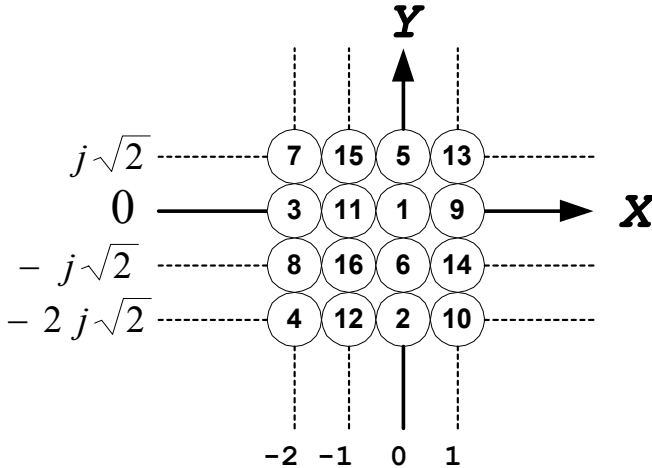


Рис. 5.1.15. Кодирование плоскости при $y=3$, $m=-1$, $r=4$ для примера 5.1.6.

Выделим на этом участке плоскости «черные» (видимые) и «белые» (невидимые) точки – см. рис. 5.1.15а. Геометрический код при этом будет иметь вид, приведенный на рис. 5.1.15в. Этот же код приведен в табл. 5.1.2а, где (в отличие от табл. 5.1.2) невидимые точки указаны без координат.

Таблица 5.1.2. Геометрический код плоскости при $\rho = j\sqrt{2}$.

N	Z	L	G
1	0+0	0000	11111
2	0-2j $\sqrt{2}$	1000	1
3	-2+0	0100	11
4	-2-2j $\sqrt{2}$	1100	1
5	0+j $\sqrt{2}$	0010	111
6	0-j $\sqrt{2}$	1010	1
7	-2+j $\sqrt{2}$	0110	11
8	-2-j $\sqrt{2}$	1110	1
9	1+0	0001	1111
10	1-2j $\sqrt{2}$	1001	1
11	-1+0	0101	11
12	-1-2j $\sqrt{2}$	1101	1
13	1+j $\sqrt{2}$	0011	111
14	1-j $\sqrt{2}$	1011	1
15	-1+j $\sqrt{2}$	0111	11
16	-1-j $\sqrt{2}$	1111	1

Таблица 5.1.2а. Геометрический код с выделенными точками плоскости при $\rho = j\sqrt{2}$.

N	Z	L	G
1	0+0	0000	11111
2			0
3	-2+0	0100	11
4			0
5			001
6			0
7			01
8	-2-j $\sqrt{2}$	1110	1
9			0111
10	1-2j $\sqrt{2}$	1001	1
11			01
12	-1-2j $\sqrt{2}$	1101	1
13	1+j $\sqrt{2}$	0011	111
14			0
15	-1+j $\sqrt{2}$	0111	11
16			0

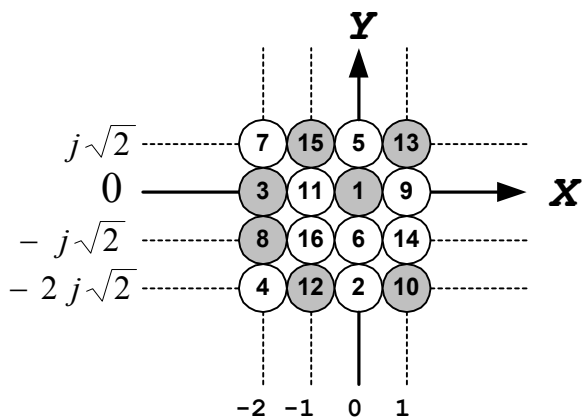


Рис 5.1.15а. Пример: плоская фигура.

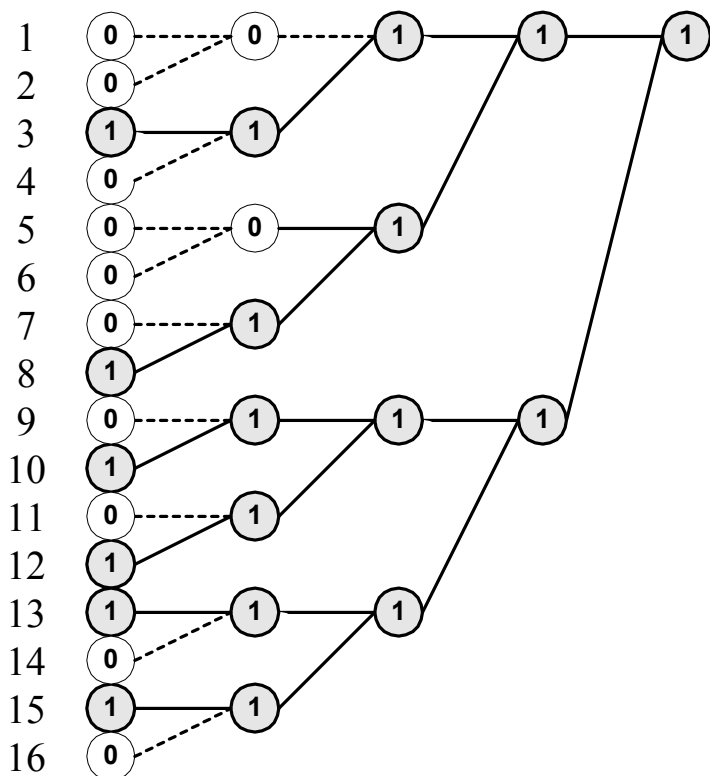


Рис 5.1.15б. Пример: дерево GC плоской фигуры.

Пусть $t = x + jy$ - произвольная точка плоскости, представленная линейным кодом в геометрическом коде по основанию $j\sqrt{2}$, а $b = |b| e^{j\varphi} = (c + jd)$ - комплексное число, представленное базисным кодом по тому же основанию. Рассмотрим те геометрические преобразования, которые эквивалентны арифметическим операциям между числами t и b .

5.1.4.2. Перенос

Перенос фигур по лучу $e^{j\varphi}$ на $|b|$ единиц эквивалентен операции $t + b$, т.е. сложению геометрического и базисного кодов.

5.1.4.3. Центроаффинное преобразование

Центроаффинное преобразование соответствует умножению действительной и мнимой частей геометрического кода одновременно на различные базисные коды $(c + jd)$ и $(g + jb)$ (*покомпонентное умножение*). Эту операцию описывает формула $z + jv = x(c + jd) + jy(g + jb)$ - здесь точка (x, y) переходит в точку (z, v) . В частных случаях центроаффинное преобразование превращается в *поворот, расширение, сдвиг* (но не ранее рассмотренный перенос) или некоторую комбинацию этих преобразований.

5.1.4.4. Аффинное преобразование

Аффинное преобразование является произведением центроаффинного преобразования и переноса и выполняется в два этапа:

1. покомпонентное умножение геометрического кода на пару базисных кодов центроаффинного преобразования,
2. сложение геометрического кода - результата предыдущей операции с базисным кодом переноса.

Пример 5.1.7 центроаффинного преобразования при

$\rho = j\sqrt{2}$ - см. рис 5.1.16 и табл. 5.1.3. Здесь обозначено:

i - номер точки,

a_i - точка исходной фигуры,

b_i - точка преобразованной фигуры,

$L(a_i)$ - линейный код точки a_i ,

$L(b_i)$ - линейный код точки b_i .

Рассмотрим фигуру, определяемую 6-ю точками a_i . Произведем центроаффинное преобразование этой фигуры так, чтобы точки $a_i = (x_i + jy_i)$ перешли в точки b_i , причем $b_i = (x_i + jy_i(1 + j\sqrt{2}))$. Это центроаффинное преобразование эквивалентно сдвигу фигуры по горизонтали на угол $\Psi = 55^0$ ($\text{tg}\Psi = \sqrt{2}$). Все коды чисел a_i изображаются единым геометрическим кодом \mathbf{G} исходной фигуры.

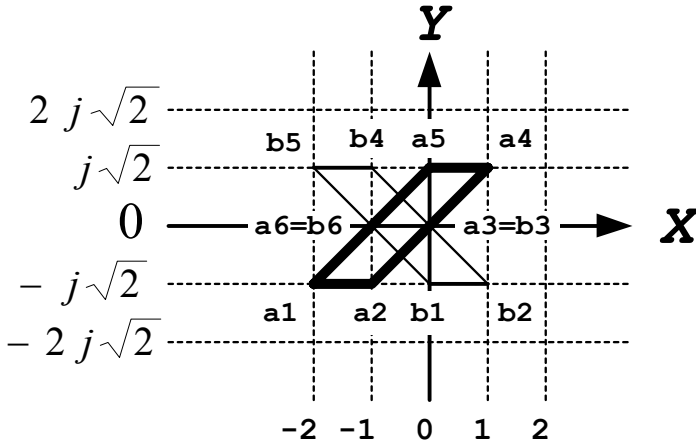


Рис. 5.1.16. Центроаффинное преобразование фигуры для примера 5.1.7

Таблица 5.1.3. Центроаффинное преобразование фигуры при $\rho = j\sqrt{2}$

N	a_i	$L(a_i)$	b_i	$L(b_i)$
1	$2-j\sqrt{2}$	1110	$0-j\sqrt{2}$	1010
2	$-1-j\sqrt{2}$	1111	$1-j\sqrt{2}$	1011
3	$0+0$	0000	$0+0$	0000
4	$1+j\sqrt{2}$	0011	$-1+j\sqrt{2}$	0111
5	$0+j\sqrt{2}$	0010	$-2+j\sqrt{2}$	0110
6	$-1+0$	0101	$-1+0$	0101

Указанное центроаффинное преобразование этой фигуры эквивалентно умножению мнимой части геометрического кода \mathbf{G} на базисный код $K=0011$ числа $(1 + j\sqrt{2})$. Такое умножение выполнено в примере 5.1.5. В результате образуется геометрический код \mathbf{R} . Декодируя код \mathbf{R} , находим, что он объединяет линейные коды точек b_i деформированной фигуры.

5.1.5. Кодирование и преобразование пространственных фигур.

В этом разделе будем полагать, что в качестве линейных кодов при построении геометрического кода используются линейные коды по основанию $\rho = j^3\sqrt{2}$. Арифметические операции с геометрическими кодами по такому основанию во многом аналогичны операциям с геометрическими кодами по основанию

$\rho = j\sqrt{2}$. Отличие заключается в том, что

- сложение кодов векторов эквивалентно сложению *трех* компонент,
- покомпонентное умножение геометрического и базисного кодов состоит в умножении каждой из *трех* компонент линейных кодов на различные базисные коды.

При кодировании трехмерных фигур будем полагать, что

- в трехмерном пространстве выделено N точек, распределенных равномерно с шагом Δx , Δy , Δz по осям прямоугольной системы координат;
- каждой точке может приписываться одно из двух значений - 0 или 1;
- фигура определяется подмножеством a точек, которым приписано единичное значение.

Если m - номер младшего яруса, n - номер старшего яруса, $r = (n - m + 1)$ - количество ярусов геометрического кода, то

- количество точек, выделенных в кодируемом участке пространства, $N = 2^r$;
- при $(m+1) = 3t$ (t - целое число) шаги по осям координат $\Delta x = |\rho|^{m+1}$, $\Delta y = \Delta x |\rho|$, $\Delta z = \Delta y |\rho|$;
- кодируемый участок пространства имеет вид параллелепипеда, грани которого параллельны координатным плоскостям.

Пусть U_1 - вектор произвольной точки трехмерной фигуры. Тогда по аналогии с предыдущим получаем, что

- сложение геометрического кода и базисного кода вектора U_5 эквивалентно сложению кодов U и U_5 , т.е. *переносу* фигуры на вектор U_5 ;

- покомпонентное умножение геометрического кода на упорядоченную тройку векторов U_2, U_3, U_4 эквивалентно аналогичной операции с каждым из векторов U_1 и состоит в вычислении вектора U по формуле $U = x_1 i * U_2 + y_1 j * U_3 + z_1 k * U_4$, т.е. эквивалентно *центроаффинному преобразованию*.

Аффинное преобразование является произведением центроаффинного преобразования и переноса и выполняется в два этапа:

1. покомпонентное умножение геометрического кода на тройку базисных кодов центроаффинного преобразования,
2. сложение геометрического кода - результата предыдущей операции с базисным кодом переноса.

В частных случаях это преобразование эквивалентно переносу, повороту, сжатию, сдвигу фигуры, векторному умножению всех векторов фигуры на базисный вектор и т.п. преобразованиям фигуры.

Аналогично кодам трехмерных фигур могут быть построены геометрические коды многомерных фигур, поскольку, как показано выше, в кольце многомерных векторов также существует позиционная система счисления двоичных кодов. Таким образом, геометрическим кодом можно закодировать многомерную фигуру и выполнять с ним аффинные преобразования этой фигуры. Последнее обстоятельство удобно использовать, например, при построении устройств для распознавания образов, поскольку признаки распознаваемых объектов часто инвариантны к определенному типу геометрических преобразований.

5.2. Атрибутные геометрические коды

5.2.1. Структура данных

Как следует из предыдущего, операции с PGC требуют многократного транспонирования. Схемы для выполнения транспонирования должны иметь большой объем, так как каждый

разряд PGC при транспонировании может поменяться местами с любым разрядом своего яруса.

Рассмотрим модификацию PGC, свободную от этого недостатка и назовем ее атрибутным GC - **AGC** (в данном разделе прилагательное «атрибутный» будет опускаться, если из контекста ясно, что речь не идет о первичном GC). Для каждой пары разрядов $\beta_{2i-1,k}$, $\alpha_{2i,k}$ в AGC включен дополнительный разряд $\gamma_{i,k}$, который является счетчиком по модулю 2 сигналов транспонирования $\tau_{i,k}$. При нулевом значении разрядов $\gamma_{i,k}$ (транспонирования не было или оно выполнялось четное число раз) коды PGC и AGC идентичны: каждому пути в дереве геометрического кода соответствует линейный код, в котором “1” стоит на месте α -разряда и “0” - на месте β -разряда. Если же значения разрядов $\gamma_{i,k} = (0,1)$, то линейный код, соответствующий данному пути в дереве AGC, определяется следующим образом: на месте $\alpha_{2i,k}$ -разряда стоит величина $\bar{\gamma}_{i,k}$, а на месте $\beta_{2i-1,k}$ -разряда - величина $\gamma_{i,k}$. Напомним, что значения разрядов α и β определяют лишь то, что путь в дереве геометрического кода открыт (или закрыт) и соответствующий линейный код включен (или не включен) в кодируемое множество линейных кодов.

Основное различие между PGC и AGC заключается в следующем. Пусть некоторому вектору X соответствует p -путь. При операции с геометрическим кодом величина этого вектора меняется на Y . В PGC после выполнения операции этому вектору будет соответствовать q -путь. Таким образом, *местоположение вектора* в PGC зависит от его величины. В отличие от этого, в AGC данному вектору всегда соответствует один и тот же путь. Можно сказать, что вектор (и соответствующая ему точка в пространстве), *сохраняет свою индивидуальность* вне зависимости от изменения величины этого вектора (местоположения точки). При этом точке можно присвоить атрибут, которым может быть имя, цвет, вес и т.п. Этот атрибут должен быть связан с терминальной вершиной того пути, где записан линейный код данного вектора (точки).

Рассмотрим изложенное более формально. Некоторому открытому пути в дереве AGC соответствует последовательность разрядов $\alpha=1$, $\beta=1$, $\gamma=(0,1)$:

$$\alpha_{p,n} \dots \beta_{2j-1,q} \dots \alpha_{2i,k} \dots \beta_{1,m} \\ \gamma_{p/2,n} \dots \gamma_{j,q} \dots \gamma_{i,k},$$

Этот путь изображает линейный код

$$\bar{\gamma}_{p/2,n} \dots \gamma_{j,q} \dots \bar{\gamma}_{i,k},$$

который будем называть кодом значения или, просто, значением данного пути. В этом пути пара разрядов $\alpha_{2i,k}$, $\gamma_{i,k}$ изображается разрядом $\bar{\gamma}_{i,k}$ значения, а пара разрядов $\beta_{2i-1,k}$, $\gamma_{i,k}$ изображается разрядом $\gamma_{i,k}$ значения.

При $\gamma \equiv 0$ для всех разрядов данного пути линейный код принимает значение

$$1 \dots 0 \dots 1 \dots 0,$$

которое будем называть кодом номера или, просто, номером данного пути. В этом коде “1” стоит на месте α -разряда и “0” - на месте β -разряда. Таким образом, в АГС каждый путь имеет номер, значение и атрибут.

На рис. 5.2.1 представлен АГС. Количество разрядов АГС определяется по следующей формуле:

$$V = 1 + 3 \sum_{k=0}^{n-m} 2^k = 3 \cdot 2^{n-m} - 2.$$

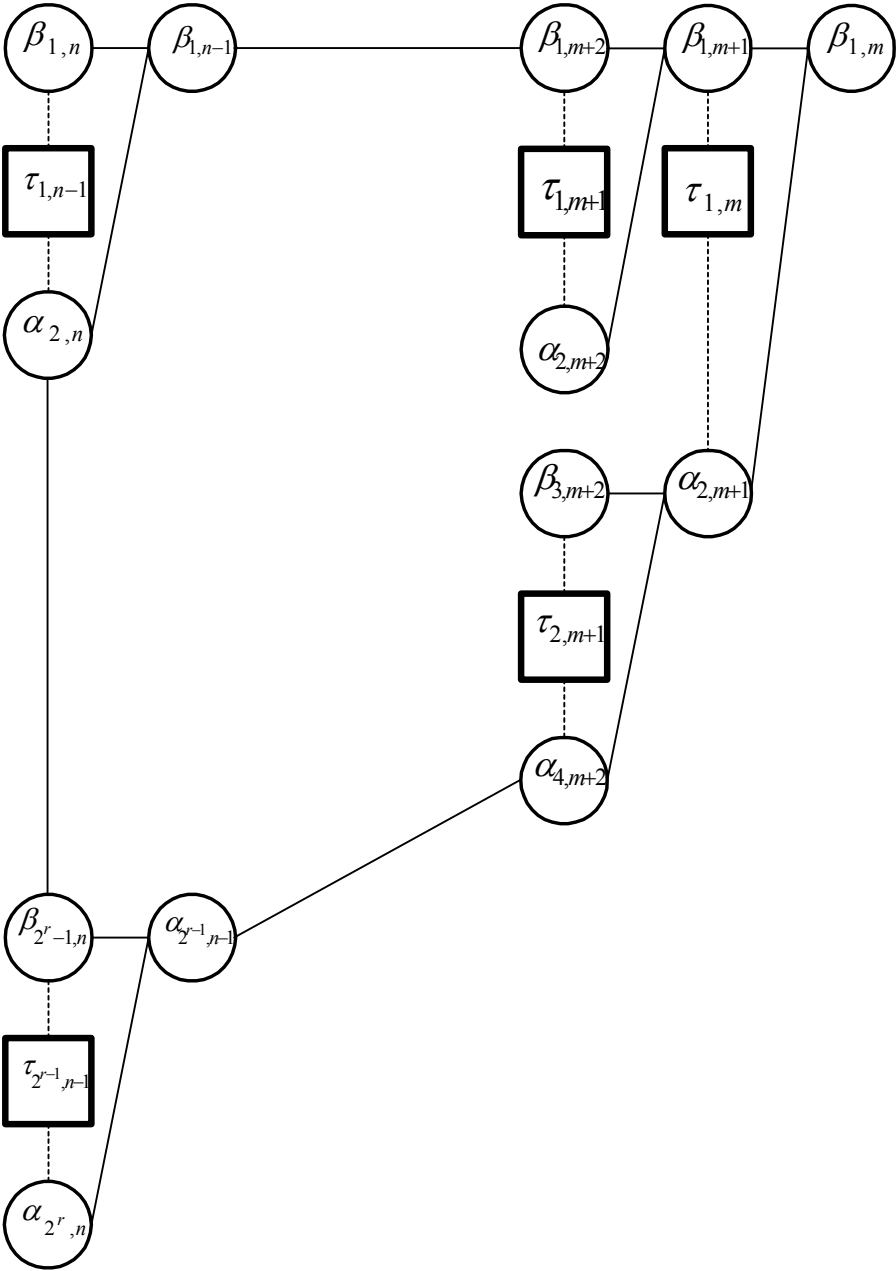


Рис. 5.2.1. Атрибутный геометрический код.

5.2.2. AGC по действительному основанию

Рассмотрим операции между базисным кодом и AGC по действительному основанию. При этом будем использовать обозначения из раздела 5.1.2.1.

5.2.2.1. Запись данного номера.

В этом случае в AGC формируется путь с номером, равным базисному коду δ . В том случае, когда все разряды $\gamma = 0$, код значения совпадает с кодом номера. Процесс определяется следующими формулами:

$$\mu = \pi \wedge \overline{\delta}, \quad \eta = \pi \wedge \delta.$$

При $\mu=1$ разряд β принимает значение “1” вне зависимости от его прежнего значения. Аналогично, при $\eta=1$ разряд α принимает значение “1”.

5.2.2.2. Запись данного значения.

Табл. 5.2.1 описывает процесс распространения переноса при записи в AGC значения, имеющего базисный код δ . Переносы определяются следующими формулами:

$$\mu = \pi \wedge \overline{(\delta \oplus \gamma)}, \quad \eta = \pi \wedge (\delta \oplus \gamma).$$

Таблица 5.2.1. Запись значения с данным кодом.

π	δ	γ	μ	η
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	0

При $\mu=1$ разряд β принимает значение “1” вне зависимости от его прежнего значения. Аналогично, при $\eta=1$ разряд α принимает значение “1”. Таким образом либо формируется новый путь и в него записывается данное значение, либо обнаруживается путь, в котором записано данное значение. В этом случае выполняется **поиск адреса данного значения.**

5.2.2.3. Чтение значения пути с данным номером.

Пусть открытый путь ($\beta \equiv 1$ и $\alpha \equiv 1$) имеет номер с линейным кодом δ . Процесс распространения переноса при чтении значения этого пути описывается табл. 5.2.2. В ней ω - соответствующий разряд линейного кода значения этого пути. Сигнал ω вырабатывается в том разряде α или β , через который прошел сигнал переноса. Итак,

$$\mu = \pi \wedge \bar{\delta}, \quad \eta = \pi \wedge \delta, \quad \omega = (\mu \wedge \gamma) \vee (\eta \wedge \bar{\gamma}).$$

Таблица 5.2.2. Чтение значения пути с данным номером

π	δ	γ	μ	η	ω
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	0	1	1
1	1	1	0	1	0

5.2.2.4. Сложение AGC с базисным кодом при $\rho=2$

описывается табл. 5.2.3. Сигналы переносов μ , η и сигнал транспонирования τ вырабатываются, как функции от π , δ , γ . После этого сигнал $\tau=1$ изменяет значение γ на противоположное, а сигналы μ и η распространяются дальше (если $\beta=1$ и $\alpha=1$ соответственно).

Таблица 5.2.3. Сложение AGC с базисным кодом при $\rho=2$

π	γ	δ	μ	η	τ
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	1	0	1
1	0	0	0	1	1
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	0

5.2.1 $\rho=2$. Пусть, как и в примере 5.1.1, базисный код $K=<2>$ или $K=10$, а атрибутивный геометрический код G изображает множество линейных кодов $\{1100, 0010, 1010, 0110\}$ или, что одно и то же, множество чисел $\{12, 2, 10, 6\}$.

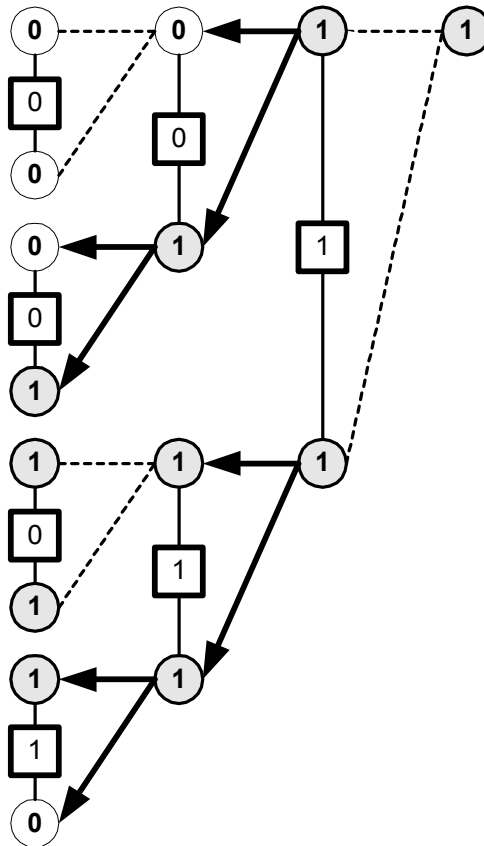


Рис. 5.2.2. К примеру 5.2.1.

Результирующий атрибутивный геометрический код $R=G+K$. На рис. 5.2.2 изображен код R . Толстыми стрелками указаны связи, по которым распространялся перенос $\pi=1$ при сложении. В квадратных окнах изображены разряды γ . Они относятся к той паре разрядов α и β , которые помещены в круглые окна, сопряженные с данным квадратным окном. Если все разряды γ обнулить, то тот же рисунок будет изображать исходный код G - сравни с геометрическим кодом G_1 в примере 5.1.1. Таким образом, результат отличается от исходного кода только

значениями разрядов γ . Заметим, что если выполнить транспонирование в соответствии со значениями разрядов γ , то образуется геометрический код G_4 результата, приведенный в примере 5.1.1.

5.2.2.5. Обратное сложение AGC с базисным кодом

при $\rho=-2$ описывается табл. 5.2.3а. Сигналы переносов μ , η и сигнал транспонирования τ вырабатываются, как функции от π , δ , γ . После этого сигнал $\tau=1$ изменяет значение γ на противоположное, а сигналы μ и η распространяются дальше (если $\beta=1$ и $\alpha=1$ соответственно).

Таблица 5.2.3а. Обратное сложение AGC с базисным кодом при $\rho=-2$

π	γ	δ	μ	η	τ
0	0	0	0	1	0
0	0	1	1	1	1
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	0	0	1
1	1	1	1	0	0

Пример 5.2.2 обратного сложения при $\rho=-2$.

Пусть, как и в примере 5.1.2, базисный код $\mathbf{K}=\langle 2 \rangle$ или $\mathbf{K} = 110$, а атрибутный геометрический код \mathbf{G} изображает множество линейных кодов $\{0000, 0100, 0010, 0110\}$ или, что одно и то же, множество чисел $\{0, 4, -2, 2\}$. Результирующий атрибутный геометрический код $\mathbf{R}=\mathbf{G}-\mathbf{K}$. На рис. 5.2.3 изображен код \mathbf{R} . Толстыми стрелками указаны связи, по которым распространялся перенос $\pi=1$ при сложении. В квадратных окнах изображены разряды γ . Они относятся к той паре разрядов α и β , которые помещены в круглые окна, сопряженные с данным квадратным окном. Если все разряды γ обнулить, то тот же рисунок будет изображать исходный код \mathbf{G} - сравни с геометрическим кодом G_1 в примере 5.1.2. Таким образом, результат отличается от исходного кода только значениями разрядов γ . Заметим, что если

выполнить транспонирование в соответствии со значениями разрядов γ , то образуется геометрический код G_4 результата, приведенный в примере 5.1.2.

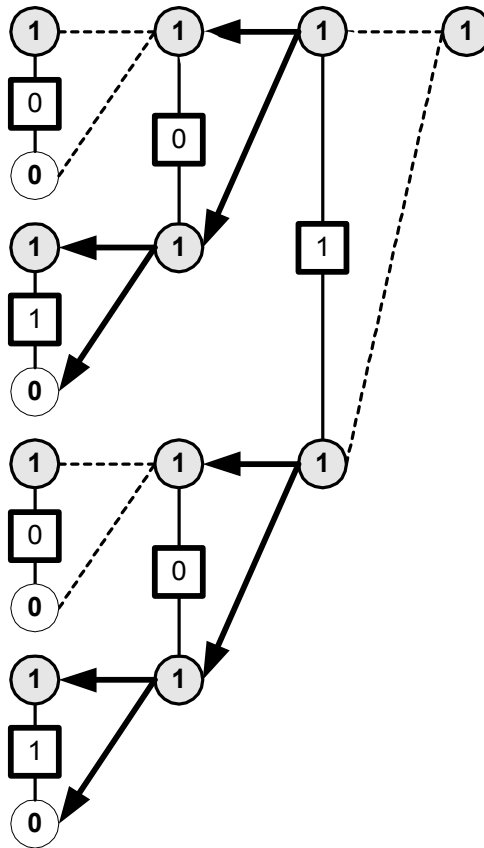


Рис. 5.2.3. К примеру 5.2.2.

Рассмотрим схему для формирования сигналов переносов μ , η и сигнала транспонирования τ в рассматриваемом сумматоре - см. рис. 5.2.3а. На этой схеме

- π - сигнал входного переноса,
- μ , η - сигналы выходных переносов,
- β - триггер разряда β ,
- α - триггер разряда α ,
- γ - триггер разряда γ ,
- δ - триггер разряда δ базисного кода,
- τ - триггер сигнала транспонирования τ ,

Sum - одноразрядная схема обратного сложения,

And - ключ сигнала транспонирования τ ,

R - сигнал разрешения транспонирования.

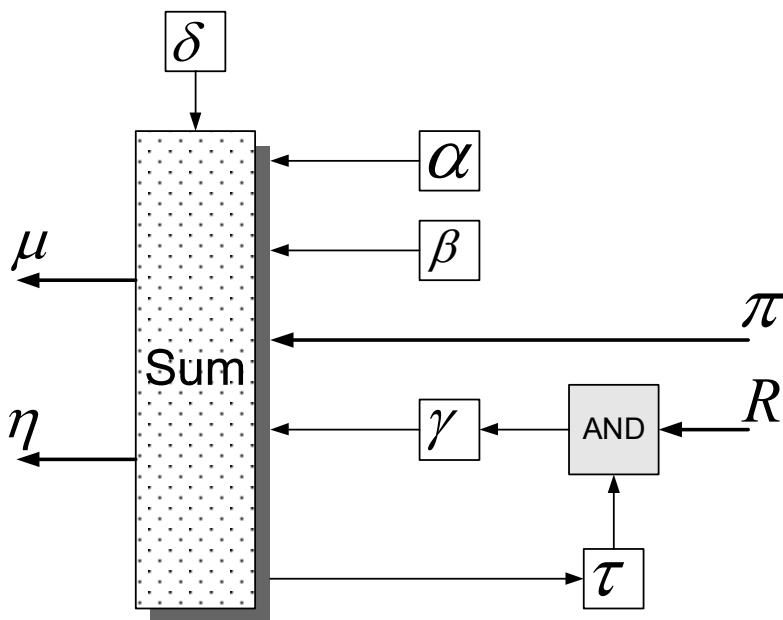


Рис. 5.2.3а. Одноразрядная схема обратного сложения

Эта схема охватывает тройку разрядов геометрического кода и вырабатывает сигналы переноса в две следующих тройки разрядов геометрического кода. Сигнал разрешения транспонирования является общим для всех разрядов и поступает из схемы управления после окончания распространения переносов через все разряды.

5.2.2.6. Инвертирование AGC при $\rho=-2$.

Табл. 5.2.4 описывает процесс распространения переноса при инвертировании геометрического кода по основанию «-2». В ней

γ - значение разряда γ в исходном коде;

τ - сигнала транспонирования для исходном кода в результирующий код (этот код образуется в регистре исходном кода).

Таблица 5.2.4. Инвертирование AGC при $p=2$.

π	γ	μ	η	τ
0	0	0	0	0
0	1	0	0	0
1	0	0	0	1
1	1	0	0	1

5.2.2.7. Алгебраическое сложение AGC распадается на операции инвертирования слагаемых и обратного сложения.

5.2.2.8. Поиск следующего открытого пути, его номера и его значения.

Здесь предполагается, что известный путь определен своим номером. Поиск состоит из трех последовательно выполняемых операций: 1) поиск пути с данным номером и фиксация его терминальной вершины; 2) поиск следующей терминальной вершины; 3) чтение номера и значения пути с данной терминальной вершиной.

5.2.2.9. Умножение AGC на базисный код.

Это умножение выполняется аналогично умножению первичного GC на базисный код – см. раздел 5.1.2.7. Отличие состоит в том, что анализируются разряды $\alpha=1$, если $\tau=0$, или $\beta=1$, если $\tau=1$ (а не разряды $\alpha=1$, как в первичном геометрическом коде). Это умножение выполняется по следующему алгоритму:

1. исходный геометрический код сдвигается влево на 1 разряд влево;
2. анализируется младший разряд B_j базисного кода, где $j = 1$;
3. если $B_j = 1$, то выполняется обратное сложение сдвинутого кода с исходным кодом; при этом образуется отрицательный геометрический код частичного произведения;
4. геометрический код частичного произведения сдвигается влево на 1 разряд влево;
5. анализируется следующий разряд B_j базисного кода;

6. если $B_j = 1$, и частичное произведение было положительным, то выполняется обратное сложение сдвинутого кода с исходным кодом; при этом образуется отрицательный геометрический код частичного произведения;
7. если $B_j = 1$, и частичное произведение было отрицательным, то выполняется обратное сложение сдвинутого кода с отрицательным исходным кодом; при этом образуется положительный геометрический код частичного произведения;
8. если все разряды исчерпаны, то умножение заканчивается;
9. выполняется переход к пункту 3.

Видно, что этот алгоритм во многом аналогичен алгоритму обычного сложения. Составляющие этот алгоритм операции были рассмотрены выше. Важно отметить, что в геометрическом коде произведения **номер атрибута увеличивается в 2^n раз** по отношению к номеру атрибута исходного геометрического кода (n – разрядность базисного кода, количество сдвигов).

5.2.3. Атрибутные геометрические коды по комплексному основанию

Как показано выше (см. раздел 3.1), в качестве основания кодирования линейных кодов могут использоваться комплексные числа. Аналогично этому могут быть построены атрибутные геометрические коды по комплексному основанию - **AGCC**. В отличие от предыдущего в таких кодах значением пути является линейный двоичный код по комплексному основанию. Но самое основное отличие состоит в алгоритмах арифметических операций. Рассмотрим алгоритмы арифметических операций с геометрическими кодами в системах кодирования 1, 2 и 3. В этих системах разряды, представляющие действительные и мнимые части комплексного числа, чередуются. Алгебраическое сложение каждой части выполняется независимо по правилам алгебраического сложения кодов действительных чисел по основанию «-2». Поэтому в дальнейшем изложении можно воспользоваться результатами предыдущего раздела. Некоторые

операции вовсе не зависят от основания кодирования и они здесь не рассматриваются.

5.2.3.1. Обратное сложение AGCC с базисным кодом

Эта операция описывается табл. 5.2.3. Сигналы переносов μ , η и сигнал транспонирования τ вырабатываются, как функции от π , δ , γ . После этого сигнал $\tau=1$ изменяет значение γ на противоположное, а сигналы μ и η распространяются дальше (если $\beta=1$ и $\alpha=1$ соответственно). Эти сигналы передаются через один ярус. Схема их распространения представлена на рис. 5.2.4.

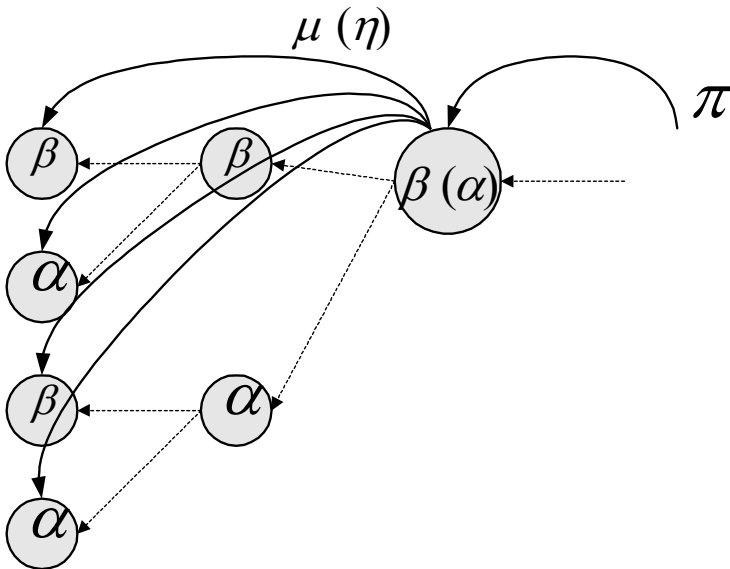


Рис. 5.2.4. Схема распространения переносов в комплексном GC

5.2.3.2. Инвертирование AGCC.

Эта операция производится аналогично тому, как описано в предыдущем разделе для кодов по основанию «-2».

5.2.3.3. Центраффинное преобразование AGCC.

Эта операция эквивалентна центраффинному преобразованию фигуры и выполняется по формуле

$$S = \text{Re } G \cdot Z' + \text{Im } G \cdot Z'',$$

где

G - исходный геометрический код,

S - результирующий геометрический код,

Z', Z'' - два линейных кода (комплексные числа).

Это умножение выполняется аналогично умножению первичного GC на базисный код – см. раздел 5.1.2.7. Отличие состоит в том, что анализируются разряды $\alpha=1$, если $\tau=0$, или $\beta=1$, если $\tau=1$ (а не разряды $\alpha=1$, как в первичном GC). Отличие состоит в том, что переносы при обратном сложении и инвертировании тут передаются через один ярус.

Центроаффинное преобразование возможно только в том случае, если комплексные коды Z', Z'' имеют единичный младший разряд. Для приведения к такому случаю эти коды должны быть преобразованы по формуле $Z \Rightarrow -\rho \cdot Z + 1$. Результирующий код после этого должен быть сдвинут на 1 разряд влево.

Укажем некоторые частные случаи:

- при $Z = Z' = Z''$ имеем обычное умножение:
 $S = G \cdot Z$,
- при $Z = Z' = Z'' = j$ имеем поворот на 90 градусов:
 $S = G \cdot j$,
- при $Z = Z' = Z'' = -j$ имеем поворот на (-90) градусов: $S = -G \cdot j$.

Последние две операции очень упрощаются в системе кодирования 1, когда код числа j имеет вид «10».

Пример 5.2.3 центроаффинного преобразования при

$\rho = j\sqrt{2}$. Рассмотрим этот пример центроаффинного преобразования AGC по аналогии с примером 5.1.7 центроаффинного преобразования GC. Рассмотрим фигуру 6-ю точками a_i - см. рис. 5.1.16 и табл. 5.1.3. Произведем центроаффинное преобразование этой фигуры так, чтобы точки $a_i = (x_i + jy_i)$ перешли в точки b_i , причем

$b_i = (x_i + jy_i(1 + j\sqrt{2})).$ Это центроаффинное преобразование эквивалентно сдвигу фигуры по горизонтали на угол $\Psi = 55^0$ ($tg\Psi = \sqrt{2}$). Все коды чисел a_i изображается единым AGC исходной фигуры. Этот код изображен на рис. 5.2.5 и объединяет точки исходной фигуры. В этом коде все разряды $\tau=0$. Декодируя этот код, можно убедиться, что линейные коды всех открытых путей имеют значение a_i , указанное в табл. 5.1.3. Код точки a_i для каждого открытого пути обозначен на рис. 5.2.5 напротив соответствующей терминальной вершины.

Указанное центроаффинное преобразование этой фигуры эквивалентно умножению мнимой части AGC на базисный код $K=0011$ числа $(1 + j\sqrt{2})$. В результате образуется AGC деформированной фигуры. Этот код изображен на рис. 5.2.6 и объединяет точки деформированной фигуры. Код точки каждого открытого пути изменяется, но положение этого пути сохраняется – сравни рис. 5.2.5 и рис. 5.2.6. В AGC деформированной фигуры НЕ все разряды $\tau=0$. Декодируя этот код, можно убедиться, что линейные коды всех открытых путей имеют значение b_i , т.е. он объединяет линейные коды точек b_i деформированной фигуры. Код точки b_i для каждого открытого пути обозначен на рис. 5.2.6 напротив соответствующей терминальной вершины.

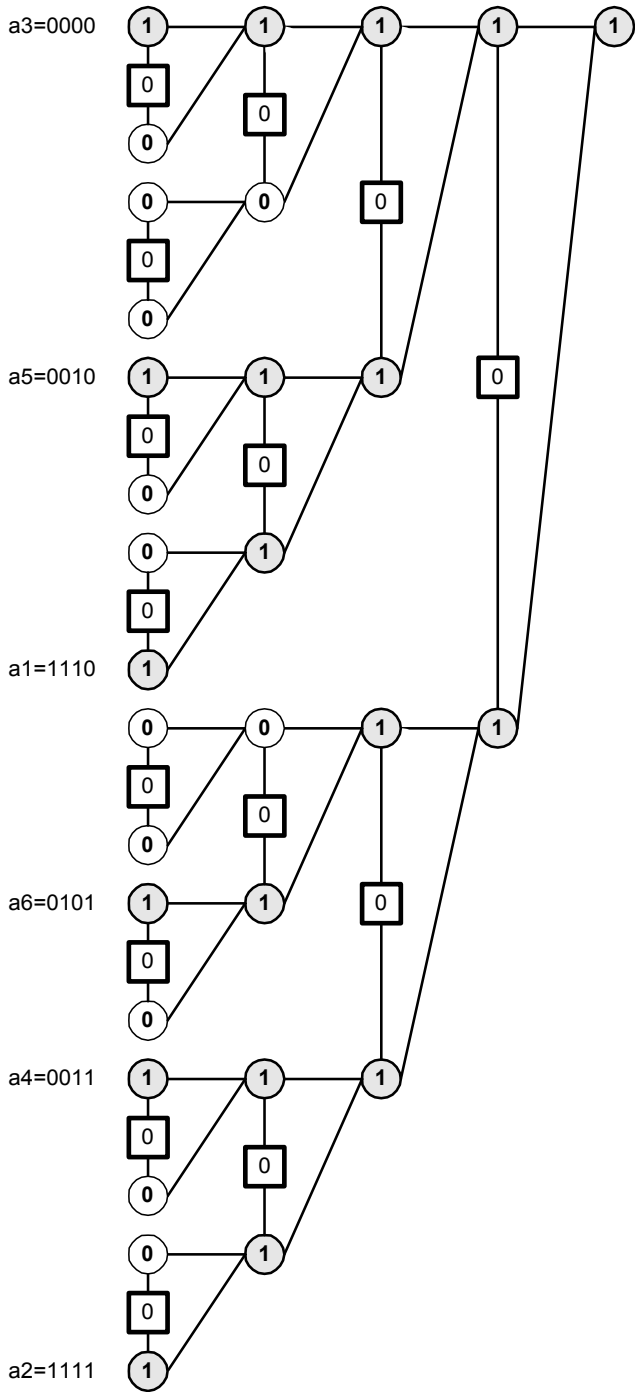


Рис. 5.2.5. К примеру 5.2.3: АГС исходной фигуры.

5.2.4. Атрибутные геометрические коды пространственных фигур

Атрибутные геометрические коды по комплексному основанию, рассмотренные выше, представляют плоские фигуры. Эти коды могут быть использованы для аффинных преобразований плоских фигур. В общем случае необходимо выполнять еще и проективные преобразования плоских фигур, а также аффинные и проективные преобразования пространственных фигур. Как известно, для проективных преобразований применяются однородные координаты. При этом точка на плоскости представляется тремя координатами, а точка в трехмерном пространстве – четырьмя координатами. При таком представлении проективное преобразование плоской фигуры включает аффинное преобразование трехмерной фигуры, а проективное преобразование трехмерной фигуры включает аффинное преобразование четырехмерной фигуры.

Таким образом, атрибутные геометрические коды плоских, трехмерных и четырехмерных фигур, с которыми выполнимы аффинные преобразования, могут быть использованы для решения всех задач геометрического преобразования. Для синтеза таких кодов применяется метод позиционного кодирования пространственных векторов - см. раздел 3. Этот метод, аналогично методу кодирования комплексных чисел, позволяет представить пространственный вектор единым двоичным кодом. Кроме того, этот метод позволяет выполнять алгебраическое сложение и умножение таких кодов.

Линейные двоичные коды векторов могут быть объединены в GC. При этом образуется GC трехмерной или четырехмерной фигуры. Арифметические операции с таким GC полностью аналогичны операциям с GC плоской фигуры. Схемы для сложения отличаются только тем, что переносы распространяются через 2 или 3 яруса (для трехмерной или четырехмерной фигуры соответственно).

Центроаффинное преобразование геометрического кода в общем случае выполняется по формуле

$$S = \text{part1}(G) \cdot Z_1 + \text{part2}(G) \cdot Z_2 + \text{part3}(G) \cdot Z_3 + \text{part4}(G) \cdot Z_4$$

где

G - исходный геометрический код,

S - результирующий геометрический код,

$part_p$ – одна из частей кода G ,

Z_p - линейные коды (векторы),

$p = \{1, 2, 3, 4\}$,

$\mathbf{i}, \mathbf{j}, \mathbf{k}, \mathbf{m}$ – орты векторного пространства,

$h \geq 0$ - целое,

r = номер яруса.

Как обычно, центроаффинное преобразование состоит в замене разрядов $\alpha_r = 1$ на линейные коды Z_p . При способе 2 кодирования векторов код замены Z выбирается следующим образом:

для трехмерных векторов

$$Z = Z_1, \quad \text{if } r = 3h+1,$$

$$Z = Z_2, \quad \text{if } r = 3h+2,$$

$$Z = Z_3, \quad \text{if } r = 3h+3;$$

для четырехмерных векторов

$$Z = Z_1, \quad \text{if } r = 4h+1,$$

$$Z = Z_2, \quad \text{if } r = 4h+2,$$

$$Z = Z_3, \quad \text{if } r = 4h+3,$$

$$Z = Z_4, \quad \text{if } r = 4h+4.$$

При способе 1 кодирования векторов код замены Z выбирается следующим образом:

для трехмерных векторов

$$Z = Z_1, \quad \text{if } r = 3h+1,$$

$$Z = j \cdot Z_2, \quad \text{if } r = 3h+2,$$

$$Z = k \cdot Z_3, \quad \text{if } r = 3h+3;$$

для четырехмерных векторов

$$Z = Z_1, \quad \text{if } r = 4h+1,$$

$$Z = j \cdot Z_2, \quad \text{if } r = 4h+2,$$

$$Z = k \cdot Z_3, \quad \text{if } r = 4h+3,$$

$$Z = m \cdot Z_3, \quad \text{if } r = 4h+4.$$

5.2.5. Сокращенные атрибутивные геометрические коды

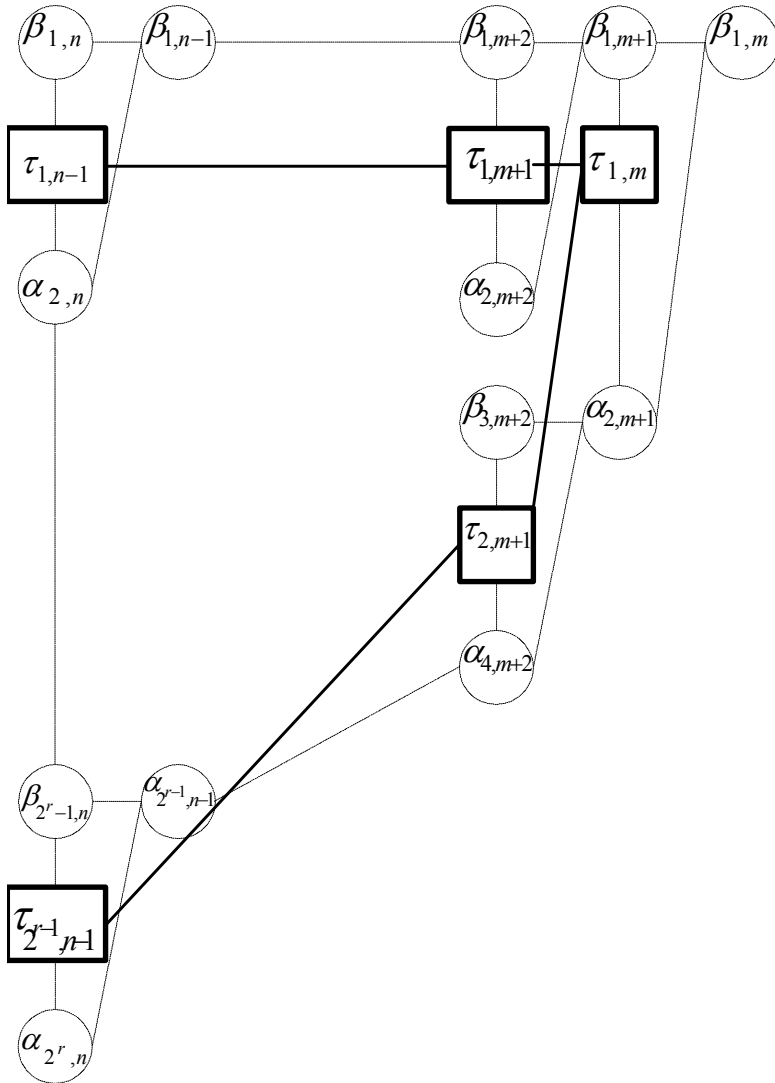


Рис. 5.2.7. Сокращенный AGC

Выше рассматривались AGC, в которых пути могли быть открытыми или закрытыми. При этом значения разрядов $\alpha=1$ (или 0) и $\beta=1$ (или 0) определяют лишь то, что путь в дереве геометрического кода открыт (или закрыт) и соответствующий линейный код включен (или не включен) в кодируемое множество

линейных кодов. Рассмотрим теперь случай, когда все линейные коды данной разрядности включены в кодируемое множество. При этом все $\alpha=1$ и все $\beta=1$. В таком случае нет необходимости реально включать эти разряды в геометрический код. Во всех манипуляциях с АГС можно предполагать, что все $\alpha=1$ и все $\beta=1$. АГС без разрядов будем называть сокращенным АГС - **САГС**.

На рис. 5.2.7 представлен сокращенный АГС. Разряды, реально отсутствующие, обозначены пунктиром. Количество разрядов сокращенного АГС определяется по следующей формуле:

$$V = \sum_{k=0}^{n-m} 2^k = 2^r - 1.$$

Каждой терминальной вершине $\tau_{k,n-1}$ дерева сокращенного АГС соответствует два атрибута – верхний, соответствующий мнимой вершине $\beta_{k,n}$, и нижний, соответствующий мнимой вершине $\alpha_{k+1,n}$.

Очевидно, в сокращенном АГС операционные схемы существенно сокращаются, а его объем сокращается в три раза. Точнее, если в обычном АГС количество разрядов

$V = 3 \cdot 2^r - 2$, то в сокращенном АГС количество разрядов $V = 2^r - 1$.

Глава 6.

Геометрический процессор

6.0. Представление данных

Будем, как и ранее, полагать, что дано множество (M) точек в p -мерном пространстве. Точки образуют область определения, которая представляет собой p -мерный куб, и распределены в этой области по узлам равномерной сетки. Координаты узла представляются p -разрядным кодом вектора с фиксированной точкой. Вся область определена множеством этих кодов, общее число которых равно $M=2^{pn}$. Каждый узел определяется триадой «адрес-вектор координат-атрибут».

Рассмотрим представление данных в оперативной памяти и арифметическом устройстве

Возможны два способа организации оперативной памяти. Первый, *простой способ*, состоит в том, что в памяти создаются два взаимосвязанных массива. В первом из них находятся векторы координат, а во втором – атрибуты. Для этого способа оперативная память (RAM) может быть выполнена и как динамическая (DRAM), и как статическая (SRAM). Будем называть память, реализующую этот способ, традиционной оперативной памятью **TRAM**.

Второй *способ, использующий ГК*, предполагает, что все коды векторов координат объединены в AGC. При этом каждому пути в AGC соответствует значение, интерпретируемое как «вектор координат», и номер, интерпретируемый как «адрес». Атрибуты в этом способе, по-прежнему, объединены в массив, связанный по адресам с AGC. Для этого способа оперативная память должна быть выполнена, как статическая (SRAM), поскольку должна быть обеспечена опеределенная логика доступа к памяти AGC. В дальнейшем будем называть оперативную память, реализующую этот способ, *специализированной оперативной памятью* **SSRAM**.

В дальнейшем будет рассматриваться только SSRAM, поскольку она (в отличие от обычной оперативной памяти) позволяет за одно

обращение к памяти определять адрес данного кода. Такой метод доступа необходим для поиска атрибута вектора и существенно повышает быстродействие процессора. Кроме того, SSRAM намного экономичнее обычной оперативной памяти. Точнее, массив координат в обычной оперативной памяти содержит 2^{pn} разрядов, а сокращенный AGC содержит 2^{pn} разрядов. Например, при $p=3$ и $n=12$ память сокращается в 36 раз. Поэтому ниже рассматривается только SSRAM.

Можно предложить две схемы построения памяти для AGC:

- *полная*, когда весь код AGC вместе со схемами распространения переносов храниться в **PSSRAM**,
- *фрагментарная*, когда весь код AGC разбит на фрагменты, хранящиеся в обычной оперативной памяти, и имеется **FSSRAM** со схемами распространения переносов для одного фрагмента. Такую структуру AGC назовем вертикальной фрагментацией.

Рассмотрим теперь представление данных в арифметическом устройстве. Первый способ заключается в том, что в АУ создается регистр полного AGC разрядностью 2^{pn} . Однако, этого не достаточно, поскольку при операциях с AGC в каждом пути из старшего разряда могут возникнуть переносы (по аналогии с векторным процессором). Код результата может иметь разрядность pr , тогда как исходный код имеет разрядность pn . Старшие разряды кодов результата могут быть объединены в прямоугольный код векторов RCV (аналогично тому, как это было сделано в векторном процессоре для полноразмерных кодов). Таким образом, в АУ должен быть регистр AGC и регистр RCV. Назовем пару этих кодов *смешанным кодом фигуры* - **MCF**. Регистр смешанного кода имеет разрядность $pr+2^{pn}$.

Арифметическое устройство с регистром MCF одновременно выполняет и функции оперативной памяти. Назовем его *максимальным арифметическим устройством геометрических фигур* **MGAU**. Очевидно, это устройство имеет очень большой объем и возможность его реализация находится на пределе возможностей современной технологии. Поэтому рассмотрим еще один вариант.

Для этого разделим MCF на несколько фрагментов MCF_q , каждый из которых состоит из фрагмента AGC_q и фрагмента RCV_q . MCF_q объединяет Q путей в коде MCF, т.е. Q кодов результата

разрядностью $(n+r)$. Если $Q=2^f$, то младшие ярусы фрагмента кода AGC_q сосредоточатся в одном пути и MCF_q будет иметь структуру, представленную на рис. 6.0.1.

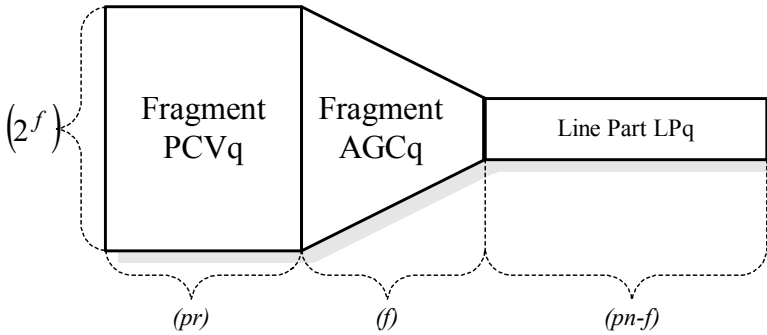


Рис. 6.0.1. Структура MCF_q

Такую структуру MCF_q назовем горизонтальной фрагментацией. При этом

- линейная часть LP_q кода MCF_q содержит $(pn-f)$ разрядов,
- прямоугольная часть PCV_q кода MCF_q содержит (Qpr) разрядов,
- геометрическая часть AGC_q кода MCF_q содержит (Q) разрядов,
- код MCF_q в целом содержит $((pn-f)+Qpr+Q)$ разрядов,
- полный MCF состоит из 2^{pn-f} фрагментов MCF_q и содержит $((pn-f)+Qpr+Q)*2^{pn-f}$ разрядов.

Соответственно, аффинное преобразование при такой структуре данных состоит из 2^{pn-f} операций. Арифметическое устройство с регистром такой структуры назовем фрагментарным GAU - FGAU.

Заметим, что горизонтальная фрагментация целесообразна для арифметического устройства, а вертикальная фрагментация целесообразна для оперативной памяти.

6.1. Полная специализированная оперативная память

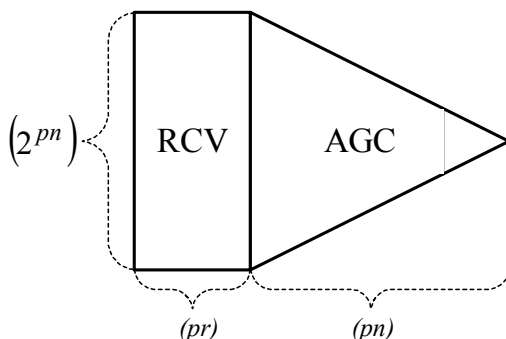


Рис. 6.1.1. Полная специализированная оперативная память

Полная специализированная оперативная память **PSSRAM** кода MCF представлена на рис. 6.1.1 и состоит из двух частей – памяти для прямоугольного кода **RCV** и памяти для AGC. Она дополняется некоторой схемой распространения переносов и благодаря этому может выполнять следующие операции:

- S1. Запись данного кода вектора – см. раздел 5.2.2.1.
- S2. Определение адреса, по которому расположен данный код вектора – см. раздел 5.2.2.2. Эта операция необходима для поиска атрибута вектора.
- S3. Чтение кода вектора по адресу, в котором он расположен – см. раздел 5.2.2.3.
- S4. Запись фрагмента кода MCF_q по его номеру q .
- S5. Чтение фрагмента кода MCF_q по его номеру q .
- S6. Определение номера старшего значащего разряда в коде MCF, что необходимо для округления.

6.2. Фрагментарная специализированная оперативная память

В этом случае для представления в памяти геометрический код G разбивается на фрагменты, объединенные в F ярусов. Фрагмент каждого яруса содержит r ярусов геометрического кода. Сказанное иллюстрируется на рис. 6.2.1, где треугольниками изображены фрагменты геометрического кода. Их нумерация имеет следующий смысл: «номер яруса», «номер фрагмента в ярусе».

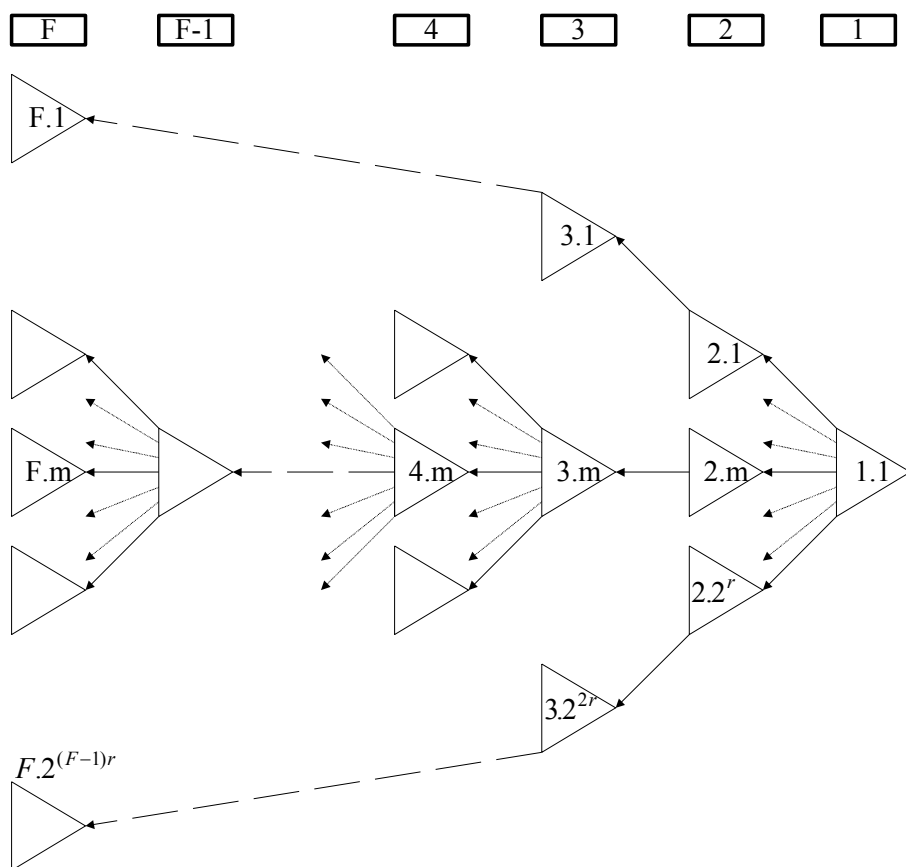


Рис. 6.2.1. Ярусы геометрического кода и сегменты линейного кода.

В памяти фрагменты располагаются последовательно ярус за ярусом:

$$\begin{aligned}
 &1.1, \\
 &2.1, 2.2, \dots, 2.2^r, \\
 &3.1, 3.2, \dots, 3.2^{2r}, \\
 &\dots \\
 &k.1, k.2, \dots, k.m, \dots, k.2^{(k-1)r}, \\
 &\dots \\
 &F.1, F.2, \dots, F.2^{(F-1)r}
 \end{aligned}$$

При этом номер j фрагмента в этой последовательности связан с номером k яруса и номером m фрагмента в ярусе следующей зависимостью:

$$j = m + \sum_{a=1}^{k-1} 2^{(a-1)r} = m + \frac{1 - 2^{(k-1)r}}{1 - 2^r}. \quad (6.2.1)$$

Если известны

- номер k яруса фрагментов, в котором расположен обрабатываемый фрагмент,
- номер m обрабатываемого фрагмента в k ярусе,
- номер v вершины в последнем ярусе обрабатываемого фрагмента, из которой выходит перенос,

то номер f фрагмента в следующем $(k+1)$ -ярусе, куда поступает этот перенос, определяется по формуле:

$$f = m + v - 1. \quad (6.2.2)$$

При этом номер фрагмента, куда поступает перенос,

$$j = f + \frac{1 - 2^{kr}}{1 - 2^r}. \quad (6.2.3)$$

Фрагменты, пронумерованные в соответствии с (6.2.1), хранятся в обычной оперативной памяти фрагментов, вызываются в устройство FSSRAM для обработки и после обработки возвращаются в память фрагментов. Фрагмент записывается или считывается из памяти фрагментов по номеру фрагмента в этой памяти. Устройство FSSRAM реализует перечисленные выше команды S1-S6, для чего

обращается к некоторым фрагментам. Непосредственно с фрагментом в устройстве FSSRAM выполняются следующие операции:

- F1. Чтение фрагмента с данным номером из памяти фрагментов.
- F2. Запись данного ОТРЕЗКА кода вектора по номеру, имеющему такой же код – см. раздел 5.2.2.1. Эта операция применима только в том случае, когда все разряды $\gamma = 0$, т.е. при начальном формировании AGC
- F3. Определение адреса, по которому расположен данный ОТРЕЗОК кода вектора – см. раздел 5.2.2.2.
- F4. Чтение ОТРЕЗКА кода вектора по адресу, в котором он расположен – см. раздел 5.2.2.3.
- F5. Запись фрагмента с данным номером в память фрагментов.
- F6. Определение номера старшего значащего разряда во фрагменте (используется только во фрагментах старшего яруса).
- F7. Определение по формуле (6.2.3) номера j следующего фрагмента, куда поступает перенос.

Сравним объем и быстродействие различных способов организации оперативной памяти. При оценке быстродействия будем полагать, что все пути в дереве GC открыты, т.е. он объединяет все коды данной разрядности. Обозначим

r - количество ярусов во фрагменте,

F - количество ярусов фрагментов в GC,

При этом имеем:

$n = r \cdot F$ - разрядность линейных кодов и количество ярусов GC

2^r - количество терминальных вершин во фрагменте,

$(2^{2^r} - 1)$ - общее количество вершин во фрагменте,

$2^{(F-1)r}$ - количество терминальных фрагментов в GC.

Как показано выше, количество разрядов в сокращенного AGC

$$V = 2^n - 1.$$

Этот код объединяет все n - разрядные коды. Общее количество бит для хранения этих кодов

$$V' = n \cdot 2^n.$$

Таким образом, применение AGC сокращает объем данных в n раз.

Рассмотрим теперь количество элементарных операций при фрагментарной записи AGC. В младшем ярусе фрагментов выполняется 1 операция с фрагментом, во втором ярусе – 2^r операций, в третьем – 2^{2r} операций, ..., в старшем – $2^{(F-1)r}$ операций. Таким образом, выполняется

$$a_1 = 1 + 2^r + 2^{2r} + \dots + 2^{(F-1)r} = \frac{1 - 2^{Fr}}{1 - 2^r}$$

элементарных операций с фрагментом или

$$a_1 \approx 2^{(F-1)r} = 2^{n-r} \quad (6.2.4)$$

Отношение разрядности оперативной памяти всех фрагментов (SSRAM) к разрядности одного фрагмента SSRAM

$$R \approx 2^{n-r} \quad (6.2.5)$$

Объем устройства FSSRAM превышает объем памяти одного фрагмента примерно в 3 раза из-за того, что это устройство содержит схемы переноса в каждом разряде. Таким образом, сложность оперативной памяти характеризуется величиной

$$\theta \approx 2^n + 3 \cdot 2^r = (3 + 2^F) \cdot 2^r \quad (6.2.6)$$

6.3. Максимальное арифметическое устройство геометрических фигур

Максимальное арифметическое устройство геометрических фигур **MGAU** аналогично полной специализированной оперативной памяти PSSRAM оперирует с кодом MCF представленным на рис. 6.1.1. Это устройство содержит развитую схему распространения переносов и благодаря этому может выполнять следующие операции:

- М1. Запись данного кода вектора – см. раздел 5.2.2.1.
- М2. Резерв.
- М3. Чтение кода вектора по адресу, в котором он расположен – см. раздел 5.2.2.3.
- М4. Резерв.

- М5. Резерв.
- М6. Определение номера старшего значащего разряда в коде MCF, что необходимо для округления.
- М7. Сложение MCF с данным кодом вектора – см. раздел 5.2.2.7.
- М8. Умножение MCF на матрицу преобразования – см. раздел 5.2.2.9.
- М9. Определение длины кода вектора по адресу.
- М10. Чтение из MGAU кода вектора по первому\следующему адресу – см. раздел 5.2.2.8.

6.4. Фрагментарное арифметическое устройство геометрических фигур

Арифметическое устройство **FGAU** для операций с кодами MCF_q представлено на рис. 6.4.1. Оно во-многом аналогично устройству FVAU. Отличие состоит в том, что FVAU содержит операционный блок разрядностью

$$R_6 = Qp(n+r), \quad (6.4.1)$$

а FGAU содержит операционный блок разрядностью

$$R_7 = (pn-f) + Qpr + 2^{pn-f}. \quad (6.4.2)$$

Операционный блок состоит из регистра MCF_q и схем распространения переноса. При этом в линейной части и прямоугольной части схемы переноса организованы по правилам векторной арифметики, а в геометрической части – по правилам арифметики геометрических кодов.

Соотношение между разрядностью операционных блоков FVAU и FGAU

$$R_6 / R_7 \approx (n+r)/r \quad (6.4.3)$$

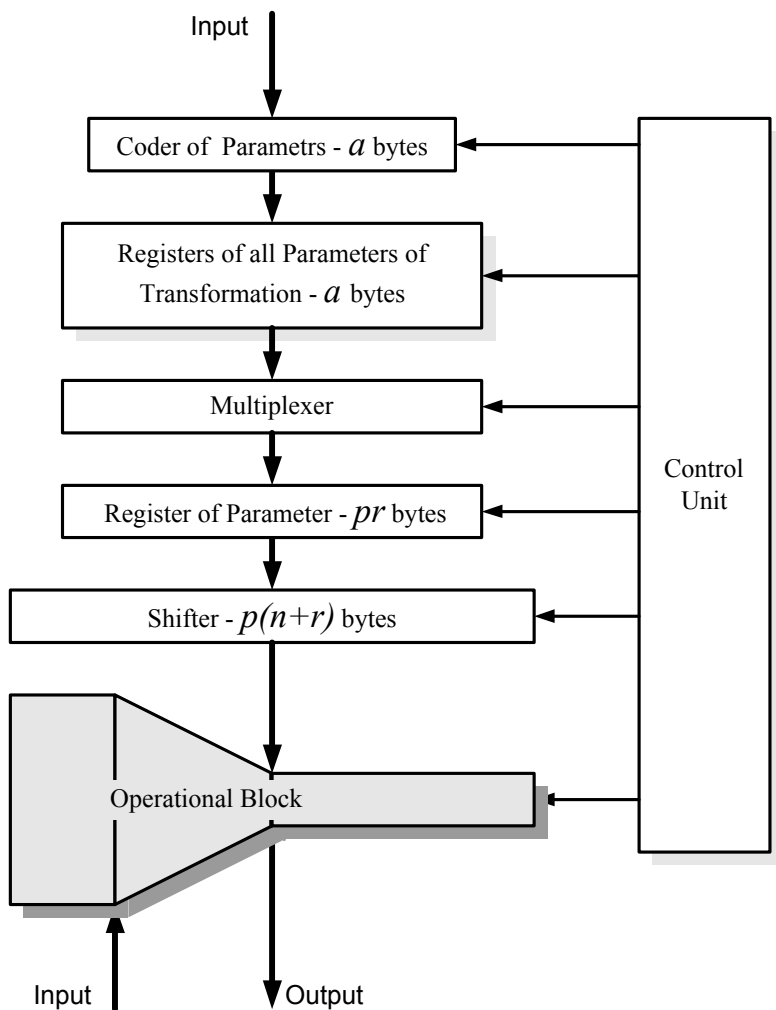


Рис. 6.4.1. Фрагментарное арифметическое устройство

Рассмотрим перечень команд со-процессора, реализуемых в FGAU:

- A1. Прием параметров преобразования.
- A2. Сложение MCF_q с вектором переноса.
- A3. Умножение MCF_q на матрицу преобразования.
- A4. Выдача кода MCF_q .
- A5. Прием кода MCF_q .

- А6. Выдача вектора по адресу - без округления или с округлением.
- А7. Определение длины кода вектора по адресу.
- А8. Определение вектора, соседнего с данным вектором, по известной координате и известному направлению по координатной оси.

6.5. Процессор с максимальным арифметическим устройством

Процессор с максимальным арифметическим устройством **PMGAU** использует арифметическое устройство MGAU для операций с кодами MCF, которое совмещает функции АУ и оперативной памяти. Со-процессор PMGAU и его место в центральном процессоре показано на рис. 6.5.1.

Итак, PMGAU содержит арифметическое устройство MGAU и блок дополнительной специализированной оперативной памяти SSRAM, кодер\декодер векторов и управляющее устройство, а также другие блоки аналогично устройству FVAU. Кодер\декодер связан с центральной оперативной памятью центрального процессора. MGAU и SSRAM связаны с блоками обычной оперативной памяти атрибутов ARAM-1 и ARAM-2, входящими в состав центрального процессора.

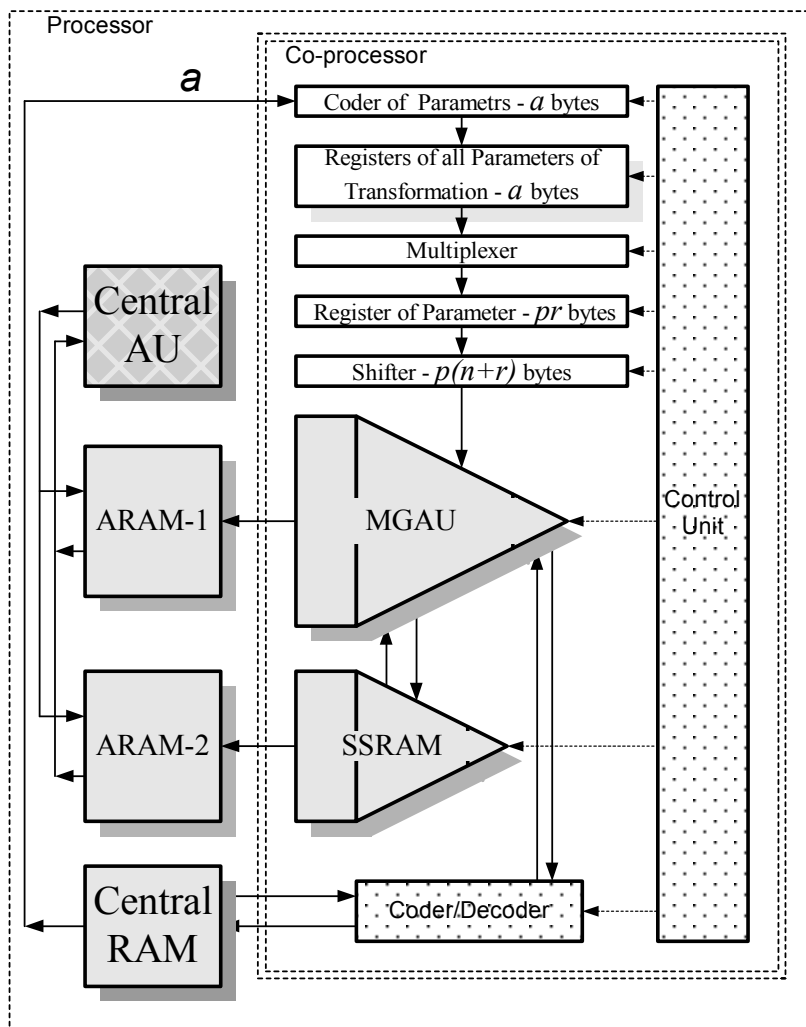


Рис. 6.5.1. Процессор с максимальным арифметическим устройством

Необходимо отметить, что со-процессор полностью освобождает центральный процессор от решения соответствующих задач так, что последний может параллельно с со-процессором решать другие задачи. Рассмотрим перечень команд со-процессора и устройства, используемые при выполнении этих команд:

- R1. Прием (по шине a) и кодирование параметров преобразования.
- R2. Сложение MCF с вектором переноса (см. операцию $M\bar{7}$).

- R3. Умножение MCF на матрицу преобразования (см. операцию *M8*).
- R4. Резерв.
- R5. Резерв.
- R6. Определение номера старшего значащего разряда в MGAU для округления (см. операцию *M6*).
- R7. Передача округленного k-вектора из MGAU в SSRAM.
- R8. Определение длины кода вектора по адресу (см. операцию *M9*).
- R9. Чтение вектора по данному адресу в SSRAM (см. операцию *S3*).
- R10. Определение вектора, соседнего с данным вектором, по известной координате и известному направлению.
- R11. Определение адреса по известному вектору в SSRAM (см. операцию *S2*).
- R12. Преобразование координат в вектор, запись его в MGAU и определение его адреса (см. операцию *M1*). При этом кодирование координат точки в код вектора выполняется на кодере.
- R13. Чтение из MGAU кода вектора по данному адресу (см. операцию *M3*) и преобразование этого вектора в координаты точки, что выполняется на декодере.
- R14. Чтение из MGAU кода вектора по первому\следующему адресу (см. операцию *M10*).

6.6. Процессор с фрагментарным арифметическим устройством

Со-процессор **PFGAU** с фрагментарным FGAU и его место в центральном процессоре показано на рис 6.6.1. Со-процессор содержит арифметическое устройство FGAU, блок

специализированной оперативной памяти SSRAM-1 и блок дополнительной специализированной оперативной памяти SSRAM-2, кодер\декодер векторов и управляющее устройство.

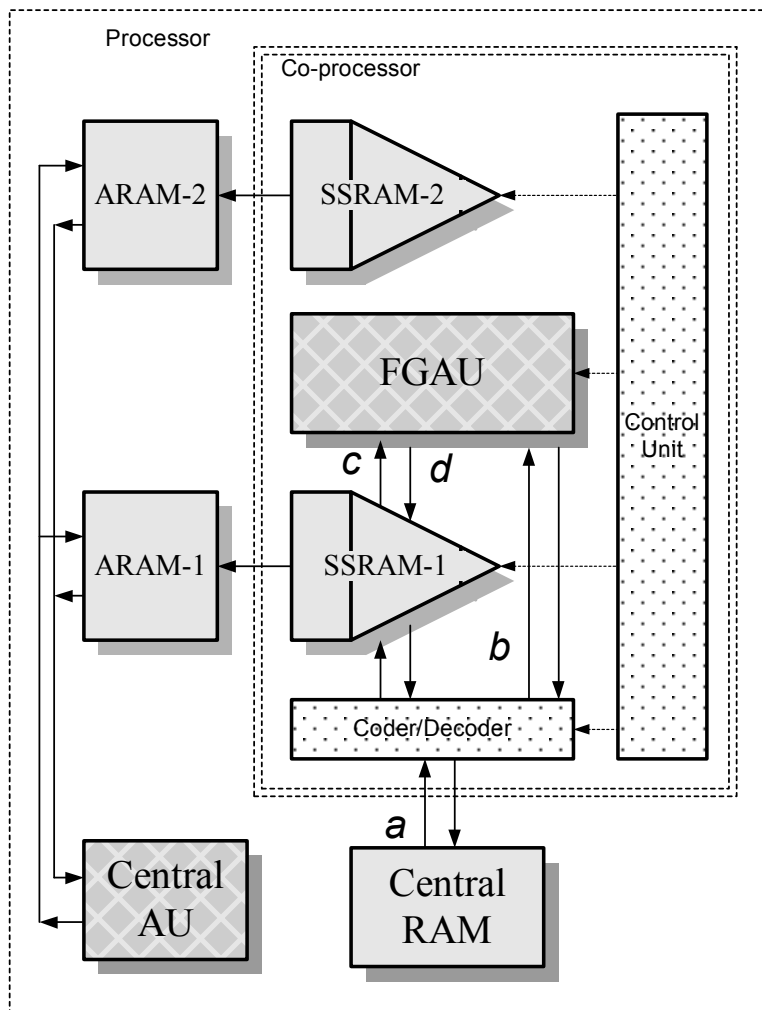


Рис 6.6.1. Процессор с фрагментарным FG AU

Кодер\декодер связан с центральной оперативной памятью центрального процессора. SSRAM-1 и SSRAM-2 связаны с блоками обычной оперативной памяти атрибутов ARAM-1 и ARAM-2, входящими в состав центрального процессора.

Необходимо отметить, что со-процессор полностью освобождает основной процессор от решения соответствующих

задач так, что основной процессор может параллельно с со-процессором решать другие задачи.

Из дальнейшего следует, что в памяти SSRAM-2 должны быть предусмотрены только две операции: $S1$ и $S3$.

Рассмотрим перечень команд со-процессора и устройства, используемые при выполнении этих команд (операции арифметического устройства FGAU обозначаются символами A):

- P1. Прием и кодирование параметров преобразования. Эти параметры передаются по шинам a и b . При этом используется операция A1.
- P2. Сложение MCF_q с вектором переноса в GAU. При этом используется операция A2.
- P3. Умножение MCF_q на матрицу преобразования в GAU. При этом используется операция A3.
- P4. Выдача кода MCF_q из GAU в SSRAM-1 по шине d . При этом используются операции A4 и S4.
- P5. Прием кода MCF_q из SSRAM-1 в GAU по шине c . При этом используются операции A5 и S5.
- P6. Определение номера старшего значащего разряда в SSRAM-1 для округления. При этом используется операция S6.
- P7. Передача округленного k -вектора из GAU в SSRAM-2. При этом используются операции A6 и S1.
- P8. Определение длины кода вектора по адресу. Предполагается, что соответствующий фрагмент находится в GAU и выполняется операция A7.
- P9. Чтение вектора по данному адресу. При этом используется операция S3 в SSRAM-2.
- P10. Определение вектора, соседнего с данным вектором, по известной координате и известному направлению. При этом используется операция A8.
- P11. Определение адреса по известному вектору. При этом используется операция S2 в SSRAM-2.
- P12. Преобразование координат в вектор, запись его в SSRAM-1 и определение его адреса. При этом

кодирование координат точки в код вектора выполняется на кодере, а для записи вектора и определения его адреса. используется операция S1.

P13. Чтение из SSRAM-1 кода вектора по данному адресу и преобразование этого вектора в координаты точки. При этом для чтения кода вектора используется операция S3, а его декодирование в координаты точки выполняется на декодере.

P14. Чтение из SSRAM-1 кода вектора по первому\следующему адресу. При этом для чтения кода вектора используется операция S3.

Заметим, что для SSRAM-2 используются только операции S1, S2, S3.

6.7. Основные процедуры.

Здесь будут использоваться следующие (описанные выше) обозначения устройств и выполняемых ими операций:

Раздел	Устройство	Тип	Используемые устройства	Операции
6.1	PSSRAM	RAM	-	S1-S6
6.2	FSSRAM	RAM	-	S1-S6; F1-F7
6.3	MGAU	AU	-	M1-M10
6.4	FGAU	AU	-	A1-A8
6.5	PMGAU	процессор	MGAU; PSSRAM	R1-R14
6.6	PFGAU	процессор	FGAU; PSSRAM or FSSRAM	P1-P14

6.7.1. Аффинное преобразование

В процессоре PFGAU:

1. Прием и кодирование параметров преобразования - операция P1.
2. Перебор всех q -фрагментов (в центральном процессоре).

2.1. Прием кода MCF_q из SSRAM-1 - операция P5.

- 2.2. Умножение MCF_q на матрицу преобразования - центроаффинное преобразование - операция P3.
- 2.3. Сложение MCF_q с вектором переноса - операция P2.
- 2.4. Выдача кода MCF_q из GAU в SSRAM-1 - операция P4.

В процессоре PMGAU:

1. Прием и кодирование параметров преобразования - операция R1.
2. Умножение MCF на матрицу преобразования - операция R7.
3. Сложение MCF с вектором переноса - операция. R2.

6.7.2. Округление.

Так будем называть операцию построения массива пар «координаты точки»-«атрибут точки» с одновременным сжатием фигуры в направлении какой-либо одной или нескольких осей координат. При этом из GC считываются комплексные коды без некоторых младших разрядов. Например, для плоских фигур

- отсутствие младшего разряда эквивалентно сжатию вдвое по оси абсцисс,
- отсутствие двух младших разрядов эквивалентно сжатию вдвое по обоим осям,
- отсутствие трех младших разрядов эквивалентно сжатию в 4 раза по оси абсцисс и сжатию вдвое по оси ординат, и т.д.

При таком сжатии одной и той же координате может соответствовать несколько атрибутов. Объединение атрибутов (как указывалось) целиком определяется их прикладным значением.

В операции округления все коды векторов округляются (отбрасываются младшие разряды) и переписываются из SSRAM-1 в SSRAM-2. Алгоритм состоит в следующем.

В процессоре PFGAU:

1. Определение номера старшего значащего разряда в SSRAM-1 - операция P6.
2. Обнуление ARAM-2 (в центральном процессоре).
3. Перебор всех q -фрагментов (в центральном процессоре).
 - 3.1. Прием кода MCF_q из SSRAM-1 в GAU - операция P5.
 - 3.2. Перебор всех локальных k -адресов в коде MCF_q (операция P14).

3.2.1. Передача округленного k -вектора из GAU в SSRAM-2 - операция P7.

3.2.2. Передача $((q-1)k)$ -атрибута из ARAM-1 в ARAM-2 (в центральном процессоре). Важно отметить, что возможен такой случай, когда атрибут добавляется к уже существующим атрибутам этой точки.

В процессоре PMGAU:

1. Определение номера старшего значащего разряда в SSRAM-1 - операция R7.
2. Обнуление ARAM-2 (в центральном процессоре).
3. Передача округленного k -вектора из GAU в SSRAM-2 - операция R7.
4. Передача $((q-1)k)$ -атрибута из ARAM-1 в ARAM-2 (в центральном процессоре). Важно отметить, что возможен такой случай, когда атрибут добавляется к уже существующим атрибутам этой точки.

6.7.3. Грубое округление.

Во всех арифметических операциях может возникнуть переполнение, т.е. возникнуть ярусы с номером, превышающим максимальный. Такое переполнение эквивалентно выходу точки за пределы кодируемой области (например, за пределы экрана). При грубом округлении все точки, вышедшие из области определения, исключаются из кода фигуры. Для этого достаточно отбросить старшие разряды кода вектора и обнулить его атрибут. Алгоритм состоит в следующем:

В процессоре PFGAU:

1. Обнуление ARAM-2 (в центральном процессоре).
2. Перебор всех q -фрагментов (в центральном процессоре).
 - 2.1. Прием кода MCF_q из SSRAM-1 в GAU - операция P5.
 - 2.2. Перебор всех локальных k -адресов в коде MCF_q - операция P14.
 - 2.2.1. Анализ длины кода k -вектора в GAU - операция P8.
 - 2.2.2. Если переполнения этого кода нет, то $((q-1)k)$ -атрибут передается из ARAM-1 в ARAM-2 (в центральном процессоре). Иначе он остается равным нулю.

В процессоре PMGAU:

1. Обнуление ARAM-2 (в центральном процессоре).
2. Перебор всех k -адресов в коде MCF - операция R14.
 - 2.1. Анализ длины кода k -вектора в GAU - операция R8.
 - 2.2. Если переполнения этого кода нет, то $((q-1)k)$ -атрибут передается из ARAM-1 в ARAM-2 (в центральном процессоре). Иначе он остается равным нулю.

6.7.4. Коррекция атрибутов.

После округления кода фигуры может быть, что в некотором узле сетки присутствует несколько точек. Это означает, что по некоторому адресу в памяти атрибутов присутствует список атрибутов. Атрибут узла определяется как функция атрибутов всех точек, оказавшихся в этом узле. Эта процедура известна и выполняется в центральном процессоре.

6.7.5. Вычисление атрибутов.

После грубого округления кода фигуры может быть, что в некотором узле сетки отсутствует какая-либо точка – ее атрибут равен нулю. Атрибут узла определяется как функция атрибутов всех соседних узлов. Эта процедура также известна и выполняется в центральном процессоре. Однако при этом необходимо обращаться к со-процессору. Алгоритм состоит в следующем:

1. Сканируется память ARAM-2 (в центральном процессоре).
2. Если по некоторому адресу атрибут равен нулю, то
 - 2.1. Определяются вектор V_0 точки C_0 по данному адресу A_0 – см. операцию P9 (или R9).
 - 2.2. Перебираются координаты и направления по координатам (в центральном процессоре). Для каждого k -варианта
 - 2.2.1. Определяется вектор V_k соседней точки C_k – см. операцию P10 (или R10).
 - 2.2.2. Определяется адрес A_k точки C_k по известному вектору V_k – см. операцию P11 (или R11).
 - 2.2.3. Находится атрибут T_k точки в ARAM-2 по известному адресу A_k (в центральном процессоре).

- 2.3. Определяется и записывается в ARAM-2 атрибут T_0 точки C_0 , как известная функция атрибутов T_k точек C_k (в центральном процессоре).

6.7.6. Кодирование фигуры.

Под этим термином понимается преобразование связанных массивов «атрибуты»-«координаты» в смешанный код фигуры. Алгоритм состоит в следующем:

Производится перебор связанных массивов. Для каждого адреса пары «атрибуты»-«координаты» выполняются следующие действия:

1. Координаты точки преобразуются в код вектора и этот вектор записывается в SSRAM-1. При этом из SSRAM-1 выдается новый адрес. Для этого используется команда P12 (или R12).
2. По этому адресу атрибут точки записывается в ARAM-1 (в центральном процессоре).

6.7.7. Декодирование фигуры.

Под этим термином понимается преобразование смешанного кода фигуры в связанные массивы «атрибуты»-«координаты». Алгоритм состоит в следующем. Производится перебор адресов ARAM-1. Для каждого адреса выполняются следующие действия:

1. По данному адресу атрибут точки переписывается из ARAM-1 в массив «атрибуты» (в центральном процессоре).
2. Код вектора считывается по адресу из SSRAM-1, после чего преобразуется в координаты точки. Для этого используется команда P13 (или R13).
3. Эти координаты записываются в массив «координаты» (в центральном процессоре).

6.8. Операционные блоки

Ниже описываются операционные блоки, входящие в арифметическое устройство и специализированную оперативную память. Эти блоки представляют собой схемы распространения переносов в AGC. Алгоритмы соответствующих операций описаны выше. При описании схем используются следующие обозначения:

π - сигнал входного переноса,

β - триггер разряда β ,

α - триггер разряда α ,

μ, η - сигналы выходных переносов, поступающие в разряды β и α соответственно,

γ - триггер разряда γ ,

δ - триггер разряда δ базисного кода,

τ - триггер сигнала транспонирования τ .

Общая схема распространения переносов представлена на рис. 5.1.2а и рис. 5.2.4. Алгоритмы соответствующих операций описаны выше.

6.8.1. Блок записи номера с данным кодом.

На рис 6.8.1 изображен фрагмент схемы для записи в AGC номера, определенного базисным кодом – см. раздел 5.2.2.1. На этом рисунке изображены блоки, вычисляющие сигналы μ, η по определенным формулам. Один из этих сигналов всегда равен «1» и передается дальше в виде сигнала π . Сигнал $\mu=1$ или $\eta=1$ устанавливает соответствующий триггер в «1». Сигнал $\mu=0$ или $\eta=0$ не изменяет состояния соответствующего триггера.

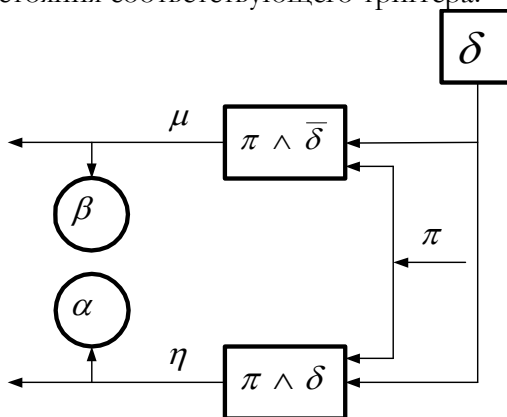


Рис. 6.8.1. Блок записи номера с данным кодом.

6.8.2. Блок записи значения с данным кодом.

На рис 6.8.2 изображен фрагмент схемы для записи в AGC значения с данным базисным кодом - см раздел 5.2.2.2. На этом рисунке изображены блоки, вычисляющие сигналы μ, η по определенным формулам. Один из этих сигналов всегда равен «1» и передается дальше в виде сигнала π . Сигнал $\mu=1$ или $\eta=1$

устанавливает соответствующий триггер в «1». Сигнал $\mu=0$ или $\eta=0$ не изменяет состояния соответствующего триггера.

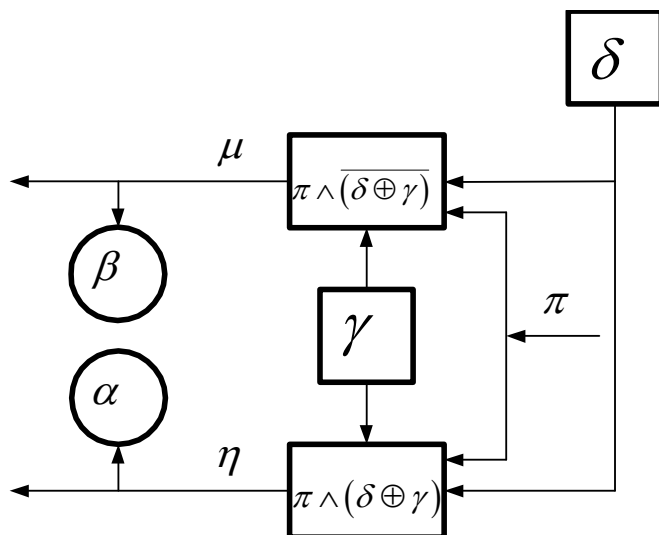


Рис. 6.8.2. Блок записи значения с данным кодом.

6.8.3. Блок чтения значения пути с данным номером.

На рис 6.8.3 изображен фрагмент схемы для чтения значения пути с известным номером с данным базисным кодом – см. раздел 5.2.2.3. Значение пути формируется как второй базисный код с разрядами ω . На представленном рисунке изображены блоки, вычисляющие сигналы μ , η по определенным формулам. Один из этих сигналов всегда равен «1» и передается дальше в виде сигнала π . Блок, вычисляющий сигнал ω записывает его в одноименный триггер регистра кода значения. Сигнал ω вырабатывается в том разряде α или β , через который прошел сигнал переноса.

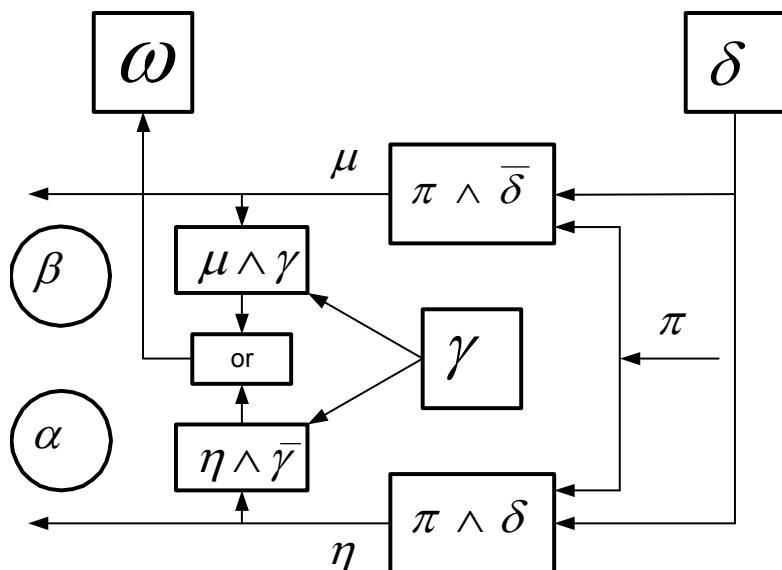


Рис. 6.8.3. Блок чтения значения с данным кодом.

6.8.4. Обратный сумматор

На рис 6.8.4 изображен фрагмент сумматора для обратного сложения AGC с базисным кодом по основанию (-2) – см. раздел 5.2.2.5. На этом рисунке изображены блоки, вычисляющие сигналы μ , η , τ по определенным формулам в соответствии с табл. 5.2.3а. Один из этих сигналов μ , η всегда равен «1» и передается дальше в виде сигнала π . Сигнал τ записывается в одноименный триггер. Сигнал разрешения транспонирования R является общим для всех разрядов и поступает из схемы управления после окончания распространения переносов через все разряды. После этого в разрядах γ устанавливается значение τ .

Эта же схема используется и при умножении. Отличие состоит в том, что сигнал начала сложения обычно подается в корневую вершину, при умножении – во все вершины определенного яруса, в которых $\alpha=0$.

В частном случае при $\delta \equiv 0$ этот же сумматор выполняет функцию инвертора.

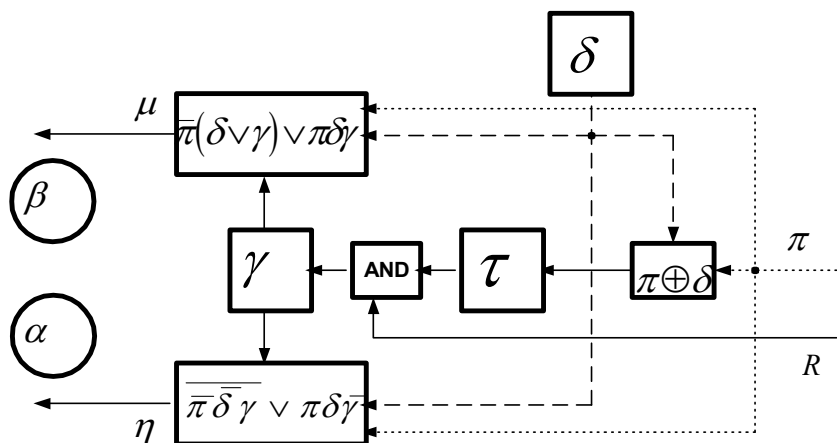


Рис. 6.8.4. Обратный сумматор.

6.8.5. Блок поиска первого открытого пути, его номера и его значения.

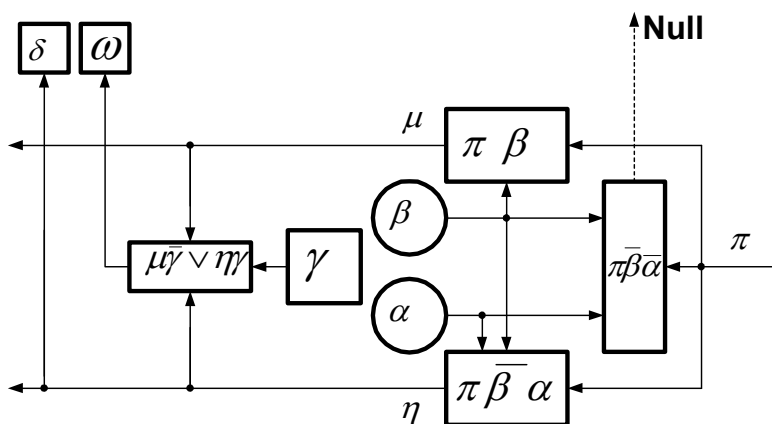


Рис. 6.8.5. Блок поиска первого открытого пути, его номера и его значения.

На рис 6.8.5 изображен фрагмент схемы для поиска первого открытого пути и чтения его номера и его значения. Код номера формируется как первый базисный код с разрядами δ , а код значения формируется как второй базисный код с разрядами ω . На

представленном рисунке изображены блоки, вычисляющие сигналы μ , η по определенным формулам. Только один из этих сигналов может быть равен «1» и передается дальше в виде сигнала π . Если оба этих сигнала равны нулю, то вырабатывается сигнал Null и процесс распространения переносов прекращается. Блоки, вычисляющие сигналы δ и ω записывают их в одноименный триггер регистра соответствующего кода. Эта схема находит самый верхний открытый путь в дереве ГК.

6.8.6. Блок чтения номера и значения пути с данной терминальной вершиной.

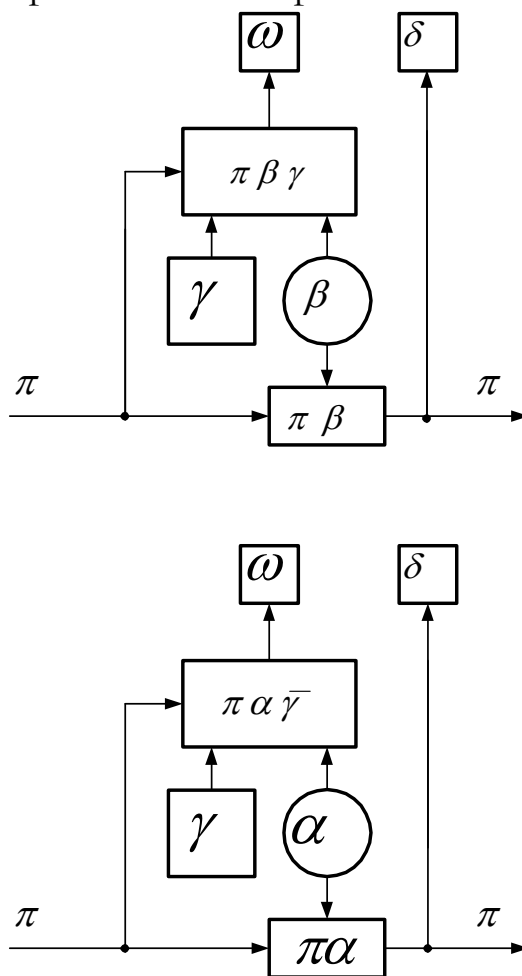


Рис. 6.8.6. Блок чтения номера и значения пути с данной терминальной вершиной.

На рис 6.8.6 изображен фрагмент схемы для чтения номера и значения пути, заканчивающегося в данной терминальной вершине. В отличие от предыдущих схем здесь переносы распространяются слева направо. При этом код номера формируется как первый базисный код с разрядами δ , а код значения формируется как второй базисный код с разрядами ω . Схемы, сопряженные с разрядами β и α , различны.

6.8.7. Блок поиска следующей терминальной вершины.

Этот блок сканирует (сигналами Carry_In и Carry_Out) терминальные вершины, начиная с данной (по сигналу InPut) и до первой вершины с единичным значение. В такой вершине вырабатывается выходной сигнал (OutPut) – см. рис. 6.8.7.

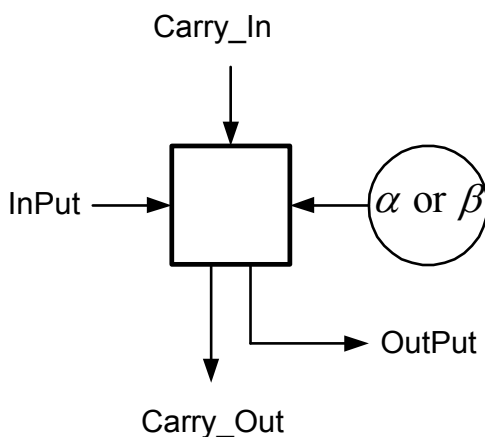


Рис. 6.8.7. Блок поиска следующей терминальной вершины.

Глава 7. Сравнительный анализ

В этом разделе сравниваются характеристики арифметических устройств и блоков оперативной памяти, рассмотренных выше. В табл. 7.1 приведен список устройств, а в табл. 7.2 указаны их характеристики, где

- T - длительность чтения\записи
- S - длительность поиска атрибута по известным координатам
- R – разрядность для оперативной памяти или эквивалентная разрядность для АУ
- A - количество операций для аффинного преобразования
- n - разрядность кода одной координаты
- r - разрядность кода параметра преобразования
- a – общая разрядность всех параметров преобразования – см. (2.1.1)
- p – размерность пространства
- $M=2^{pn}$ – количество точек пространства - см. (2.1.2)
- F – количество ярусов фрагментов при вертикальной фрагментации
- $Q=2^f$ – количество точек во фрагменте при горизонтальной фрагментации
- $D=p(p-1)$ – количество сложений при аффинном преобразовании – см. (2.2.1)
- в п. 1, предполагается, что поиск выполняется в неупорядоченном массиве TRAM

Таблица 7.1. Список сравниваемых устройств.

№		
1	TRAM	Оперативная память в традиционном исполнении
2	PSSRAM	Полная специализированная оперативная память
3	FSSRAM	Специализированная фрагментарная оперативная память
4	SAU	Простейшее арифметическое устройство
5	MSAU	Арифметическое устройство с прямоугольными кодами
6	FSAU	Арифметическое устройство с фрагментарными прямоугольными кодами
7	VAU	Векторное арифметическое устройство
8	MVAU	Векторное арифметическое устройство с прямоугольными кодами
9	FVAU	Векторное арифметическое устройство с фрагментарными прямоугольными кодами
10	FGAU	Арифметическое устройство с фрагментарными геометрическими кодами
11	MGAU	Арифметическое устройство с геометрическими кодами, совмещенное с оперативной памятью

Таблица 7.2. Характеристики сравниваемых устройств.

№		T	S	R	A
1	TRAM	1	$M/2$	$Mp(n+r)$	-
2	PSSRAM	1	1	$Mpr+M$	-
3	FSSRAM	F	F	$F\left(\frac{M}{F}pr + \sqrt{M}\right)$	-
4	SAU	-	-	$7(n+r)+a$	$M(D+p^2)$
5	MSAU	-	-	$7M(n+r)+a$	$D+p^2$
6	FSAU	-	-	$7Q(n+r)+a$	$(D+p^2)M/Q$
7	VAU	-	-	$7p(n+r)+a$	M
8	MVAU	-	-	$7Mp(n+r)+a$	1
9	FVAU	-	-	$7Qp(n+r)+a$	M/Q
10	FGAU	-	-	$((pn-f)+Qpr+Q)$	M/Q
11	MGAU	1	1	$(Mpr+M)$	1

Длительность τ одной операции практически не зависит от типа каждого из перечисленных устройств. Поэтому длительность аффинного преобразования в каждом из этих устройств равна $t = A \tau$, За единицу времени каждое устройство решает $z = 1/t = 1/A \tau$ задач аффинного преобразования.

Естественно характеризовать *качество* арифметического устройства объемом устройства, решающего определенное число задач аффинного преобразования, или *относительным объемом* устройства: чем меньше этот относительный объем W , тем выше качество устройства. Очевидно, относительным объемом арифметического устройства $W \equiv R/z$ или $W = AR$.

Аналогично, качество оперативной памяти можно характеризовать ее объемом, отнесенным к количеству операций доступа к памяти, выполняемых в единицу времени. Если рассматривать операцию чтения\записи, то относительный объем оперативной памяти $W1 = TR$. Если же рассматривать операцию поиска точки с данными координатами в неупорядоченном массиве, то относительный объем оперативной памяти $W2 = SR$. Целесообразно также рассматривать заданную смесь этих операций, но для этого должна быть известна статистика операций доступа к памяти.

В табл. 7.3. указан относительный объем всех рассмотренных выше устройств.

Таблица 7.3. Относительный объем сравниваемых устройств.

№		<i>AR</i>	<i>SR</i>
1	TRAM		$\frac{M^2 p (n + r)}{2}$
2	PSSRAM		$F^2 \left(\frac{M}{F} pr + \sqrt[n]{M} \right)$
3	FSSRAM		<i>Mpr</i>
4	SAU	$14 Mp^2 (n + r)$	
5	MSAU		
6	FSAU		
7	VAU	$7Mp(n+r)$	
8	MVAU		
9	FVAU		
10	FGAU	<i>Mpr</i>	
11	MGAU	<i>Mpr</i>	<i>Mpr</i>

На основе этой таблицы построена более наглядная табл. 7.4. относительного объема основных устройств.

Таблица 7.4. Относительный объем основных устройств.

№	Устройство	$W=AR$ для процессора; $W1=TR$ или $W2=SR$ для оперативной памяти
1	Обычная память	$W_1 = Mp (n + r)$ $W_2 = M^2 p (n + r) / 2$
3	Специальная память	$W_1 = W_2 =$ $F^2 \left(\frac{M}{F} pr + \sqrt[n]{M} \right)$
5	Скалярный процессор	$14 Mp^2 (n + r)$
8	Векторный процессор	$7Mp(n+r)$
10	Геометрический процессор	<i>Mpr</i>

На рис. 1.1 (во введении) приведена гистограмма качества рассмотренных арифметических устройств при $n=r$. Единицей измерения на гистограмме является величина $14 * M$. Например, при $p=3$ величины качества рассмотренных арифметических устройств относятся как (84:14:1).

Относительный объем W_2 устройства TRAM при $n=r$ в M раз превышает относительный объем W_2 устройства PSSRAM. При $F>1$ относительные объемы W_2 устройств TRAM и FSSRAM относятся как $\frac{M(n+r)}{2Fr}$. Например, при $n=r$ эти объемы относятся как M/F .

Относительный объем W_1 устройства TRAM при $n=r$ в 2 раза превышает относительный объем W_1 устройства PSSRAM. При $F>1$ относительные объемы W_1 устройств TRAM и FSSRAM относятся как $\frac{(n+r)}{Fr}$. Например, при $n=r$ эти объемы относятся

как $2/F$, т.е. относительный объем W_1 устройства TRAM меньше в $F/2$ раз относительного объема W_1 устройства FSSRAM.

Предположим теперь, что в данной задаче операции чтения\записи встречаются в H раз чаще, чем операции поиска. Тогда относительный объем оперативной памяти должен определяться по формуле $W = R(TH + S)$. Эта величина может быть найдена из табл. 7.2. Для устройств TRAM и FSSRAM относительный объем равен соответственно

$$W_T = Mp(n+r)(H + M/2)$$

и

$$W_F = F \left(\frac{M}{F} pr + \sqrt[3]{M} \right) (FH + F) \approx FMpr (H + 1).$$

$$\text{Отношение } \frac{W_F}{W_T} \approx \frac{FMpr (H + 1)}{Mp(n+r)(H + M/2)}.$$

При $1 \ll H \ll M$, $n = r$ это отношение $\frac{W_F}{W_T} \approx \frac{HF}{M}$. Таким

образом, относительный объем FSSRAM меньше относительного

объема TRAM, если $\frac{W_F}{W_T} \approx \frac{HF}{M} < 1$ или $HF < M$. Среднее

число ярусов при вертикальной фрагментации специализированной оперативной памяти $F \approx 10$. Следовательно,

относительный объем специализированного запоминающего устройства меньше относительного объема традиционного запоминающего устройства в $\frac{M}{10 H}$ раз.

Обозначения

Add - сумматор М-кодов

AGC – атрибутный геометрический код (attributic geometrical code)

AGCC – атрибутный геометрический комплексный код (attributic geometrical complex code)

ARAM - традиционная оперативная память атрибутов

AU – арифметическое устройство (arithmetic unit)

CAGC – сокращенный атрибутный геометрический код (contracted attributic geometrical code)

CoderPM - кодер положительного Р-кода в М-код

С-код – комплексный код по комплексному основанию

DecoderMP - декодер Р-кода в М-код

Deven – одноразрядная схема декодирования для разряда с четным номером

Dodd - одноразрядная схема декодирования для разряда с нечетным номером

DRAM - динамическая оперативная память (dynamic random access memory)

FGAU – фрагментарное геометрическое арифметическое устройство (fragmentntary geometrical arithmetic unit)

FSAU – фрагментарное скалярное арифметическое устройство (fragmentntary scalar arithmetic unit)

FSSRAM – фрагментарная специализированная статическая оперативная память (fragmentntary specialized static random access memory)

FVAU – фрагментарное векторное арифметическое устройство (fragmentntary vectorial arithmetic unit)

GAU –геометрическое арифметическое устройство (geometrical arithmetic unit)

GC - геометрический код (geometrical code)

Inv - инвертор М-кода

InvAdd - инверсный сумматор М-кодов

LP - линейная часть кода MCF

- MCF** - смешанный код фигуры (mixed code of figure)
- mDecoderMP** - полный декодер Р-кода в М-код
- Meven** – одноразрядная схема кодирования для разряда с четным номером
- MGAU** – максимальное геометрическое арифметическое устройство (maximum geometrical arithmetic unit)
- Modd** - одноразрядная схема кодирования для разряда с нечетным номером
- MSAU** – максимальное скалярное арифметическое устройство (maximum scalar arithmetic unit)
- MVAU** – максимальное векторное арифметическое устройство (maximum vectorial arithmetic unit)
- М-код** – код действительных чисел по основанию «-2»
- nSign** - знакоопределитель М-кода
- Partitioning** - распределитель частей кода
- PFGAU** - процессор с фрагментарным арифметическим устройством FGАU
- PGC** – первичный геометрический код (primary geometrical code)
- PMGAU** - процессор с арифметическим устройством MGAU
- PreCoder** - прекодер Р-кода в М-код
- PSSRAM** – полная специализированная статическая оперативная память (perfect specialized static random access memory)
- Р-код** – традиционный прямой код по основанию «2»
- RAM** - оперативная память (random access memory)
- RCS** - прямоугольный код чисел (rectangular code of scalars)
- RCV** - прямоугольный код векторов (rectangular code of vectors)
- RGP** – растровый геометрический процессор (raster geometrical processor)
- SAU** - скалярное арифметическое устройство (scalar arithmetic unit)
- Seven** – одноразрядная схема знакоопределителя для разряда с четным номером
- Sodd** - одноразрядная схема знакоопределителя для разряда с нечетным номером
- SRAM** - статическая оперативная память (static random access memory)
- SSRAM** - специализированная статическая оперативная память (specialized static random access memory)

Sub – вычитатель М-кодов

TRAM - традиционная оперативная память (traditional random access memory)

VAU - векторное арифметическое устройство (vectorial arithmetic unit)

Список примеров

Пример 5.1.1 сложения при $\rho=2 \setminus 74$

Пример 5.1.2 обратного сложения при $\rho=-2 \setminus 77$

Пример 5.1.3 умножения при $\rho=-2 \setminus 80$

Пример 5.1.3а умножения при $\rho=-2 \setminus 83$

Пример 5.1.4 обратного сложения при $\rho = j\sqrt{2} \setminus 85$

Пример 5.1.5 умножения мнимой части GC при

$$\rho = j\sqrt{2} \setminus 87$$

Пример 5.1.6 кодирования плоскости при $\rho = j\sqrt{2} \setminus 91$

Пример 5.1.7 центроаффинного преобразования при

$$\rho = j\sqrt{2} \setminus 95$$

Пример 5.2.1 сложения при $\rho=2 \setminus 104$

Пример 5.2.2 обратного сложения при $\rho=-2 \setminus 105$

Пример 5.2.3 центроаффинного преобразования при

$$\rho = j\sqrt{2} \setminus 112$$

Список таблиц

- Таблица 3.1.1. Системы кодирования комплексных чисел \ 24
- Таблица 3.1.2. Двоичные системы кодирования \ 24
- Таблица 3.2.1. Умножение трехмерных векторов \ 25
- Таблица 3.2.2. Умножение комплексных чисел \ 28
- Таблица 3.2.3. Умножение четырехмерных векторов \ 28
- Таблица 3.4.2. Одноразрядная схема инвертирования \ 34
- Таблица 3.4.3. Одноразрядная схема инверсного суммирования \ 35
- Таблица 3.4.4. Одноразрядная схема суммирования \ 36
- Таблица 3.4.5. Одноразрядная схема вычитания \ 37
- Таблица 3.4.6.1. Одноразрядная схема знакоопределителя для четного разряда \ 38
- Таблица 3.4.6.2. Одноразрядная схема знакоопределителя для нечетного разряда \ 39
- Таблица 3.6.1. Одноразрядное скалярное умножение \ 45
- Таблица 3.6.2. Одноразрядное векторное умножение \ 46
- Таблица 3.6.3. Переносы при скалярном умножении \ 48
- Таблица 3.6.4. Переносы при скалярном умножении \ 48
- Таблица 3.6.5. Переносы при векторном умножении \ 50
- Таблица 3.7.5.1. Одноразрядная схема кодера для четного разряда \ 56
- Таблица 3.7.5.2. Одноразрядная схема кодера для нечетного разряда \ 56
- Таблица 3.7.6.1. Одноразрядная схема декодера для четного разряда \ 58
- Таблица 3.7.6.2. Одноразрядная схема декодера для нечетного разряда \ 59
- Таблица 3.7.8. Прекодер Р-кода в М-код \ 60
- Таблица 3.7.9. Распределитель частей кода \ 60
- Таблица 4.2.1. Сравнительные характеристики АУ \ 65
- Таблица 4.2.2. Числовые характеристики АУ при $p=2$, $r=6$, $M=10^6$, $n=12$, $Q=256$, $a=72$ \ 66

- Таблица 4.2.3. Числовые характеристики АУ при $p=3$, $r=6$,
 $M=10^6$, $n=12$, $Q=256$, $a=90 \setminus 66$
- Таблица 4.2.4. Числовые характеристики АУ при $p=4$, $r=6$,
 $M=10^6$, $n=12$, $Q=256$, $a=240 \setminus 66$
- Таблица 5.1.1a. Сложение геометрического и базисного кодов
 при $p=2 \setminus 74$
- Таблица 5.1.1b. Обратное сложение ГС с базисным кодом при
 $p=-2 \setminus 76$
- Таблица 5.1.2. Геометрический код плоскости при
 $\rho = j\sqrt{2} \setminus 92$
- Таблица 5.1.2a. Геометрический код с выделенными точками
 плоскости при $\rho = j\sqrt{2} \setminus 92$
- Таблица 5.1.3. Центроаффинное преобразование фигуры при
 $\rho = j\sqrt{2} \setminus 96$
- Таблица 5.2.1. Запись значения с данным кодом $\setminus 102$
- Таблица 5.2.2. Чтение значения пути с данным номером $\setminus 103$
- Таблица 5.2.3. Сложение АГС с базисным кодом при $p=2 \setminus 103$
- Таблица 5.2.3a. Обратное сложение АГС с базисным кодом при
 $p=-2 \setminus 105$
- Таблица 5.2.4. Инвертирование АГС при $p=-2 \setminus 107$
- Таблица 7.1. Список сравниваемых устройств $\setminus 148$
- Таблица 7.2. Характеристики сравниваемых устройств $\setminus 148$
- Таблица 7.3. Относительный объем сравниваемых
 устройств $\setminus 150$
- Таблица 7.4. Относительный объем основных устройств $\setminus 150$

Список рисунков

- Рис. 1.1. Гистограмма относительного объема арифметических устройств \ 13
- Рис. 2.2.1. Простейшее арифметическое устройство \ 18
- Рис. 2.2.2. Арифметическое устройство с фрагментарными прямоугольными кодами \ 22
- Рис. 3.4.1. Многоразрядная схема алгебраического сложения \ 33
- Рис. 3.4.2. Одноразрядная схема инвертирования \ 34
- Рис. 3.4.3. Одноразрядная схема инверсного сумматора \ 35
- Рис. 3.4.4. Одноразрядная схема сумматора \ 36
- Рис. 3.4.5. Одноразрядная схема вычитателя \ 37
- Рис. 3.4.6.1. Одноразрядная схема знакоопределителя \ 38
- Рис. 3.4.6.2. Знакоопределитель \ 39
- Рис. 3.6.1. Сумматор в блоке скалярного умножения \ 49
- Рис. 3.6.2. Сумматор в блоке векторного умножения \ 51
- Рис. 3.7.5.1. Кодер положительного Р-кода в М-код \ 55
- Рис. 3.7.5.2. Одноразрядная схема кодера \ 55
- Рис. 3.7.6.1. Декодер М-кода в Р-код \ 57
- Рис. 3.7.6.2. Одноразрядная схема декодера \ 58
- Рис. 3.7.7. Полный декодер \ 59
- Рис. 4.1.1. Векторное арифметическое устройство \ 62
- Рис. 4.1.2. Векторное арифметическое устройство с прямоугольными кодами \ 64
- Рис. 5.1.1. Дерево геометрического кода \ 68
- Рис. 5.1.2. Пример. Дерево бинарных разрядов \ 70
- Рис. 5.1.2а. Схема распространения переносов в GC \ 72
- Рис. 5.1.3. Пример. Транспонированный код \ 73
- Рис. 5.1.4. К примеру 5.1.1 \ 75
- Рис. 5.1.5. К примеру 5.1.2 \ 78
- Рис. 5.1.6. К примеру 5.1.3 \ 80
- Рис. 5.1.7. К примеру 5.1.3 \ 81
- Рис. 5.1.8. К примеру 5.1.3 \ 81
- Рис. 5.1.9. К примеру 5.1.3 \ 81
- Рис. 5.1.9а. К примеру 5.1.3а \ 83
- Рис. 5.1.9b. К примеру 5.1.3а \ 83
- Рис. 5.1.10. К примеру 5.1.4 \ 86

- Рис. 5.1.11. К примеру 5.1.5 \ 88
- Рис. 5.1.12. К примеру 5.1.5 \ 89
- Рис. 5.1.13. К примеру 5.1.5 \ 90
- Рис. 5.1.14. Кодирование плоской фигуры \ 91
- Рис. 5.1.15. Кодирование плоскости при $y=3$, $m=-1$, $r=4$ для примера 5.1.6 \ 92
- Рис 5.1.15a. Пример: плоская фигура \ 94
- Рис 5.1.15b. Пример: дерево GC плоской фигуры \ 94
- Рис. 5.1.16. Центроаффинное преобразование фигуры для примера 5.1.7 \ 96
- Рис. 5.2.1. Атрибутный геометрический код \ 101
- Рис. 5.2.2. К примеру 5.2.1 \ 104
- Рис. 5.2.3. К примеру 5.2.2 \ 106
- Рис. 5.2.3a. Одноразрядная схема обратного сложения \ 107
- Рис. 5.2.4. Схема распространения переносов в комплексном GC \ 111
- Рис. 5.2.5. К примеру 5.2.3: AGC исходной фигуры \ 114
- Рис. 5.2.6. К примеру 5.2.3: AGC деформированной фигуры \ 115
- Рис. 5.2.7. Сокращенный AGC \ 119
- Рис. 6.0.1. Структура MCF \ 123
- Рис. 6.1.1. Полная специализированная оперативная память \ 124
- Рис. 6.2.1. Ярусы геометрического кода и сегменты линейного кода \ 125
- Рис. 6.4.1. Фрагментарное арифметическое устройство \ 130
- Рис. 6.5.1. Процессор с максимальным арифметическим устройством \ 132
- Рис 6.6.1. Процессор с фрагментарным арифметическим устройством \ 135
- Рис. 6.8.1. Блок записи номера с данным кодом \ 141
- Рис. 6.8.2. Блок записи значения с данным кодом \ 142
- Рис. 6.8.3. Блок чтения значения с данным кодом \ 143
- Рис. 6.8.4. Обратный сумматор \ 144
- Рис. 6.8.5. Блок поиска первого открытого пути, его номера и его значения \ 145
- Рис. 6.8.6. Блок чтения номера и значения пути с данной терминальной вершиной \ 146
- Рис. 6.8.7. Блок поиска следующей терминальной вершины \ 147

Часть 1. Коды функций

Аннотация

В этой части описываются малоизвестные методы построения специализированных компьютеров для обработки функций. Такие процессоры могут найти применение для медицины, метеорологии, сейсмологии, радиоастрономии, физике, противовоздушной обороне и т.п.

Описывается теория кодирования функций одного и многих аргументов – структура кодов, алгоритмы кодирования, декодирования, арифметические операции. Теория дополняется многочисленными примерами. Рассматривается устройство функционального процессора – представление данных, операционные блоки, техническая реализация алгоритмов кодирования, декодирования и арифметических операций. Оценивается быстродействие этого процессора.

Содержание

Предисловие \ 4-4

Глава 1. Позиционные коды функций \ 4-6

Глава 2. Кодирование тригонометрических рядов \ 4-23

Глава 3. Кодирование функций многих аргументов \ 4-56

Глава 4. Четверичные тригонометрические треугольные коды \ 4-23

Глава 5. Арифметическое устройство для операций с функциями \ 4-95

Обозначения \ 4-118

Предисловие

В этой части книги описывается компьютер, оперирующий с функциями. Впервые вычислительная машина для операций с функциями была предложена и разработана М.А. Карцевым в 1967 году [55]. В число операций этой вычислительной машины входили «сложение, вычитание и умножение функций, сравнение функций, аналогичные операции над функцией и числом, отыскание максимума функций, вычисление неопределенного интеграла, вычисление определенного интеграла от производной двух функций, сдвиг функции по абсциссе и т.д.» По архитектуре эта вычислительная машина являлась (пользуясь современной терминологией) векторным процессором. В ней использовался тот факт, что «многие из этих операций могут быть истолкованы как известные операции над векторами: сложение и вычитание функций - как сложение и вычитание векторов, вычисление определенного интеграла от производной двух функций - как вычисление скалярного произведения двух векторов, сдвиг функций по абсциссе - как поворот вектора относительно осей координат и т.д.».

В отличие от этого предлагаемый компьютер основан на представлении функции единым двоичным кодом. В нем указанные операции с функциями (и, кроме того, дифференцирование и интегрирование функций) выполняются как уникальные машинные операции с такими кодами на единственном арифметическом устройстве. При этом объем данных сокращается, а конструкция процессора резко упрощается: нет задачи организации параллельных вычислений на множестве скалярных арифметических устройств. Однако вместо множества простых скалярных арифметических устройств в предлагаемом компьютере появляется сложное арифметическое устройство для операций с кодами функций – функциональное арифметическое устройство.

Итак, для операций с функциями можно предложить

- компьютер Т – традиционный компьютер, содержащий единственное скалярное арифметическое устройство,
- компьютер Р – векторный компьютер, содержащий несколько скалярных арифметических устройств,
- компьютер F – компьютер, содержащий функциональное арифметическое устройство.

Ниже показано, что по сравнению с компьютером Т у компьютера Р быстродействие растет пропорционально объему, а у компьютера F быстродействие увеличивается в 8 раз быстрее увеличения объема.

Теория предлагаемых компьютеров была предложена и разработана автором в работах [12, 23-25, 32]. Важно отметить, что в программировании для предлагаемых компьютеров используется существующий математический аппарат, не учитывающий, естественно, специфических возможностей этих компьютеров. Можно надеяться, что при распространении таких компьютеров будут найдены не только другие методы решения задач, но и другие неожиданные области применения, как это непрерывно происходит с существующими компьютерами.

Эта часть книги содержит 5 глав. В **первой главе** описываются позиционные коды функций общего вида – структура кодов и алгоритмы операций с ними. Во **второй главе** детально рассматриваются коды тригонометрических рядов. Этому вопросу уделяется особое внимание в связи с тем, что тригонометрические ряды Фурье накладывают наименьшие ограничения на вид функции (которые представляются этими рядами), а операции с рядами Фурье очень широко используются в прикладных задачах. В **третьей главе** рассматриваются позиционные коды функций многих аргументов. В **четвертой главе** подробно рассматриваются алгоритмы операций с четверичными тригонометрическими треугольными кодами, описанные в общем случае в предыдущих главах. Целью этой главы является конкретизация теории кодирования функций до такой степени, которая допускает постановку задачи технического проектирования соответствующих компьютеров. В **пятой главе** рассматривается конструкция и система команд одного варианта арифметического устройства для операций с кодами функций. В этой главе производится также сравнительный анализ, на который мы уже ссылались выше.

Глава 1. Позиционные коды функций

1. Треугольные коды
2. Алгебраическое сложение кодов вещественных чисел
3. Алгебраическое сложение треугольных кодов
4. Деление треугольных кодов на параметр
5. Умножение треугольных кодов
6. Кодирование и декодирование треугольных кодов
7. Дифференцирование треугольных кодов
8. Ступенчатые коды

1. Треугольные коды

Ниже предполагается, что функция определена функциональным рядом. При этом тривиальный способ кодирования функций мог бы заключаться в задании коэффициентов этих рядов. Однако такой способ создает коды большого объема и не эффективен для умножения.

В дальнейшем для общности будем полагать, что функция

$\Phi(x)$ задана отношением $\Phi(x) = h \frac{f_1(x)}{f_2(x)}$, где $f_1(x)$ и $f_2(x)$ -

функции, представленные рядами или их кодами, h - числовой множитель — *экспонента функции*. Такое задание значительно расширяет класс представимых функций и позволяет избежать операции деления. Экспонента h снимает ограничения с величины коэффициентов ряда, которые возникают при кодировании функций. Дадим некоторые определения.

Определение 1.1.1. Двойная сумма вида

$$F(x) = \sum_{k=0}^n \sum_{m=0}^k \alpha_{mk} R^k y^{k-m} (1-y)^m, \quad (1.1.1)$$

где

α_{mk} - действительные числа,

R - целое положительное число,

$y=f(x)$ - некоторая функция аргумента x ,

m, k, n - целые положительные числа или нули

называется разложением функции $F(x)$ по основанию y с параметром R ; в этом разложении

- функция $\varphi = \alpha_{mk} R^k y^{k-m} (1-y)^m$ называется mk -разрядом;
- функция $\psi = R^k y^{k-m} (1-y)^m$ называется весом mk -разряда,
- число α_{mk} называется величиной mk -разряда.

Таким образом,

$$F(x) = \sum_{k=0}^n \sum_{m=0}^k \alpha_{mk} \psi_{mk} \quad (1.1.2)$$

или

$$F(x) = \sum_{k=0}^n \sum_{m=0}^k \varphi_{mk} \quad (1.1.3)$$

При известном x и, следовательно, известном y значение функции $F(x)$ может быть вычислено непосредственно по (1.1.1). Однако для такого вычисления эта формула может быть представлена в более удобном виде:

$$F = (((((c_n R y + c_{n-1}) R y + c_{n-2}) R y + \dots) R y + \dots + c_1) R y + c_0),$$

где

$$c_k = (((((\alpha_{kk} b + \alpha_{k-1,k}) b + \alpha_{k-2,k}) b + \dots) b + \dots + \alpha_{1,k}) b + \alpha_{0,k}),$$

$$b = (1-y)/y.$$

Определение 1.1.2. Треугольная матрица, составленная из величин α_{mk} разложения функции $F(x)$ по основанию y с параметром R таким образом, что каждая величина α_{mk} принадлежит k -столбцу и m -строке этой матрицы, называется треугольным кодом функции $F(x)$ по основанию y с параметром R ; этот треугольный код обозначается символами $TK(F(x))$, где обозначение функции может быть опущено, если по контексту ясно, о какой функции идет речь. Таким образом,

$$\begin{array}{cccccc} & & & & & \alpha_{nn} \\ & & & & \dots & \dots \\ & & & \alpha_{kk} & \dots & \alpha_{kn} \\ & & \dots & \dots & \dots & \dots \\ & \alpha_{mm} & \dots & \alpha_{mk} & \dots & \alpha_{mn} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \alpha_{11} & \dots & \alpha_{1m} & \dots & \alpha_{1k} & \dots & \alpha_{1n} \\ \text{TK}(F(x)) = \alpha_{00} & \alpha_{01} & \dots & \alpha_{0m} & \dots & \alpha_{0k} & \dots & \alpha_{0n} \end{array}$$

Вес разрядов ТК зависит от его местоположения. В этом смысле треугольные коды функций можно называть также позиционными кодами.

Определение 1.1.3. Треугольный код называется R-ым и обозначается символом TK_R , если величины α_{mk} принимают значения из множества $D_R = \{-r_1, -r_1 + 1, \dots, -1, 0, 1, \dots, r_2 - 1, r_2\}$, где $R = r_1 + r_2 + 1$, $r_1 > 0$, $r_2 > 0$.

Например, треугольный код называется троичным, если $\alpha_{mk} \in \{-1, 0, 1\}$, и - четверичным, если $\alpha_{mk} \in \{-2, -1, 0, 1\}$. Будем также использовать следующие написания ТК:

$$\begin{array}{ccccc} & & & & \text{A} \\ & & & & \text{B} \text{ C} \\ \text{TK} = (m) & \text{D} & \text{E} & \text{F} \\ & & & & (k) \end{array}$$

или

$$\begin{array}{ccccc} & & & & \text{A} \\ & & & & * \text{ } * \\ \text{TK} = \text{D} & * & \text{F} \end{array}$$

где в скобках указаны (в случае необходимости) число отброшенных нижних строк (m) и левых столбцов (k), а номера разрядов не обозначены. Разряды, не участвующие в данном преобразовании, обозначаются как *.

Рассмотрим некоторые свойства ТК.

Свойство 1.1.1. Треугольные коды mk -разряда и веса mk -разряда имеют соответственно вид

$$\begin{array}{ccccc} & & & 0 & \\ & & & 0 & 0 \\ \text{TK}(\varphi_{mk})=(m) & \alpha_{mk} & 0 & 0 & \\ & (k) & & & \end{array}$$

и

$$\begin{array}{ccccc} & & & 0 & \\ & & & 0 & 0 \\ \text{TK}(\psi_{mk})=(m) & 1 & 0 & 0 & \\ & (k) & & & \end{array}$$

Свойство 1.1.2. Преобразование кода

$$\begin{array}{ccccc} & & & A & \\ & & & B & C \\ \text{TK}(f(x))=(p) & D & E & F & \\ & (q) & & & \end{array}$$

в код

$$\begin{array}{ccccc} & & & A & \\ & & & B & C \\ \text{TK}(h(x))=(p+m) & D & E & F & \\ & (q+k) & & & \end{array}$$

называемое *mk*-сдвигом, соответствует следующей операции с кодируемыми функциями: $h(x) = f(x)R^k y^{k-m}(1-y)^m$; в частности, если $k > 0$ и $m > 0$, то $h(x) = f(x) \cdot \psi_{mk}$, то-есть *mk*-сдвиг соответствует умножению ТК на вес *mk*-разряда. Справедливо также обратное утверждение. Аналогично, если $k < 0$ и $m < 0$, то $h(x) = f(x)/\psi_{mk}$, то-есть *mk*-сдвиг соответствует делению ТК на вес *mk*-разряда.

Свойство 1.1.3. Умножение на число, сложение и вычитание ТК суть аналогичные операции с каждым разрядом или с одноименными разрядами ТК.

Свойство 1.1.4. Число разрядов ТК, имеющего $(n+1)$ столбцов, $s = (n+1)(n+2)/2$.

Свойство 1.1.5. Из очевидной формулы

$$R\psi_{mk} = \psi_{m,k+1} + \psi_{m+1,k+1}$$

следует, что

$$\begin{bmatrix} 0 \\ Ra & 0 \end{bmatrix} = \begin{bmatrix} a \\ 0 & a \end{bmatrix}, \quad \begin{bmatrix} a \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & \\ Ra & -a \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 0 & a \end{bmatrix} = \begin{bmatrix} -a \\ Ra & 0 \end{bmatrix}.$$

2. Алгебраическое сложение кодов вещественных чисел.

Вначале рассмотрим алгебраическое сложение кодов $K_R(a)$ вещественных чисел a по основанию R с разрядами, принимающими положительное и отрицательное значения из множества D_R , содержащего R чисел. Такие коды составляют прямоугольный код функции, а алгоритм их алгебраического сложения служит (как будет ясно из дальнейшего) аналогией алгебраического сложения треугольных кодов.

Алгебраическое сложение пары одноименных разрядов кодов чисел описывается формулой

$$S_{mk} = Q_{mk} + \pi'_{mk}, \quad (1.2.1)$$

причем

$$\left. \begin{aligned} Q_{mk} &= \alpha_{mk} + \beta_{mk} \text{ при сложении,} \\ Q_{mk} &= \alpha_{mk} - \beta_{mk} \text{ при вычитании,} \\ Q_{mk} &= -\alpha_{mk} - \beta_{mk} \text{ при обратном сложении,} \\ Q_{mk} &= -\alpha_{mk} \text{ при инвертировании,} \\ Q_{mk} &= R\alpha_{mk} \text{ при умножении на параметр,} \end{aligned} \right\} \quad (1.2.2)$$

где S_{mk} - разрядный результат,
 π'_{mk} - перенос из предыдущего разряда,
 Q_{mk} - частный разрядный результат,
 α_{mk}, β_{mk} - разряды слагаемых кодов.

Представим разрядный результат в виде

$$S_{mk} = \sigma_{mk} + R\pi_{mk}, \quad (1.2.3)$$

где

σ_{mk} - разряд результирующего кода,
 π_{mk} - перенос из данного разряда.

Из изложенного следует, что алгоритм алгебраического сложения R -ых треугольных кодов чисел существует, если

- переносы π_{mk}, π'_{mk} принимают значения из общего ограниченного множества P ,
- любая сумма вида (1.2.1) представима также в виде (1.2.3), где $\sigma_{mk} \in D_R$.

Приведенные формулы позволяют построить таблицы, описывающие процесс одноразрядного алгебраического сложения. В применении к синтезу таких таблиц вышеприведенные условия трансформируются в условия полноты таблицы:

- если π_{mk} принимает в таблице некоторое значение, то π'_{mk} также принимает это значение,
- сумма S_{mk} любой комбинации чисел α_{mk} , β_{mk} , π'_{mk} присутствует в таблице.

В качестве примера приведена табл. 1.2.1, описывающая одноразрядное сложение при $R=3$ и $D_R = \{-1, 0, 1\}$. Нетрудно убедиться, что эта таблица удовлетворяет условиям полноты.

Таблица 1.2.1.

S_{mk}	$K_R(S_{mk})$	σ_{mk}	π_{mk}
0	0	0	0
1	1	1	0
2	1	-1	1
3	1	0	1
-1	-1	-1	0
-2	-1	1	-1
-3	-1	0	-1

3. Алгебраическое сложение треугольных кодов

Алгебраическое сложение ТК (сложение, обратное сложение, инвертирование, умножение на параметр R) связано с выполнением одноименной операции над парой одноименных разрядов слагаемых кодов или над каждым разрядом кода, умножаемого на (-1) или R . В свойстве 1.1.3 отмечено, что эти операции выполняются чрезвычайно просто, если на величины α_{mk} не накладывается ограничений. Однако для возможности использования ТК в вычислительных устройствах и упрощения технической реализации операций с ними необходимо, чтобы эти величины принимали значения из ограниченного множества. Этому требованию удовлетворяют TK_R , описанные в определении 1.1.3. Ограничения, накладываемые на величины α_{mk} , естественно, вызывают усложнение алгоритмов операций, что связано, как и в обычных кодах, с возникновением переносов в старшие разряды при получении в данном разряде величины $\alpha_{mk} \notin D_R$. Однако, если в обычных R -ых позиционных кодах выработка переносов основывается на соотношении $K(R)=10$, то в данном случае используется соотношение

$$TK_R(R) = 0 \quad 1 \quad (1.3.1)$$

Таким образом, в треугольных кодах перенос распространяется в два разряда старшего столбца (перенос-'вилка'). Следовательно, в данный разряд могут поступить переносы также из двух младших разрядов. Поэтому алгебраическое сложение пары одноименных разрядов кодов описывается формулой

$$S_{mk} = Q_{mk} + \pi'_{mk} + \pi''_{mk}, \quad (1.3.1)$$

причем Q_{mk} определяется по (1.2.2). Представим разрядный результат в виде (1.2.3). Тогда получим:

$$TK(S_{mk}) = \sigma_{mk} \begin{matrix} \pi_{mk} \\ \pi_{mk} \end{matrix},$$

где

σ_{mk} - разряд результирующего кода,

π_{mk} - перенос из данного разряда.

Из изложенного следует, что алгоритм алгебраического сложения R -ых треугольных кодов существует, если

- переносы π_{mk} , π'_{mk} , π''_{mk} принимают значения из общего ограниченного множества \mathbf{P} ,
- любая сумма вида (1.3.1) представима также в виде (1.3.2), где $\sigma_{mk} \in D_R$.

Нетрудно убедиться, что эти условия выполняются при $R > 2$. Приведенные формулы позволяют построить таблицы, описывающие процесс одноразрядного алгебраического сложения. В применении к синтезу таких таблиц вышеприведенные условия трансформируются в условия полноты таблицы:

- если π_{mk} принимает в таблице некоторое значение, то π'_{mk} и π''_{mk} также принимают это значение,
- сумма S_{mk} любой комбинации чисел α_{mk} , β_{mk} , π'_{mk} и π''_{mk} присутствует в таблице.

В качестве примера приведена табл. 1.3.1, описывающая одноразрядное сложение при $R=3$. Нетрудно убедиться, что эта таблица удовлетворяет условиям полноты.

Таблица 1.3.1.

S_{mk}	$TK(S_{mk})$	σ_{mk}	π_{mk}
0	0	0	0
1	1	1	0
-1	-1	-1	0
2	1		
	-1	1	-1
3	1		
	0	1	0
4	1		
	1	1	1
-2	-1		
	1	-1	1
-3	-1		
	0	-1	0
-4	-1		
	-1	-1	-1

4. Деление треугольных кодов на параметр

Эта операция основана на использовании свойства 1.1.5

$$\begin{bmatrix} (\pi_{mk}/R) \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \pi_{mk} \end{bmatrix} - \begin{bmatrix} 0 \\ -\pi_{mk}/R \end{bmatrix},$$

откуда следует, что деление каждого разряда вызывает переносы в два нижних разряда. Следовательно, разрядный результат в этой операции является суммой частного от деления данного разряда на R и переносов из двух верхних разрядов:

$$S_{mk} = \frac{\alpha_{mk}}{R} - \frac{\pi'_{mk}}{R} + \pi''_{mk} \quad (1.4.1)$$

где

α_{mk} - mk -разряд делимого кода,

π'_{mk} - перенос из $(m+1, k)$ -разряда одноименного столбца в (m, k) -разряд,

π''_{mk} - перенос из $(m+1, k+1)$ -разряда из старшего столбца в (m, k) -разряд.

Представим разрядный результат в виде

$$S_{mk} = \sigma_{mk} + \frac{\pi_{mk}}{R}, \quad (1.4.2)$$

где

σ_{mk} - mk -разряд делимого кода,

π_{mk} - перенос из (m, k) -разряда в нижнюю строку,

Из изложенного следует, что алгоритм деления на параметр R -ых треугольных кодов существует, если

- Переносы $(\pi_{mk}, \pi'_{mk}, \pi''_{mk}) \in D_R$,
- любая сумма вида (1.4.1) представима также в виде (1.4.2),

где σ_{mk} - целое число.

Нетрудно убедиться, что эти условия выполняются при любых R . Приведенные формулы позволяют построить таблицу, описывающую процесс деления и удовлетворяющую условиям полноты:

- если π'_{mk}, π''_{mk} принимают в таблице некоторое значение, то π_{mk} также принимает это значение,

- любая комбинация чисел $\alpha_{mk}, \pi'_{mk}, \pi''_{mk}$ присутствует в таблице.

В качестве примера приведена табл. 1.4.1, описывающая одноразрядное деление на $R=3$. Из этой таблицы следует, что $-2 \leq \sigma_{mk} \leq 2$. Следовательно, после деления должно выполняться распространение переносов из разрядов, не удовлетворяющих условию $-1 \leq \sigma_{mk} \leq 1$.

Таблица 1.4.1.

S_{mk}	$TK(S_{mk})$	σ_{mk}	π_{mk}
0	0	0	0
1	1	1	0
-1	-1	-1	0
1/3	0		
	1	-1/3	0
2/3	0		
	-1	1/3	1
4/3	1		
	1	-1/3	1
5/3	2		
	-1	1/3	2
-1/3	0		
	-1	1/3	0
-2/3	-1		
	1	-1/3	-1
-4/3	-1		
	-1	1/3	-1
-5/3	-2		
	1	-1/3	-2

Переносы, возникающие в разрядах 0-строки треугольного кода, не могут распространяться ниже. В связи с этим линейный код, разрядами которого являются переносы π_{0k} , рассматривается как R -ый линейный код функции - остатка $o(x)$, вес $0k$ -разряда которого равен функции ψ_{0k} :

$$\Lambda K_R(o(x)) = \pi_{00} \dots \pi_{0k} \dots \pi_{0k}$$

Коды функций частного и остатка могут быть представлены в виде единого так называемого смешанного кода. Подробнее линейные и смешанные коды будут рассмотрены ниже.

$$\begin{array}{cccccc}
 & & & & & \beta_{0n} \\
 & & & & \dots & \dots \\
 & & & \beta_{0n} & \dots & \beta_{0n} \\
 & & \beta_{0n} & \beta_{0n} & \dots & \beta_{0n} \\
 \text{SK}_R(F(x)/R) = & \beta_{0n} & \beta_{0n} & \beta_{0n} & \dots & \beta_{0n} \\
 & \pi_{00} & \pi_{00} & \pi_{00} & \dots & \pi_{00}
 \end{array}$$

Иллюстрация деления на параметр будет рассмотрена далее.

5. Умножение треугольных кодов

Произведение $P(x)=M(x)F(x)$, где $F(x)$ имеет разложение (1.1.2), может быть определено следующим образом:

$$P(x) = \sum_{k=0}^n \sum_{m=0}^k \alpha_{mk} \psi_{mk} M(x), \quad (1.5.1)$$

Из свойства 1.1.2 следует, что умножение кода некоторой функции на вес ψ_{mk} mk -разряда соответствует mk -сдвигу кода функции. Умножение кода функции на целое (положительное или отрицательное) число α_{mk} состоит из нескольких операций алгебраического сложения. Таким образом, если функции представлены своими R -ыми треугольными кодами, то вычисление по формуле (1.5.1) состоит лишь из операций mk -сдвига и алгебраического сложения кодов. При этом обход mk -разрядов множителя $F(x)$ может производиться в любой последовательности.

Итак, для R -ых треугольных кодов функций соблюдается общий для всех позиционных кодов принцип выполнения умножения. Заметим в заключение, что, согласно (1.3.1), умножение на параметр R описывается формулой

$$R \cdot F(x) = \psi_{01} F(x) + \psi_{11} F(x),$$

то есть состоит из двух сдвигов и сложения.

6. Кодирование и декодирование треугольных кодов

Вначале сформулируем без доказательств две теоремы.

Теорема 1.6.1. Функция, имеющая разложение вида (1.1.1), представима также разложением вида

$$F(x) = \sum_{v=0}^n A_v \psi_{0v}, \quad (1.6.1)$$

где

$$A_v = \sum_{k=v}^n \sum_{m=k-v}^k R^{k-v} (-1)^{m+k+v} \alpha_{mk} C_m^{k-v} \quad (1.6.2)$$

Теорема 1.6.2. Функция, представленная в виде (1.6.1), где

$$A_v = \sum_{i=0}^j R^i \beta_{iv}, \quad \beta_{iv} \in D_R, \quad (1.6.3)$$

представима также разложением вида

$$F(x) = \sum_{i=0}^j \sum_{v=0}^n R^i \psi_{0v} \beta_{iv}, \quad (1.6.4)$$

или

$$F(x) = \sum_{i=0}^j R^i f_i(x), \quad (1.6.5)$$

где

$$f_i(x) = \sum_{v=0}^n \beta_{iv} \psi_{0v}. \quad (1.6.6)$$

Определение 1.6.1. Прямоугольная матрица, составленная из величин $\beta_{iv} \in D_R$ разложения (1.6.4) функции $F(x)$ таким образом, что каждая величина β_{iv} принадлежит v -столбцу и i -строке этой матрицы, называется R -ым прямоугольным кодом функции $F(x)$ по основанию u и обозначается как $PK_R(F(x))$

				$K_R(A_v):$		
	β_{j0}	β_{j1}	...	β_{jv}	...	β_{jv}

$LK_R(f_i(x)):$	β_{i0}	β_{i1}	...	β_{iv}	...	β_{in}

$PK_R(F(x))=$	β_{00}	β_{01}	...	β_{0v}	...	β_{0n}

Отметим некоторые свойства кодов $PK_R(F(x))$.

Свойство 1.6.1. В ν -столбце кода $\text{PK}_R(F(x))$ расположен код $\text{K}_R(A_\nu) = \beta_{0\nu} \dots \beta_{i\nu} \dots \beta_{j\nu}$ числа A_ν по основанию R .

Свойство 1.6.2. В i -строке кода $\text{PK}_R(F(x))$ расположен код $\text{AK}_R(f_i(x)) = \beta_{i0} \dots \beta_{i\nu} \dots \beta_{in}$ функции $f_i(x)$, который будем называть R -ым линейным кодом функции $f_i(x)$. Отметим, что выражения $\text{K}_R(A_\nu)$ и $\text{AK}_R(f_i(x))$, несмотря на внешнее сходство, имеют принципиальное различие, так как вес i -го разряда первого кода равен числу R^i , а вес ν -го разряда второго кода равен функции $\psi_{0\nu}$.

Свойство 1.6.3. Вес i -разряда $\beta_{i\nu}$ кода $\text{PK}_R(F(x))$ есть функция $R^i \psi_{0\nu}$.

Свойство 1.6.4. Сдвиг кода $\text{PK}_R(F(x))$ на одну строку вверх/вниз эквивалентен умножению/делению функции $F(x)$ на R .

Обозначим:
$$F_i(x) = \sum_{e=i}^j R^{e-i} f_e(x).$$

Тогда имеем:
$$F_0(x) = F(x), \quad F_j(x) = f_j(x),$$

$$F_{i-1}(x) = RF_i(x) + f_{i-1}(x), \quad (1.6.7)$$

$$\frac{1}{R} F_{i-1}(x) = F_i(x) + \frac{1}{R} f_{i-1}(x). \quad (1.6.8)$$

Заметим, что структура смешанных кодов допускает выполнение с ними операций алгебраического сложения. Действительно, при выполнении этих операций код SK_R может рассматриваться как совокупность независимых позиционных кодов действительных чисел (разряды кода AK_R) и кода TK_R . Взаимодействие между этими кодами возникает только при возникновении переносов из строки AK_R в 0-строку. Однако переносы между составляющими кодами имеют тот же характер, что и переносы внутри этих кодов. Поэтому такое взаимодействие происходит по обычным правилам распространения переносов в соответствующих кодах.

На использовании этих свойств основаны алгоритмы преобразования прямоугольных кодов $\text{PK}_R(F(x))$ в треугольные коды $\text{TK}_R(F(x))$ и обратно. Действительно, каждая строка $\text{AK}_R(f_i(x))$ кода $\text{PK}_R(F(x))$ может рассматриваться как код

$TK_R(f_i(x))$, содержащий нулевые разряды во всех строках, за исключением нулевой, которая совпадает с $LK_R(f_i(x))$. При этом рекуррентная формула (1.6.7) может рассматриваться как формула, содержащая операции умножения на параметр и сложения кодов TK_R . Ее $(j+1)$ -кратное применение позволяет вычислить искомый код $TK_R(F(x))$ при известном коде $PK_R(F(x))$. С другой стороны, формула (1.6.8) описывает операцию деления на параметр кода $TK_R(F_{i-1}(x))$, в результате чего образуется код частного $TK_R(F_i(x))$ и код остатка $TK_R(f_{i-1}(x))$. Последовательное $(j+1)$ -кратное деление исходного кода $TK_R(F(x))$ на параметр дает в результате код $PK_R(F(x))$.

Таким образом, при известных позиционных кодах чисел A_v разложения (1.1.2) функции $F(x)$ может быть (в силу свойства 1.6.1 кодов PK_R) построен код $PK_R(F(x))$, который преобразуется в код $TK_R(F(x))$. Обратное преобразование позволяет вычислить позиционные коды чисел A_v . Наконец, связь чисел A_v и коэффициентов степенного ряда функции $F(x)$ по переменной y очевидна. Следовательно, любой полином вида

$$F(x) = \sum_{v=0}^n A_v y^v, \quad (1.6.9)$$

коэффициенты A_v которого суть целые и кратные R^v действительные числа, представим кодом $TK_R(F(x))$.

7. Дифференцирование треугольных кодов

Производная функции (1.1.1) равна $\frac{d}{dx}F(x) = \frac{\partial}{\partial y}F(x)\frac{dy}{dx}$.

Поэтому дифференцирование треугольных кодов функции $F(x)$ заключается в определении треугольного кода частной производной $\frac{\partial}{\partial y}F(x)$ и умножении его на известный треугольный

код производной $\frac{dy}{dx}$. Рассмотрим определение треугольного кода

частной производной $\frac{\partial}{\partial y}F(x)$. Оно основано на очевидном

соотношении

$$\frac{\partial}{\partial x}\psi_{mk} = R(k-m)\psi_{m,k-1} - Rm\psi_{m-1,k-1}. \quad (1.7.1)$$

В соответствии со свойством 1.1.5 треугольных кодов имеем:

$$R\psi_{m-1,k-1} = \psi_{m-1,k} + \psi_{m,k}. \quad (1.7.2)$$

Из (1.7.1) и (1.7.2) находим:

$$\frac{\partial}{\partial x}\psi_{mk} = -m\psi_{m-1,k} + (k-2m)\psi_{m,k} + (k-m)\psi_{m+1,k}. \quad (1.7.3)$$

Из (1.7.1) получаем:

$$\frac{\partial}{\partial x} \begin{Bmatrix} & & 0 \\ & 0 & \alpha_{mk} \\ 0 & 0 & 0 \end{Bmatrix} = \begin{Bmatrix} & & 0 \\ R(k-m)\alpha_{mk} & & 0 \\ 0 & -Rm\alpha_{mk} & 0 \end{Bmatrix}. \quad (1.7.4)$$

Из (1.7.3) получаем:

$$\frac{\partial}{\partial x} \begin{Bmatrix} & & 0 \\ & 0 & \alpha_{mk} \\ 0 & 0 & 0 \end{Bmatrix} = \begin{Bmatrix} & & (k-m)\alpha_{mk} \\ 0 & (k-2m)\alpha_{mk} & \\ 0 & 0 & -m\alpha_{mk} \end{Bmatrix}. \quad (1.7.5)$$

Аппаратный способ дифференцирования заключается в схемной организации переносов из mk -разряда в $(m+1,k)$ -разряд и в $(m-1,k)$ -разряд. Величина этих переносов определяется согласно (1.7.5), а их суммирование в данном разряде производится по тем же правилам, что и в коротких операциях.

8. Ступенчатые коды

Рассмотрим еще один способ позиционного кодирования функций, являющийся обобщением вышеизложенного способа “треугольного кодирования”. Предлагаемый способ позволяет использовать в качестве исходных более сложные разложения кодируемых функций по некоторому основанию y , в которые y входит в дробных и отрицательных степенях. Рассмотрим выражение следующего вида:

$$F(x) = \sum_{k=0}^n \sum_{m=0}^{kr} \alpha_{mk} R^k y^{k-m} u^m \quad (1.8.1)$$

где α_{mk} - действительные числа,
 R - целое положительное число,
 $y(x)$, $u(x)$ - функции аргумента x ,
 m, k, r, n - целые положительные числа или нули.

Определение 1.8.1. Двойная сумма вида (1.8.1), в которой функции y и u связаны зависимостью

$$y \sum_{i=0}^r u^i = 1, \quad (1.8.2)$$

называется ступенчатым разложением порядка r функции $F(x)$ по основанию y с параметром R .

Нетрудно заметить аналогию между разложениями (1.1.1) и (1.8.1). Более того, при $r=1$ разложение (1.8.1) превращается в разложение (1.1.1), ибо в этом случае $u=(1-y)/y$ и $y^r u^m = y^{k-m} (1-y)^m$. При $r>1$ зависимость между y и u оказывается более сложной. В частности, при $r=2$

$$u = \frac{-1}{2} \pm \sqrt{\frac{1}{y} - \frac{3}{4}}.$$

По аналогии с определением 1.1.1 для ступенчатого разложения (1.8.1) также вводятся понятия mk -разряда, веса mk -разряда, величины mk -разряда и, наконец, ступенчатого кода функции $F(x)$ – **СТК**($F(x)$). Рассмотрим в качестве иллюстрации выражение (1.8.1) при $r=2$ в развернутом виде:

$$\begin{array}{cccc}
 & & \alpha_{42} & R^2 y^2 u^4 \\
 & & \alpha_{32} & R^2 y^2 u^3 \\
 \alpha_{21} & Ry u^2 & \alpha_{22} & R^2 y^2 u^2 \\
 \alpha_{11} & Ry u & \alpha_{12} & R^2 y^2 u \\
 \text{STK} = \alpha_{00} & \alpha_{01} & Ry & R^2 y^2
 \end{array}$$

Продолжая эту аналогию, отметим, что среди ступенчатых кодов можно выделить R -ые ступенчатые коды функций $\text{STK}_R(F(x))$, у которых разряды α_{mk} принимают значения из множества D_R . Для кодов STK_R существуют правила выполнения арифметических операций, аналогичные тем, которые описаны для кодов TK_R . Основное отличие связано с тем, что выработка переноса при операциях с кодами STK_R основывается на соотношении

$$\text{STK}_R(R) = \begin{array}{c} \left. \begin{array}{c} 1 \\ 1 \\ \dots \\ 1 \end{array} \right\} (r+1) \\ 0 \quad 1 \end{array}$$

то есть перенос распространяется в $(r+1)$ разряд старшего столбца. Поэтому в данный разряд могут поступить переносы одновременно из $(r+1)$ разрядов. Таким образом, для ступенчатых кодов должно выполняться условие $R > 2r$.

Глава 2. Кодирование тригонометрических рядов

1. Треугольные коды функций по основанию $\sin^2(x)$
2. Тригонометрические треугольные коды-ТТК
3. Операции с ТТК
4. Кодирование и декодирование ТТК
5. Погрешность кодирования ТТК
6. Укорочение ТТК
7. Гиперболические треугольные коды

1. Треугольные коды по основанию $\sin^2(x)$

Рассмотренные выше свойства треугольных кодов не зависят от вида основания. В частности, в качестве основания может быть взят сам аргумент: $y=x$. Однако для приложений необходимо выбирать такие основания, которые позволяют кодировать функции наиболее общего вида. Поэтому ниже рассматриваются коды по основанию $y = \sin^2(x)$, которыми могут быть представлены тригонометрические ряды и, в частности, ряды Фурье. Кроме того, коды тригонометрических рядов обладают (как будет видно) рядом достоинств по сравнению с общим случаем кодирования функций.

Далее обозначено: $i=\{0,1,2,3\}$ - верхний индекс соответствующей величины, C_b^a -число сочетаний из b по a .

Предварительно сформулируем ряд теорем, опуская их доказательства для ограничения объема книги. Заметим только, что эти доказательства сводятся к достаточно элементарным, но громоздким преобразованиям.

Теорема 1.1. Числа $S^i(v, n)$ определены по формулам табл. 1.1 и при $v > 0$ связаны следующими рекуррентными зависимостями:

$$S^i(v, v+1) = 2 + S^i(v-1, v);$$

$$S^i(v, n) = 2S^i(v, n-1) - S^i(v, n-2) + S^i(v-1, n-1), n > v+1.$$

Таблица 1.1.

i	$S^i(v, \omega)$	$S^i(0, \omega), \omega > 0$
0	$\frac{2\omega}{\omega+v} C_{\omega+v}^{2v} = C_{\omega+v}^{2v} + C_{\omega+v-1}^{2v}$	2
1	$\frac{2\omega+1}{\omega+v+1} C_{\omega+v+1}^{2v+1} = C_{\omega+v+1}^{2v+1} + C_{\omega+v}^{2v+1}$	$2\omega+1$
2	$C_{\omega+v}^{2v}$	1
3	$C_{\omega+v+1}^{2v+1}$	$\omega+1$

В соответствии с этим числа $S^i(v, \omega)$ могут быть вычислены по следующей схеме:

	n-2	n-1	n
v-1		$+S^i(v-1, n-1)$	
v	$-S^i(v, n-2)$	$+2S^i(v, n-1)$	$S^i(v, n) =$

В таблицах 1.2. i приведены значения чисел S при $i=(0,1,2,3)$ соответственно и при $v < 5$ и $n < 5$.

Таблица 1.2.0. Числа $S^0(v, n)$.

$v \backslash n$	0	1	2	3	4
0	1	2	2	2	2
1		1	4	9	16
2			1	6	20
3				1	8
4					1

Таблица 1.2.1. Числа $S^1(v, n)$.

$v \backslash n$	0	1	2	3	4
0	1	3	5	7	2
1		1	5	14	16
2			1	7	20
3				1	8
4					1

Таблица 1.2.2. Числа $S^2(v, n)$.

$v \backslash n$	0	1	2	3	4
0	1	1	1	1	1
1		1	3	6	10
2			1	5	15
3				1	7
4					1

Таблица 1.2.3. Числа $S^3(v, n)$.

$v \backslash n$	0	1	2	3	4
0	1	2	3	4	5
1		1	4	10	20
2			1	6	21
3				1	8
4					1

Теорема 1.2. Числа $L^i(v, n)$ определены по формулам табл. 1.3

и связаны следующими рекуррентными зависимостями:

$$L^i(v, v) = 1, \quad v \geq 0;$$

$$L^i(v, 0) = L^i(v-1, 0)l^i(v), \quad v \geq 1;$$

$$L^i(v, v-1) = 2L^i(v-1, v-2), \quad v \geq 2;$$

$$L^i(v, n) = L^i(v-1, n-1) + 2L^i(v-1, n) + L^i(v-1, n+1),$$

$$v-2 \geq n > 0, \quad v \geq 3.$$

Таблица 1.3.

i	$L^i(v, \omega)$	$l^i(v)$
0	$C_{2v}^{v-\omega}$	$2(2v-1)/v$
1	$C_{2v+1}^{v-\omega}$	$2(2v+1)/(v+1)$
2	$C_{2v}^{v-\omega} - C_{2v}^{v-\omega-1}$	$2(2v-1)/(v+1)$
3	$C_{2v}^{v-\omega} - C_{2v}^{v-\omega-2}$	$2(2v+1)/(v+2)$

В соответствии с этим числа $L^i(v, n)$ могут быть вычислены по следующей схеме:

	n-1	n	n+1
v-1	$+L^i(v-1, n-1)$	$+2L^i(v-1, n)$	$+L^i(v-1, n+1)$
v		$L^i(v, n) =$	

В таблицах 1.4. i приведены значения чисел L при $i=(0,1,2,3)$ соответственно и при $v \leq 5$ и $n \leq 5$.

Таблица 1.4.0. Числа $L^0(v, n)$.

$v \backslash n$	0	1	2	3	4
0	1				
1	2	1			
2	6	4	1		
3	20	15	6	1	
4	70	56	28	8	1

Таблица 1.4.1. Числа $L^1(v, n)$.

$v \backslash n$	0	1	2	3	4
0	1				
1	3	1			
2	10	5	1		
3	35	21	7	1	
4	126	84	36	9	1

Таблица 1.4.2. Числа $L^2(v, n)$.

$v \backslash n$	0	1	2	3	4
0	1				
1	1	1			
2	2	3	1		
3	5	9	5	1	
4	14	28	20	7	1

Таблица 1.4.3. Числа $L^3(v, n)$.

$v \backslash n$	0	1	2	3	4
0	1				
1	2	1			
2	5	4	1		
3	14	14	6	1	
4	42	48	27	8	1

Теорема 1.3. Функции γ_n^i , определенные по формулам табл.

1.5, имеют разложение по основанию $y = \sin^2(x)$ в виде

$$\gamma_n^i = \sum_{v=0}^n (-1)^v \left(\frac{R}{4}\right)^{n-v} S^i(v, n) \cdot \psi_{ov} \quad (1.1)$$

и при $n > 1$ связаны рекуррентной зависимостью

$$\gamma_n^i = 2\cos(2x)\gamma_{n-1}^i - \left(\gamma_{n-2}^i b^i\right) / a_{n-2}^i \quad (1.2)$$

Напомним, что $\psi_{ov} = y^v = \sin^{2v}(x)$.

Таблица 1.5.

i	$\left(\frac{4}{R}\right)^\omega \gamma_\omega^i$	γ_0^i	$\left(\frac{4}{R}\right)\gamma_1^i$	a_ω^i	b^i
0	$2\cos(2\omega x)$	1	$2-4\sin^2 x$	1 ($\omega=0$) 2 ($\omega>0$)	2
1	$\frac{\sin(2\omega+1)x}{\sin(x)}$	1	$3-4\sin^2 x$	1	1
2	$\frac{\cos((2\omega+1)x)}{\cos(x)}$	1	$1-4\sin^2 x$	1	1
3	$\frac{\sin(2(\omega+1)x)}{\sin(2x)}$	1	$2-4\sin^2 x$	1	1

2. Кодирование тригонометрических рядов

Рассмотрим теперь коды $\text{TK}(\gamma_\omega^i)$ по основанию $\sin^2 x$. В соответствии с табл. 1.5 и формулой (1.2), а также учитывая, что, $2\cos(2x) = \gamma_1^i$ находим:

$$\begin{aligned}\gamma_2^0 &= \gamma_1^0 \gamma_1^0 - 2\gamma_0^0, & \gamma_2^i &= \gamma_1^0 \gamma_1^i - \gamma_0^i, \\ \gamma_n^0 &= \gamma_1^0 \gamma_{n-1}^0 - 4\gamma_{n-2}^0, & \gamma_n^i &= \gamma_1^0 \gamma_{n-1}^i - 2\gamma_{n-2}^i.\end{aligned}$$

Табл. 1.6 кодов $\text{TK}(\gamma_\omega^i)$ для начальных значений ω приведена ниже.

Таблица 1.6.

i	$\text{TK}(\gamma_0^i)$	$\text{TK}(\left(\frac{4}{R}\right)\gamma_1^i)$	$\text{TK}(\left(\frac{4}{R}\right)\gamma_2^i)$
0	1	2	$-4/R$
1	1	3	$-4/R$
2	1	1	$-4/R$
3	1	2	$-4/R$

Теорема 1.4. Функции λ_ω^i , ε_ω^i , η^i определены по формулам табл. 1.7 и связаны следующими соотношениями:

$$\eta^i \gamma_\omega^i = \left(\frac{R}{4}\right)^\omega a_\omega^i c^i \varepsilon_\omega^i \quad (1.3)$$

$$\lambda_\omega^i = \left(\frac{R}{4}\right)^\omega \eta^i \gamma_\omega^i \quad (1.4)$$

$$\lambda_\omega^i = a_\omega^i c^i \varepsilon_\omega^i \quad (1.5)$$

Таблица 1.7.

i	ε_ω^i	λ_ω^i	c^i	η^i
0	$\cos 2\omega x$	$2\cos 2\omega x$	1	1
1	$\sin(2\omega + 1)x$	$\sqrt{R}\sin(2\omega + 1)x$	\sqrt{R}	$\sqrt{R}\sin x$
2	$\cos(2\omega + 1)x$	$\sqrt{R}\cos(2\omega + 1)x$	\sqrt{R}	$\sqrt{R}\cos x$
3	$\sin(2(\omega + 1))x$	$R/2 \sin(2(\omega + 1))x$	$R/2$	$\frac{R}{2} \sin 2x$

Рассмотрим теперь разложения функций по основанию $y = \sin^2(x)$ с весом m -разряда, соответственно равным

$$\psi_{mk} = R^k [\sin^2(x)]^{k-m} [\cos^2(x)]^m. \quad (1.6)$$

В частности,

$$\psi_{ok} = R^k \sin^{2k}(x). \quad (1.7)$$

Далее мы будем часто оперировать функциональными рядами вида

$$F^i(x) = \sum_{\omega=0}^n D_{\omega}^i \varepsilon_{\omega}^i, \quad (1.8)$$

$$F^i(x) = \eta^i \sum_{v=0}^n A_v \psi_{0v}, \quad (1.9)$$

$$F^i(x) = \sum_{\omega=0}^n E_{\omega}^i \gamma_{\omega}^i, \quad (1.10)$$

$$F^i(x) = \sum_{\omega=0}^n H_{\omega}^i \lambda_{\omega}^i, \quad (1.11)$$

где D_{ω}^i , A_v , E_{ω}^i , H_{ω}^i - коэффициенты рядов, действительные числа. Далее утверждается, что тригонометрический ряд общего вида может быть представлен четырьмя составляющими - частичными рядами функций $F^i(x)$ по функциям ε_{ω}^i , а каждый из этих частичных рядов может быть преобразован в ряд по функциям λ_n^i , γ_n^i , ψ_{0v} и, далее, каждый из этих рядов представим ТТК по основанию $y = \sin^2(x)$ с весом разряда в виде (1.6)

Теорема 1.5. Функция $F^i(x)$, представленная одним из функциональных рядов вида (1.8-11), разложима также в любой из этих рядов и их коэффициенты связаны следующими соотношениями:

$$H_{\omega}^i = \sum_{v=\omega}^n (R/4)^v A_v L^i(v, \omega), \quad (1.12)$$

$$E_{\omega}^i = (4/R)^{\omega} H_{\omega}^i, \quad (1.13)$$

$$E_{\omega}^i = (4/R)^{\omega} \sum_{v=\omega}^n (R/4)^v A_v L^i(v, \omega), \quad (1.14)$$

$$D_{\omega}^i = a_{\omega}^i c^i (-1)^{\omega} H_{\omega}^i, \quad (1.15)$$

$$A_v = (-1)^v (4/R)^v \sum_{v=\omega}^n S^i(v, \omega) H_{\omega}^i (-1)^{\omega}, \quad (1.16)$$

$$D_{\omega}^i = a_{\omega}^i c^i (-1)^{\omega} \sum_{v=\omega}^n (R/4)^v A_v L^i(v, \omega), \quad (1.17)$$

$$A_v = \frac{(-1)^v}{c^i} \left(\frac{4}{R}\right)^v \sum_{v=\omega}^n S^i(v, \omega) \frac{D_\omega^i}{a_\omega^i} \quad (1.18)$$

$$E_\omega^i = \frac{(-1)^\omega}{a_\omega^i c^i} \left(\frac{4}{R}\right)^\omega D_\omega^i \quad (1.19)$$

Из этих формул следует, что

$$\left\| \begin{array}{l} \text{функция } F^i(x) \text{ имеет равное количество} \\ \text{членов во всех разложениях вида (1.8-11)} \end{array} \right\| \quad (1.20)$$

Из (1.17) в частности следует, что

$$\eta^i \psi_{on} = \sum_{\omega=0}^n D_\omega^i \varepsilon_\omega^i, \quad (1.21)$$

где

$$D_\omega^i = a_\omega^i c^i (-1)^\omega \left(\frac{R}{4}\right)^n L^i(n, \omega). \quad (1.22)$$

В частности, последний член ряда (1.21)

$$g_n^i = (-1)^n \left(\frac{R}{4}\right)^n a_n^i c^i \varepsilon_n^i. \quad (1.23)$$

Пример 1.1. Преобразование частичного ряда при $i=3$.

Рассмотрим функцию

$$\Phi_1(x) = 4 \sin x \cos x \cdot \left\{ \begin{array}{l} -3408 + 13538\psi_{01} - 16776\psi_{02} + 11056\psi_{03} \\ -4310\psi_{04} + 1032\psi_{05} - 140\psi_{06} + 8\psi_{07} \end{array} \right\}$$

Найдем для этой же функции коэффициенты ряда (1.8) по формуле (1.17) – в верхней строке таблицы указан коэффициент A_v , во втором столбце таблицы указан коэффициент $a_\omega^i c^i (-1)^\omega$, а в таблице приведены коэффициенты $L^i(v, \omega)$:

ω		-3408	13538	-16776	11056	-4310	1032	-140	8	D_ω^3
0	2*	1	2	5	14	42	132	429	1430	2312
1	-2*		1	4	14	48	165	572	2002	-1108
2	2*			1	16	27	110	429	1638	-492
3	-2*				1	8	44	208	910	-288
4	2*					1	10	65	350	-580
5	-2*						1	12	90	-144
6	2*							1	14	-56
7	-2*								1	-16

Таким образом,

$$\Phi_1(x) = \begin{cases} 2312\sin 2x - 1108\sin 4x - 492\sin 6x - \\ 288\sin 8x - 580\sin 10x - 144\sin 12 - \\ 56\sin 14x - 16\sin 16x \end{cases}$$

Далее будем применять следующее обозначение:
 $[F \parallel h_k] = \{h_0, h_1, \dots, h_k, \dots\}$ - множество коэффициентов h_k функционального ряда для функции F . Таким образом, для некоторая функция F может быть представлена несколькими множествами коэффициентов:

$$[F \parallel D_\omega^i], [F \parallel A_v], [F \parallel E_\omega^i], [F \parallel H_\omega^i].$$

Пример 1.2. Преобразование частичного ряда функции γ .

Построим функциональный ряд (1.9) для функции $F(x) = \gamma_3^0(x)$ при $R=4$. В соответствии с (1.1) находим:

$$\gamma_3^0 = \begin{bmatrix} S^0(0,3)\psi_{00} - S^0(1,3)\psi_{01} - S^0(2,3)\psi_{02} - S^0(3,3)\psi_{03} = \\ 2\psi_{00} - 9\psi_{01} + 6\psi_{02} - \psi_{03} \end{bmatrix}$$

В соответствии с (1.9) при $\eta^0 = 1$ имеем:

$$[\gamma_3^0 \parallel A_v] = \{+2, -9, +6, -1\}.$$

Применяя формулу (1.12) для функции H_ω^0 , найдем (в каждом слагаемом первый сомножитель - число A_v , а второй сомножитель - число $L^i(v, \omega)$):

v	0	1	2	3	
H_0^0	+2*1	-9*2	+6*6	-1*20	=0
H_1^0	0	-9*1	+6*4	-1*15	=0
H_2^0			+6*1	-1*6	=0
H_3^0				-1*1	=-1

Таким образом, $\left[\gamma_3^0 \parallel H_\omega^0 \right] = \{0,0,0,-1\}$.

При известных H_ω^0 вновь по формуле (1.16) найдем числа A_v для функции γ_3^0 (в каждом слагаемом первый сомножитель - число $(H_\omega^0(-1)^\omega)$, а второй сомножитель - число $S^i(v, \omega)$):

Учитывая формулу (1.15), находим $\left[\gamma_3^0 \parallel D_\omega^0 \right] = \{0,0,0,-2\}$

ω	0	1	2	3	
A_0	0*1	0*2	0*2	1*2	=2
$-A_1$		0*1	0*4	1*9	=9
A_2			0*1	1*6	=6
$-A_3$				1*1	=1

Полученный результат совпадает с приведенным в начале примера.

В следующих примерах приняты следующие обозначения:

$$d = \sum A_v L^i(v, \omega), \quad D_\omega^i = (-1)^\omega a_\omega^i c^i d,$$

$$z = \sum D_\omega^i \frac{S^i(v, \omega)}{a_\omega^i}, \quad A_v = z \frac{(-1)^v}{c^i}.$$

Пример 1.3а. Преобразование частичного ряда при $i=0$.

Рассмотрим преобразование функции вида (1.9) $f(x) = (-11 + 3\psi_{01} - 2\psi_{02})$ по (1.17) при $R=4$, $i=0$. При этом $c^i = 1$, $a_{\omega=0}^i = 1$, $a_{\omega>0}^i = 2$. В следующей таблице множителями являются числа $L^i(v, \omega)$.

0	1	2	d	D_ω^i
A_0	$2 A_1$	$6 A_2$	-17	-17
	A_1	$4 A_2$	-5	-10
		A_2	-2	-4

Таким образом, данная функция имеет разложение вида (1.8):

$$f(x) = -17-10\text{Cos}(2x)-4\text{Cos}(4x).$$

Пример 1.3. Взаимные преобразования частичного ряда при $i=0$.

Рассмотрим преобразования функций (1.17) и (1.18) при $R=4$, $i=0$. При этом $c^i = 1$, $a_{\omega=0}^i = 1$, $a_{\omega>0}^i = 2$. В следующей таблице множителями являются числа $L^i(v, \omega)$.

0	1	2	3	4	d	D_{ω}^i
A_0	$2 A_1$	$6 A_2$	$20 A_3$	$70 A_4$	32	32
	A_1	$4 A_2$	$15 A_3$	$56 A_4$	21	-42
		A_2	$6 A_3$	$28 A_4$	-4	-8
			A_3	$8 A_4$	2	-4
				A_4	-1	-2

Таким образом, данная функция имеет разложение вида (1.8):

$$F(x) = 32 - 42\text{Cos}2x - 8\text{Cos}4x - 4\text{Cos}6x - 2\text{Cos}8x.$$

В следующей таблице множителями являются числа $S^i(v, \omega) / a_{\omega}^i$.

0	1	2	3	4	z	A_v
D_0	D_1	D_2	D_3	D_4	-24	-24
	$\frac{D_1}{2}$	$2 D_2$	$\frac{9 D_3}{2}$	$8 D_4$	-71	71
		$\frac{D_2}{2}$	$6 D_3$	$10 D_4$	-36	-36
			$\frac{D_3}{2}$	$4 D_4$	-10	10
				$\frac{D_4}{2}$	-1	-1

Таким образом, данная функция имеет разложение вида (1.9):

$$F(x) = -24 + 71\psi_{01} - 36\psi_{02} + 10\psi_{03} - \psi_{04}.$$

Пример 1.4. Взаимные преобразования частичного ряда при $i=1$.

Рассмотрим преобразования функций (1.17) и (1.18) при $R=4$, $i=1$. При этом $c^i = 2$, $a_{\omega}^i = 1$. В следующей таблице множителями являются числа $L^i(v, \omega)$.

0	1	2	d	D_{ω}^i
A_0	$3 A_1$	$10 A_2$	128	256
	A_1	$5 A_2$	-20	10
		A_2	4	2

Таким образом, данная функция имеет разложение вида (1.8):

$$F'(x) = 256\sin x + 10\sin 3x + 2\sin 5x.$$

В следующей таблице множителями являются числа $S^i(v, \omega) / a_{\omega}^i = S^i(v, \omega)$.

0	1	2	z	A_v
D_0	$3 D_1$	$5 D_2$	296	148
	D_1	$5 D_2$	20	-10
		D_2	2	1

Таким образом, данная функция имеет разложение вида (1.9):

$$F'(x) = 2\sin x(148 - 10\psi_{01} + \psi_{02}).$$

Пример 1.5. Взаимные преобразования частичного ряда при $i=2$.

Рассмотрим преобразования функций (1.17) и (1.18) при $R=4$, $i=2$. При этом $c^i = 2$, $a_{\omega}^i = 1$. В следующей таблице множителями являются числа $L^i(v, \omega)$.

0	1	2	d	D_{ω}^i
A_0	A_1	A_2	128	256
	A_1	$3 A_2$	5	-10
		A_2	1	2

Таким образом, данная функция имеет разложение вида (1.8):

$$F''(x) = 256\cos x - 10\cos 3x + 2\cos 5x.$$

В следующей таблице множителями являются числа $S^i(v, \omega) / a_{\omega}^i = S^i(v, \omega)$.

0	1	2	z	A_v
D_0	D_1	D_2	248	124
	D_1	$3 D_2$	-4	2
		D_2	2	1

Таким образом, данная функция имеет разложение вида (1.9):

$$F''(x) = 2\text{Cos}x(124 + 2\psi_{01} + \psi_{02}).$$

Пример 1.6. Взаимные преобразования частичного ряда при $i=3$.

Рассмотрим преобразования функций (1.17) и (1.18) при $R=4$, $i=3$. При этом $c^i = 2$, $a_{\omega}^i = 1$. В следующей таблице множителями являются числа $L^i(v, \omega)$.

0	1	2	3	d	D_{ω}^i
A_0	$2 A_1$	$5 A_2$	$14 A_3$	42	84
	A_1	$4 A_2$	$14 A_3$	-16	32
		A_2	$6 A_3$	12	24
			A_3	-8	16

Таким образом, данная функция имеет разложение вида (1.8):

$$F'''(x) = 84 + 32\text{Sin}2x + 24\text{Sin}4x + 16\text{Sin}8x.$$

В следующей таблице множителями являются числа

$$S^i(v, \omega) / a_{\omega}^i = S^i(v, \omega).$$

0	1	2	3	z	A_v
D_0	$2 D_1$	$3 D_2$	$4 D_3$	284	142
	D_1	$4 D_2$	$10 D_3$	288	-144
		D_2	$6 D_3$	120	60
			D_3	16	-8

Таким образом, данная функция имеет разложение вида (1.9):

$$F'''(x) = 4\text{Sin}x\text{Cos}x(142 - 144\psi_{01} + 60\psi_{02} - 8\psi_{03}).$$

2. Тригонометрические треугольные коды

Вначале сформулируем две теоремы.

Теорема 2.1. Функция $\Phi(x)$, заданная тригонометрическим рядом общего вида

$$\Phi(x) = \sum_{u=0}^U (a_u \cos(ux) + b_u \sin(ux)),$$

представима также разложениями вида

$$\Phi(x) = \sum_{i=0}^3 F^i(x), \quad \Phi(x) = \sum_{i=0}^3 \eta^i f^i(x),$$

где функции $F^i(x) = \eta^i f^i(x)$ имеют разложения вида (1.8), где

$$D_{\omega}^i = a_u \quad \text{при} \quad u = 2\omega \quad i = 0,$$

$$D_{\omega}^i = b_u \quad \text{при} \quad u = 2\omega + 1, \quad i = 1,$$

$$D_{\omega}^i = a_u \quad \text{при} \quad u = 2\omega + 1, \quad i = 2,$$

$$D_{\omega}^i = b_u \quad \text{при} \quad u = 2\omega + 2, \quad i = 3,$$

Теорема 2.2. Функция $\Phi(x)$, определенная в соответствии с теоремой 2.1, имеет разложение вида

$$\Phi(x) = \sum_{k=0}^n \sum_{m=0}^k \alpha_{mk} \psi'_{mk} \quad (2.1)$$

где $\psi'_{mk} = \sqrt{\psi_{mk}} = \psi_{m/2, k/2}$ или

$$\psi'_{mk} = R^{k/2} \sin^{k-m} x \cos^m x. \quad (2.2)$$

Сравнивая формулы (1.1.2) и (2.1), замечаем их полную идентичность, откуда следует, что на основе последней формулы также может быть построен треугольный код, который, в отличие от рассмотренного ранее, будем называть тригонометрическим треугольным кодом функции $\Phi(x)$ и обозначать как $\text{ТТК}(\Phi(x))$. В этом случае также вводится понятие $\text{ТТК}_R(\Phi(x))$. Для иллюстрации запишем разложение (2.1) в развернутом виде:

				$\alpha_{nn} R^{n/2} \text{Cos}^n x$
		
		$\alpha_{kk} R^{k/2} \text{Cos}^k x$...	$\alpha_{kn} R^{n/2} \text{Sin}^{n-k} x \text{Cos}^k x$

$\alpha_{11} \sqrt{R} \text{Cos} x$...	$\alpha_{1k} R^{k/2} \text{Sin}^{k-1} x \text{Cos} x$...	$\alpha_{1n} R^{n/2} \text{Sin}^{n-1} x \text{Cos} x$
α_{00}	$\alpha_{01} \sqrt{R} \text{Sin} x$...	$\alpha_{ok} R^{k/2} \text{Sin}^k x$...
				$\alpha_{on} R^{n/2} \text{Sin}^n x$

Свойство 2.1. Тригонометрические треугольные коды обладают свойствами 1, 2, 3, 4 обычных треугольных кодов.

Свойство 2.2. Имеет место соотношение

$$R\psi'_{mk} = \psi'_{m,k+2} + \psi'_{m+2,k+2}.$$

Свойство 2.3. ТТК($\Phi(x)$) является композицией четырех кодов ТТК($F^i(x)$), где функции $F^i(x)$ удовлетворяют условиям теоремы 2.1; на схеме цифрами 0, 1, 2, 3 обозначены разряды кодов при $i = 0, 1, 2, 3$ соответственно:

$$\begin{array}{ccccccc} & & & & & 2 & \\ & & & & & 0 & 3 \\ & & & & 2 & 1 & 2 \\ & & & 0 & 3 & 0 & 3 \\ & & 2 & 1 & 2 & 1 & 2 \\ & 0 & 3 & 0 & 3 & 0 & 3 \end{array}$$

Свойство 2.4. Каждая из функций η^i имеет ТТК $_R(\eta^i(x))$ - см. теорему 1.4 и табл. 2.1. Умножение кода ТТК $_R$ некоторой функции на код ТТК $_R(\eta^i(x))$ равносильно сдвигу кода этой функции, точнее, 00-, 01-, 11-, 12- сдвигу при $i = 0, 1, 2, 3$ соответственно.

Таблица 2.1.

i	0	1	2	3
η^i	1	$\sqrt{R} \text{Sin} x$	$\sqrt{R} \text{Cos} x$	$\frac{R}{2} \text{Sin} 2x$
ТТК $_R(\eta^i)$	<div>0</div> <div>0 0</div> <div>1 0 0</div>	<div>0</div> <div>0 0</div> <div>0 1 0</div>	<div>0</div> <div>1 0</div> <div>0 0 0</div>	<div>0</div> <div>0 1</div> <div>0 0 0</div>

Таким образом,

$$\psi'_{ab} = \left\{ \begin{array}{l} \eta^0 \psi_{mk}, \text{ if } a = 2m, \ b = 2k; \\ \eta^1 \psi_{mk}, \text{ if } a = 2m, \ b = 2k + 1; \\ \eta^2 \psi_{mk}, \text{ if } a = 2m + 1, \ b = 2k + 1; \\ \eta^3 \psi_{mk}, \text{ if } a = 2m + 1, \ b = 2k + 2; \end{array} \right\} \quad (2.3)$$

Определение 2.1. *Расширением* кода **ТК** в код **ТТК** называется перестановка mk -разрядов исходного кода **ТК** на место $(2m, 2k)$ -разрядов результирующего кода **ТТК**. Обратное преобразование называется *сжатием*.

Например, если $\begin{matrix} & & 1 \\ & & 0 \end{matrix}$, то $\begin{matrix} & & 0 \\ & & 0 \end{matrix}$.

$\text{ТК}_R = \begin{matrix} 1 & 1 \end{matrix}$ $\text{ТТК}_R = \begin{matrix} 1 & 0 & 1 \end{matrix}$

Каждая из функций $f^i(x)$ может быть представлена кодом $\text{ТТК}(f^i(x))$ по основанию $y = \sin^2(x)$, что следует из предыдущих теорем. Расширением каждый из этих кодов может быть преобразован в тригонометрический код $\text{ТТК}(f^i(x))$, вес mk -разряда которого имеет вид (2.2). Из сказанного, а также из теоремы 2.1 и свойств 2.3 и 2.4 следует, что код $\text{ТТК}(\Phi(x))$ может быть получен расширением, сдвигом и совмещением кодов $\text{ТТК}(f^i(x))$. Такое преобразование будем называть *компоновкой* $\langle \text{ТТК}(f^i(x)) \rangle \dots \langle \text{ТТК}(\Phi(x)) \rangle$, а обратное ему - *декомпоновкой* $\langle \text{ТТК}(\Phi(x)) \rangle \dots \langle \text{ТТК}(f^i(x)) \rangle$.

Пример 2.1. Расширение, сдвиг и компоновка кодов.

Пусть $R = 4$, $\alpha \in \{-2, -1, 0, 1\}$,

$$\begin{matrix} & & & & & & 0 \\ & & & & & 0 & 0 \\ & & & & 0 & 0 & 0 \\ & & & 0 & 1 & 0 & -1 \\ & & 1 & -1 & 0 & 0 & 0 \\ & 1 & -1 & -1 & 0 & 0 & 0 \\ \text{ТТК}(\Phi(x)) = -2 & 1 & 1 & 0 & 0 & 0 & 0 \end{matrix}$$

Выделяя из этого кода составляющие коды и сжимая их, то есть производя декомпоновку, получаем:

$$\begin{aligned} \text{TK}_4(f^0(x)) &= \begin{pmatrix} 1 & -1 \\ -2 & 1 \end{pmatrix}, & \text{TK}_4(f^1(x)) &= \begin{pmatrix} 0 & 0 \\ 1 & -1 \end{pmatrix}, \\ \text{TK}_4(f^2(x)) &= \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix}, & \text{TK}_4(f^3(x)) &= \begin{pmatrix} 0 & 0 \\ -1 & 0 \end{pmatrix} \end{aligned}$$

Расширение, сдвиг и совмещение, то есть компоновка, дает в результате снова исходный код.

Пример 2.2. Кодирование функции. Построим код функции $\Phi(x) = F(x) + F'(x) + F''(x) + F'''(x)$, где слагаемые функции определены в примерах 1.3-6:

$$F(x) = 32 - 42\text{Cos}2x - 8\text{Cos}4x - 4\text{Cos}6x - 2\text{Cos}8x,$$

$$F'(x) = 256\text{Sin}x + 10\text{Sin}3x + 2\text{Sin}5x,$$

$$F''(x) = 256\text{Cos}x - 10\text{Cos}3x + 2\text{Cos}5x,$$

$$F'''(x) = 84 + 32\text{Sin}2x + 24\text{Sin}4x + 16\text{Sin}8x,$$

или

$$F(x) = -24 + 71\psi_{01} - 36\psi_{02} + 10\psi_{03} - \psi_{04},$$

$$F'(x) = 2\text{Sin}x(148 - 10\psi_{01} + \psi_{02}),$$

$$F''(x) = 2\text{Cos}x(124 + 2\psi_{01} + \psi_{02}),$$

$$F'''(x) = 4\text{Sin}x\text{Cos}x(142 - 144\psi_{01} + 60\psi_{02} - 8\psi_{03}).$$

Следовательно,

ТТК $F(x)$ =		0	0	0	0	0	0	0	0
	-24	0	71	0	-36	0	10	0	-1
ТТК $F'(x)$ =		0	0	0	0	0	0	0	0
	0	148	0	-10	0	1	0	0	0
ТТК $F''(x)$ =		124	0	2	0	1	0	0	0
	0	0	0	0	0	0	0	0	0
ТТК $F'''(x)$ =		0	142	0	-144	0	60	0	-8
	0	0	0	0	0	0	0	0	0
ТТК $\Phi(x)$ =		124	142	2	-144	1	60	0	-8
	-24	148	71	-10	-36	1	10	0	-1

3. Операции с тригонометрическими треугольными кодами

3.1. Короткие операции. Этим термином мы будем называть операции алгебраического сложения, умножения и деления на параметр. Следствием свойства 2.3 является то, что при выполнении коротких операций тригонометрические треугольные коды можно рассматривать как состоящие из четырех независимых частей - треугольных кодов и выполнять операции над этими частями по правилам, описанным выше. Независимость четырех частей тригонометрического треугольного кода следует из того, что в треугольном коде выработка переноса основывается на соотношении (1.3.1), а в тригонометрическом треугольном коде - на соотношении

$$\begin{matrix} & & 1 \\ & 0 & 0, \\ \text{ТТК}_R(R) = & 0 & 0 & 1 \end{matrix}$$

являющимся следствием свойства 2. Отсюда, например, получаем:

$$\left\{ \begin{matrix} & & 0 \\ & 0 & 0 \\ R & R & 0 \\ R & R & 0 & 0 \end{matrix} \right\} = \left\{ \begin{matrix} & & & 0 \\ & & 1 & 1 \\ & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{matrix} \right\}$$

На этом примере видно, что переносы, возникающие в четырех младших разрядах, не взаимодействуют между собой.

3.2. Умножение тригонометрических треугольных кодов полностью аналогично умножению треугольных кодов. Эта аналогия объясняется тем, что для построения и тех и других используются однотипные разложения, а умножение кода на вес разряда в обоих случаях эквивалентно сдвигу.

Для сравнения рассмотрим обычное умножение тригонометрических рядов.

Пример 3.1. Умножение тригонометрических рядов.

Найдем произведение тригонометрических рядов

$\Phi_1(x) = F(x) \cdot F'''(x)$, где функции $F(x)$, $F'''(x)$ определены в примере 2.2. При этом

$$\Phi_1(x) = \left\{ \begin{array}{l} (32 - 42\cos 2x - 8\cos 4x - 4\cos 6x - 2\cos 8x) \cdot \\ (84 + 32\sin 2x + 24\sin 4x + 16\sin 8x) \end{array} \right\}$$

Применяя формулу $\sin \alpha \cos \beta = \frac{1}{2} \{ \sin(\alpha - \beta) + \sin(\alpha + \beta) \}$, получаем:

$$\Phi_1(x) = \left\{ \begin{array}{l} 2312\sin 2x - 1108\sin 4x - 492\sin 6x - \\ 288\sin 8x - 580\sin 10x - 144\sin 12 - \\ 56\sin 14x - 16\sin 16x \end{array} \right\}$$

Пример 3.2. Умножение тригонометрических рядов.

Найдем произведение рядов вида (1.9) $\Phi_1(x) = F(x) \cdot F'''(x)$, где функции $F(x)$, $F'''(x)$ определены в примере 2.2. При этом

$$\Phi_1(x) = \left\{ \begin{array}{l} (-24 + 71\psi_{01} - 36\psi_{02} + 10\psi_{03} - \psi_{04}) \cdot \\ (4\sin x \cos x \cdot (142 - 144\psi_{01} + 60\psi_{02} - 8\psi_{03})) \end{array} \right\}$$

Применяя формулу $(a\psi_{0i}) \cdot (b\psi_{0j}) = ab\psi_{0,i+j}$, получаем:

$$\Phi_1(x) = 4\sin x \cos x \cdot \left\{ \begin{array}{l} -3408 + 13538\psi_{01} - 16776\psi_{02} + 11056\psi_{03} \\ -4310\psi_{04} + 1032\psi_{05} - 140\psi_{06} + 8\psi_{07} \end{array} \right\}$$

В примере 1.1 показано, что эта же функция может быть представлена рядом вида (1.8):

$$\Phi_1(x) = \left\{ \begin{array}{l} 2312\sin 2x - 1108\sin 4x - 492\sin 6x - \\ 288\sin 8x - 580\sin 10x - 144\sin 12 - \\ 56\sin 14x - 16\sin 16x \end{array} \right\}$$

Таким образом, результат этого примера совпадает с полученным в примере 3.1.

3.3. Дифференцирование тригонометрических треугольных кодов основано на очевидном соотношении

$$\frac{d}{dx} \psi'_{mk} = (k - m) \psi'_{m+1,k} - m \psi'_{m-1,k}. \quad (3.1)$$

В соответствии со свойством 1.1.5 треугольных кодов имеем:

$$R \psi'_{m-1,k-2} = \psi'_{m+1,k} + \psi'_{m-1,k}. \quad (3.2)$$

Из (3.1) и (3.2) находим:

$$\frac{d}{dx} \psi'_{mk} = k \psi'_{m+1,k} - R m \psi'_{m-1,k-2}. \quad (3.3)$$

Из (3.1) получаем:

$$\frac{d}{dx} \begin{Bmatrix} 0 \\ 0 & 0 & \alpha_{mk} \\ 0 & 0 & 0 \end{Bmatrix} = \begin{Bmatrix} (k-m)\alpha_{mk} \\ 0 & 0 \\ 0 & 0 & -m\alpha_{mk} \end{Bmatrix}. \quad (3.4)$$

Из (3.3) получаем:

$$\frac{d}{dx} \begin{Bmatrix} 0 \\ 0 & 0 & \alpha_{mk} \\ 0 & 0 & 0 \end{Bmatrix} = \begin{Bmatrix} k\alpha_{mk} \\ 0 & 0 \\ -Rm\alpha_{mk} & 0 \end{Bmatrix}. \quad (3.5)$$

Возможны, по крайней мере, три способа дифференцирования.

Первый из них заключается в том, что по последним формулам определяются коды $\text{ТТК}_R\left(\frac{d}{dx}\psi'_{mk}\right)$ и дифференцирование выполняется по формуле

$$\frac{d}{dx}\Phi(x) = \sum_{k=0}^n \sum_{m=0}^k \alpha_{mk} \frac{d}{dx}\psi'_{mk}.$$

Недостаток этого способа заключается в необходимости использовать таблицы кодов функций $\left(\frac{d}{dx}\psi'_{mk}\right)$, требующие определенного объема памяти и времени для обращения. Заметим, что коды можно хранить в виде треугольных кодов $\text{ТК}_R\left(\frac{d}{dx}\psi'_{mk}\right)$, т.к. они содержат значащие разряды только в одной составляющей тригонометрического кода.

Второй способ дифференцирования - аппаратный - заключается в схемной организации переносов из mk -разряда в $(m+1, k)$ -разряд и в $(m-1, k)$ -разряд. Величина этих переносов определяется согласно (3.4), а их суммирование в данном разряде производится по тем же правилам, что и в коротких операциях.

Наконец, третий способ состоит в вычислении по формуле

$$\frac{d}{dx}\Phi(x) = \left[\frac{\psi'_{10} \sum_{k=0}^n k \sum_{m=0}^k \alpha_{mk} \psi'_{mk} -}{\psi'_{12} \sum_{m=0}^n m \sum_{k=m}^m \alpha_{mk} \psi'_{mk}} \right],$$

которая является следствием соотношения (3.5). Для вычисления по этой формуле используется следующий

Алгоритм 3.1. Дифференцирование кода $\text{ТТК}_R(\Phi(x))$.

1. Все столбцы исходного кода последовательно умножаются на номер столбца, а полученные произведения суммируются, т.е. вычисляется величина

$$\sum_{k=0}^n k \sum_{m=0}^k \alpha_{mk} \psi'_{mk}$$

2. Производится 10-сдвиг результата п.1, т.е. вычисляется величина

$$\psi'_{10} \sum_{k=0}^n k \sum_{m=0}^k \alpha_{mk} \psi'_{mk}$$

3. Все строки исходного кода последовательно умножаются на номер строки, а полученные произведения суммируются, т.е. вычисляется величина

$$\sum_{m=0}^n m \sum_{k=m}^n \alpha_{mk} \psi'_{mk}$$

4. Производится (-1,-2)-сдвиг результата п.3. В результате этого сдвигается 0-строка и диагональ сдвигаемого кода. Это соответствует упрощению формулы (3.1) для теряемых разрядов:

$$\frac{d}{dx} \psi'_{0k} = k \psi'_{1k} \psi, \quad \frac{d}{dx} \psi'_{kk} = -k \psi'_{k-1,k}.$$

5. Производится умножение на параметр результата по п. 4, т.е. вычисляется величина

$$\frac{R}{\psi'_{12}} \sum_{m=0}^n m \sum_{k=m}^n \alpha_{mk} \psi'_{mk}.$$

6. Из результата п. 2 вычитается результат п. 5, т.е. определяется искомая функция.

3.4. Интегрирование тригонометрических треугольных кодов. Введем обозначение

$$J_{mk} = \int \psi'_{mk}(x) dx. \quad (3.6)$$

При этом

$$\int \Phi(x) = \sum_{k=0}^n \sum_{m=0}^k \alpha_{mk} J_{mk}.$$

Интегрирование можно выполнять непосредственно по этой формуле. Для этого предварительно необходимо вычислить коды $\text{ТТК}_R(J_{mk}(x))$ для всех m и k . Как и при дифференцировании, коды можно хранить в виде треугольных кодов $\text{ТК}_R(J_{mk}(x))$, т.к. они содержат значащие разряды только в одной составляющей тригонометрического кода.

2. Кодирование тригонометрических рядов

При интегрировании функции, имеющей постоянную составляющую, возникает составляющая, пропорциональная аргументу x . Кодирование этой составляющей должно производиться с учетом области изменения аргумента x .

Формулы для вычисления J_{mk} , которые должны быть использованы для составления таблиц кодов $\text{ТТК}_R(J_{mk}(x))$, могут быть определены непосредственным интегрированием функций веса ψ'_{mk} . Заменяя в (3.1) (m) на $(m-1)$ и интегрируя находим:

$$\psi'_{m-1,k} = (k - m + 1)J_{m,k} - mJ_{m-2,k}$$

или

$$J_{m,k} = \frac{\psi'_{m-1,k}}{(k - m + 1)} + \frac{mJ_{m-2,k}}{(k - m + 1)}. \quad (3.7)$$

Заменяя в (3.3) (m) на $(m-1)$ и интегрируя находим:

$$\psi'_{m-1,k} = kJ_{mk} - RmJ_{m-2,k-2}$$

или

$$J_{m,k} = \frac{\psi'_{m-1,k}}{k} + \frac{RmJ_{m-2,k-2}}{k}. \quad (3.8)$$

Из (3.7) получаем:

$$\left\{ \begin{array}{ccc} & 0 & \\ 0 & J_{mk}\alpha_{mk} & \\ 0 & 0 & 0 \end{array} \right\} = \left\{ \begin{array}{ccc} \frac{\psi'_{m-1,k}\alpha_{mk}}{k - m + 1} & & \\ 0 & 0 & \\ 0 & 0 & \frac{-J_{m+1,k}m\alpha_{mk}}{k - m + 1} \end{array} \right\}.$$

Из (3.8) получаем:

$$\frac{d}{dx} \left\{ \begin{array}{ccc} & 0 & \\ 0 & J_{mk}\alpha_{mk} & \\ 0 & 0 & 0 \end{array} \right\} = \left\{ \begin{array}{ccc} \frac{\psi'_{m-1,k}\alpha_{mk}}{k} & & \\ & 0 & 0 \\ \frac{-J_{m+1,k-2}Rm\alpha_{mk}}{k} & & 0 \end{array} \right\}.$$

По последним формулам могут быть подготовлены необходимые для интегрирования коды $\text{ТТК}_R(J_{mk}(x))$ для всех m и k .

3.5. Инвертирование аргумента. Так мы будем называть операцию вычисления функции $\Phi(-x)$ по известной функции $\Phi(x)$. Очевидно,

$$\psi'_{mk}(-x) = (-1)^{k-m} \psi'_{mk}(x).$$

Следовательно, инвертирование аргумента заключается в инвертировании тех разрядов кода, у которых $(k-m)$ нечетно. Возникающие при этом переносы учитываются также, как и в коротких операциях.

3.6. Смещение оси ординат. Под этим термином понимается вычисление функции $\Phi(a \pm x)$ по известной функции $\Phi(x)$. Ясно, что в частном случае эта операция превращается в инвертирование аргумента. Но и при $a = \pi$ она во многом аналогична предыдущей операции. Действительно,

$$\psi'_{mk}(\pi + x) = (-1)^k \psi'_{mk}(x).$$

$$\psi'_{mk}(\pi - x) = (-1)^m \psi'_{mk}(x).$$

Следовательно, и в этом случае смещение оси ординат заключается в инвертировании некоторых разрядов исходного кода. При других значениях a требуется более сложное преобразование исходного кода. Приведем формулы для некоторых случаев:

$$\psi'_{mk}\left(\frac{\pi}{2} \pm x\right) = \mp \psi'_{k-m,k}(x).$$

$$\psi'_{mk}\left(-\frac{\pi}{2} + x\right) = (-1)^{k-m} \psi'_{k-m,k}(x).$$

$$\psi'_{mk}\left(-\frac{\pi}{2} - x\right) = (-1)^k \psi'_{k-m,k}(x).$$

Из этих формул следует, что при $a = \pm \pi / 2$ требуется перестановка разрядов исходного кода с последующим инвертированием некоторых разрядов.

4. Кодирование и декодирование тригонометрических треугольных кодов

Излагаемые здесь алгоритмы кодирования и декодирования содержат два этапа:

- предварительный, на котором вычисляются некоторые константы и строятся таблицы величин, необходимых при кодировании/декодировании функций/кодов,
- оперативный, на котором производится собственно кодирование или декодирование с использованием полученных на предварительном этапе величин.

Алгоритм 4.1. Кодирование функции $\Phi(x)$.

Предварительный этап:

Вычисление кодов $\text{TK}_R(\gamma_\omega^i)$ в соответствии с теоремой 1.3.

Заметим, что при $R \neq 4$ функции γ_ω^i могут быть представлены кодами $\text{TK}_R(\gamma_\omega^i)$ лишь приближенно. Поэтому целесообразно применять именно четверичные тригонометрические треугольные коды - см. раздел 4.1.

Оперативный этап

1. Выделение из тригонометрического ряда функции $\Phi(x)$ рядов

$$\eta^i f^i(x) = \sum_{\omega=0}^n D_\omega^i \varepsilon_\omega^i \quad (4.1)$$

- см. теорему 2.1. Числа D_ω^i должны быть целыми.

2. Вычисление чисел E_ω^i в зависимости от чисел D_ω^i в соответствии с теоремой 1.5 – см. (1.19).
3. Кодирование чисел E_ω^i , то есть образование кодов $\text{TK}_R(E_\omega^i)$.
4. Вычисление кодов $\text{TK}_R(\eta^i f^i(x))$ по известным кодам $\text{TK}_R(E_\omega^i)$ и $\text{TK}_R(\gamma_\omega^i)$ по формуле

$$\eta^i f^i(x) = \sum_{\omega=0}^n E_\omega^i \gamma_\omega^i. \quad (4.2)$$

Эта формула следует из теоремы 1.5: если функция имеет разложение (4.1), то она имеет также разложение (4.2), где коэффициенты E_ω^i и D_ω^i связаны соотношением (1.14).

5. Компонировка кода $\text{ТТК}_R(\Phi(x))$ из кодов $\text{ТК}_R(f^i(x))$ - см. определение 2.1.

Алгоритм 4.2. Кодирование функции $\Phi(x)$.

Предварительный этап: определение чисел $S^i(v, n)$ согласно теореме 1.1.

Оперативный этап

1. Выделение из тригонометрического ряда функции $\Phi(x)$ рядов (4.1) аналогично п.1 алгоритма 4.1.
2. Определение чисел A_v^i в зависимости от чисел D_ω^i в соответствии с теоремой 1.5 – см. (1.18).
3. Кодирование функций $f^i(x)$ при известных числах A_v^i , то есть образование кодов $\text{ТК}_R(f^i(x))$ - см. раздел 1.6.
4. Компонировка кода $\text{ТТК}_R(\Phi(x))$ из кодов $\text{ТК}_R(f^i(x))$ - см. определение 2.1.

Алгоритм 4.3. Декодирование кода $\text{ТТК}_R(\Phi(x))$.

Предварительный этап: определение чисел $L^i(v, n)$ согласно теореме 1.2.

Оперативный этап:

1. декомпозиция кода $\text{ТТК}_R(\Phi(x))$, то есть образование кодов $\text{ТК}_R(f^i(x))$;
2. декодирование кодов $\text{ТК}_R(f^i(x))$, то есть определение чисел A_v^i - см. раздел 1.6;
3. вычисление коэффициентов D_ω^i тригонометрического ряда функций $f^i(x)$ в зависимости от значений чисел A_v^i в соответствии с теоремой 1.5 – см. (1.17);
4. формирование тригонометрического ряда функции $\Phi(x)$, в соответствии с теоремой 2.1.

5. Погрешность кодирования тригонометрических треугольных кодов

При оценке погрешности кодирования тригонометрических рядов следует, очевидно, говорить о погрешности кодирования коэффициентов D_{ω}^i этих рядов - абсолютной и относительной.

Как указано в описании алгоритмов 4.1 и 4.2, числа D_{ω}^i должны быть целыми. Следовательно, абсолютная погрешность кодирования $\Delta=1$. Относительную погрешность кодирования δ естественно относить к величине максимального по модулю коэффициента:

$$\delta = \Delta / \left(\max_{\omega, i} |D_{\omega}^i| \right). \quad (5.1)$$

Относительная погрешность кодирования отдельных гармоник тригонометрического ряда увеличивается с уменьшением амплитуды этих гармоник. Структура кода тригонометрического ряда такова, что допустимые при заданной разрядности кода амплитуды гармоник уменьшаются с ростом их частоты. К счастью, это же характерно для большинства реальных процессов. Поэтому увеличение относительной погрешности кодирования высших гармоник вполне допустимо. Оценим величину δ в зависимости от разрядности более строго, для чего рассмотрим (без доказательства) две теоремы.

Теорема 5.1. Числа

$$p(m, k, \omega) = \sum_{v=(\omega, k-m) \max}^k (-1)^{v+k+m+\omega} 4^{k-v} C_m^{k-v} C_{2v}^{v-\omega} \quad (5.2)$$

связаны следующими рекуррентными соотношениями:

$$p(0, 0, 0) = 1;$$

$$p(m, m, \omega) = C_{2m}^{m-\omega}, \text{ if } 0 \leq \omega \leq m;$$

$$p(m, k, 0) = 2p(m, k-1, 0) - p(m, k-1, 1), \text{ if } k \geq m;$$

$$p(m, k, \omega) = \begin{bmatrix} -p(m, k-1, \omega-1) \\ +2p(m, k-1, \omega) - \\ p(m, k-1, \omega+1) \end{bmatrix}, \quad \text{if } k \geq m \text{ and } 1 \leq \omega < k;$$

$$p(m, k, k) = (-1)^{m+k}, \quad \text{if } k \geq m;$$

$$p(m, k, \omega) = 0, \quad \text{if } \omega > k \text{ or } k < m.$$

В табл. 5.1. ω приведены значения чисел $p(m, k, \omega)$ при $\omega = (0, 1, 2, 3, 4)$ соответственно и при $k < 6$ и $m < 6$. В этих же таблицах указаны числа

$\Pi_1(k, \omega)$ - сумма всех положительных чисел $p(m, k, \omega)$ при постоянных k и ω ;

$\Pi_2(k, \omega)$ - сумма всех отрицательных чисел $p(m, k, \omega)$ при постоянных k и ω ;

Таблица 5.1.0. Числа $p(m, k, 0)$.

$k \backslash m$	0	1	2	3	4	5	$\Pi_1(k, 0)$	$\Pi_2(k, 0)$
0	1						1	0
1	2	-1					4	0
2	6	-4	1				14	0
3	20	-15	6	-1			48	0
4	70	-56	28	-8	1		166	0
5	252	-210	120	-45	10	-11	584	0

Таблица 5.1.1. Числа $p(m, k, 1)$.

$k \backslash m$	0	1	2	3	4	5	$\Pi_1(k, 0)$	$\Pi_2(k, 0)$
1	2	1					1	-1
2	2	0	-1				4	-4
3	4	-1	-2	1			16	-16
4	10	-4	-4	4	-1		60	-60
5	28	-14	-8	13	-6	1	226	-226

Таблица 5.1.2. Числа $p(m, k, 2)$.

$k \backslash m$	0	1	2	3	4	5	$\Pi_1(k, 0)$	$\Pi_2(k, 0)$
2	6	4	1				2	-1
3	4	1	-2	-1			12	-4
4	6	0	-4	0	1		56	-12
5	12	-2	-8	3	2	-1	240	-32

Таблица 5.1.3. Числа $p(m, k, 3)$.

$k \backslash m$	0	1	2	3	4	5	$\Pi_1(k, 0)$	$\Pi_2(k, 0)$
3	20	15	6	1			2	-2
4	10	4	-4	-4	-1		12	-12
5	12	2	-8	-3	2	1	61	-61

Таблица 5.1.4. Числа $p(m, k, 4)$.

$k \backslash m$	0	1	2	3	4	5	$\Pi_1(k, 0)$	$\Pi_2(k, 0)$
4	70	56	28	8	1		3	-2
5	28	14	-8	-13	-6	-1	24	-12

В табл. 5.2 приведены значения чисел $p(m, k, \omega)$ при

$$k < 10, \omega < 10, m = \begin{cases} k/2, & \text{if } k - \text{even}, \\ (k-1)/2, & \text{if } k - \text{odd}. \end{cases}$$

Таблица 5.2. Числа $p(m, k, \omega)$.

m	$k \backslash \omega$	0	1	2	3	4	5	6	7	8	9
0	0	1									
0	1	-2	-1								
1	2	-2	0	1							
1	3	4	1	-2	-1						
2	4	6	0	-4	0	1					
2	5	-12	-2	8	3	-2	-1				
3	6	-20	0	15	0	-6	0	1			
3	7	40	5	-30	-9	12	5	-2	-1		
4	8	70	0	-56	0	28	0	-8	0	-1	
4	9	-140	-14	112	28	-56	-20	16	7	-2	-1

Определим еще следующие числа

$$p^0(m, k, \omega) = p(m, k, \omega); \quad (5.3)$$

$$p^1(m, k, \omega) = p(m, k, \omega) - p(m, k, \omega + 1); \quad (5.4)$$

$$p^2(m, k, \omega) = p(m, k, \omega) + p(m, k, \omega + 1); \quad (5.5)$$

$$p^3(m, k, \omega) = p(m, k, \omega) - p(m, k, \omega + 2); \quad (5.6)$$

Теорема 5.2. Функция $\eta^i \psi_{mk}$ имеет разложение вида

$$\eta^i \psi_{mk} = \sum_{\omega=0}^k q^i(m, k, \omega) \cdot \varepsilon^i, \quad (5.7)$$

где

$$q^i(m, k, \omega) = a_{\omega}^i c^i\left(\frac{R}{4}\right)^k p^i(m, k, \omega). \quad (5.8)$$

Из последней теоремы следует, что функция $F^i(x)$, имеющая разложение вида

$$F^i(x) = \eta^i \sum_{k=0}^n \sum_{m=0}^k \alpha_{mk} \psi_{mk}, \quad (5.9)$$

представима также рядом (1.8), где

$$D_{\omega}^i = \sum_{k=\omega}^n \sum_{m=0}^k q^i(m, k, \omega) \cdot \alpha_{mk}^i. \quad (5.10)$$

Используя эти теоремы, можно по знакам чисел $p^i(m, k, \omega)$ определить вид треугольного кода (расположение разрядов с минимальным и максимальным значениями), максимизирующий значение коэффициента D_{ω}^i при данном ω , и вычислить это значение. Обозначим его через M_{ω}^i . В разделе 4.5 рассмотрены величины M_{ω}^0 для кодов ТТК₄ $(F^0(x))$ при $\alpha_{mk}^i \in \{-2, -1, 0, 1\}$. Там приведены зависимости $\lg(M_{\omega}^0)$ от ω при различных значениях n и зависимости $\lg(M_{\omega}^0)$ от n при различных значениях ω .

Итак, для оценки относительной погрешности кодирования при данном n достаточно найти максимум величины M_{ω}^i : величина, обратная этому максимуму, и является относительной погрешностью δ кодирования тригонометрического ряда.

6. Укорочение тригонометрических треугольных кодов

При кодировании функций и при выполнении операций с кодами функций может возникать переполнение разрядной сетки. Для ликвидации такого переполнения вводится операция *укорочения* треугольных кодов, рассматриваемая ниже.

Назовем *значащим столбцом* треугольного кода тот столбец, в котором имеется хотя бы один разряд, отличный от нуля. Назовем, далее, номер N старшего значащего столбца *длиной* треугольного кода. Некоторая функция $\Phi_0(x)$ в общем случае может быть представлена в виде произведения

$$h_0 \cdot [\text{ТТК}_R(\Phi_0(x))], \quad (6.1)$$

где

h_0 - числовой множитель, экспонента,

$\text{ТТК}_R(\Phi_0(x))$ - код длины N_0 .

Укорочение $\text{ТТК}_R(\Phi_0(x))$ заключается в преобразовании выражения (6.1) к виду

$$h_1 \cdot [\text{ТТК}_R(\Phi_1(x))], \quad (6.2)$$

где

h_1 - новая экспонента,

$\text{ТТК}_R(\Phi_1(x))$ - код длины $N_1 < N_0$,

$$\Phi_1(x) \approx \Phi_0(x) \frac{h_0}{h_1}. \quad (6.3)$$

Последнее соотношение показывает, что укорочение в общем случае приносит некоторую погрешность.

Алгоритмически укорочение заключается в последовательном u -кратном делении исходного кода длины N_0 на параметр R до получения кода заданной длины N_1 с одновременной коррекцией числового множителя: $h_1 = h_0 R^u$. В результате деления R -го треугольного кода на параметр образуется R -ый смешанный код - см. раздел 1.4. В связи с этим возникает задача округления смешанного кода, которая более точно формулируется следующим образом:

- известен $\text{SK}_R(f(x))$;

- необходимо вычислить $TK_R(f_0(x))$ такой функции $f_0(x)$, что функция $(f(x) - f_0(x))$ имеет тригонометрический ряд вида (1.8) с коэффициентами, достаточно малыми по абсолютной величине.

Прежде, чем описывать алгоритм округления, рассмотрим функцию

$$\delta_n^i = (-1)^{n+1} \left(\gamma_n^i - (-1)^n \psi_{0n} \right) \quad (6.4)$$

или

$$\psi_{0n} = \delta_n^i + (-1)^n \gamma_n^i \quad (6.5)$$

Учитывая (1.1) из (6.4) находим:

$$\delta^i(n) = (-1)^{n+1} \sum_{v=0}^n \left[(-1)^v \left(\frac{R}{4} \right)^{n-v} S^i(v, n) \cdot \psi_{0v} \right] - (-1)^n \psi_{0n}$$

или

$$\delta^i(n) = (-1)^{n+1} \sum_{v=0}^{n-1} \left[(-1)^v \left(\frac{R}{4} \right)^{n-v} S^i(v, n) \cdot \psi_{0v} \right]. \quad (6.6)$$

Учитывая (1.3) из (6.5) находим:

$$\eta^i \psi_{0n} = \eta^i \delta_n^i + (-1)^n \left(\frac{R}{4} \right)^n a_n^i c^i \varepsilon_n^i. \quad (6.7)$$

Учитывая, далее, (1.23), отсюда получаем:

$$\eta^i \psi_{0n} = \eta^i \delta_n^i + g_n^i; \quad (6.8)$$

Таблица 6.1.

Ряд	Функция	Количество членов ряда	Обоснование
(1.9) по $\psi_{0v}(x)$	$\eta^i \psi_{0n}$	$(n+1)$	единственный n -член ряда
	$\eta^i \delta_n^i$	n	(6.6)
	$\eta^i \gamma_n^i$	$(n+1)$	(1.1)
(1.8) по $\varepsilon_\omega^i(x)$	$\eta^i \psi_{0n}$	$(n+1)$	(1.21)
	$\eta^i \delta_n^i$	n	(6.6) и (1.20)
	$\eta^i \gamma_n^i$	$(n+1)$	(1.1) и (1.20)

2. Кодирование тригонометрических рядов

Сравним количество членов в функциональных рядах функций $\eta^i \psi_{0n}$, $\eta^i \delta_n^i$, $\eta^i \gamma_n^i$ - см. табл. 6.1. Из этих замечаний следует, что ряды (1.8-11) функции $\eta^i \delta_n^i$ короче рядов (1.8-11) функции $\eta^i \psi_{0n}$ на один член. Этот факт может быть положен в основу алгоритма округления. Действительно, функция $\eta^i \psi_{0n}$ может быть заменена функцией $\eta^i \delta_n^i$ отбрасыванием старшей гармоники g_n^i тригонометрического ряда функции $\eta^i \psi_{0n}$. Эту операцию мы и будем называть *округлением тригонометрического ряда*. Очевидно, абсолютная погрешность такого округления

$$\Delta_n^i = \left(R/4\right)^n a_n^i c^i.$$

В частности, при $n > 0$ и $R = 4$ имеем: $\Delta_n^i = 2$. Итак, при округлении функции $\eta^i \psi_{0n}$ образуется округленная функция $\eta^i \delta_n^i$, ряд (1.9) которой короче на один член одноименного ряда округляемой функции. Из этого, вообще говоря, не следует, что код $\text{TK}_R(\eta^i \delta_n^i)$ округленной функции короче кода $\text{TK}_R(\eta^i \psi_{0n})$ округляемой функции. Однако *в среднем* при округлении код **R-TK** укорачивается и на этом основан алгоритм укорочения. Таким образом, округление эквивалентно отбрасыванию старших гармоник тригонометрического ряда, имеющих относительно малую амплитуду.

Итак, некоторый разряд ψ_{0n} кода $\text{TK}_R(f^i(x))$ может быть заменен кодом $\text{TK}_R(\delta_n^i)$ с погрешностью округления Δ_n^i . Аналогично, разряд $\psi_{-1,n}$ смешанного кода функции $f^i(x)$ может быть заменен кодом функции δ_n^i / R с погрешностью округления Δ_n^i / R . Смешанные коды $B_n^i = \text{SK}_R\left(\delta_n^i / R\right)$ функций δ_n^i / R могут быть получены по формуле (6.4) при известных $\eta^i \gamma_n^i$.

7. Гиперболические треугольные коды

Подробное рассмотрение именно тригонометрических треугольных кодов связано с тем, что тригонометрические ряды занимают центральное место в теории и практике функциональных рядов. Если же отвлечься от этого обстоятельства, то можно указать еще одно основание кодирования функций $Ch^2(x)$, обладающее неменьшими возможностями по сравнению с основанием $Sin^2(x)$ с точки зрения формальной теории треугольных кодов. Это утверждение основано на известной аналогии между соотношениями обычной и гиперболической тригонометрии: для получения соотношений гиперболической тригонометрии достаточно в формулах обычной тригонометрии произвести замену $Sin(x)$ на $jSh(x)$ и $Cos(x)$ на $Ch(x)$.

Треугольные коды по основанию $Ch^2(x)$ назовем гиперболическими треугольными кодами. Очевидно, свойства гиперболических треугольных кодов аналогичны свойствам тригонометрических треугольных кодов.

Глава 3. Кодирование функций многих аргументов

1. Пирамидальные коды
2. Гиперпирамидальные коды

1. Пирамидальные коды

Выше рассматривались коды функций одного аргумента. Дальнейшее обобщение приводит нас к понятию кодов функций нескольких переменных. При этом естественно применить следующую схему рассуждений: код числа имеет линейную структуру; код функции одного аргумента является плоским (треугольным, ступенчатым); очевидно, код функции двух аргументов должен быть трехмерным; код функции четырех аргументов - четырехмерным и так далее.

Вначале рассмотрим объемное обобщение треугольного кода - пирамидальный код функции двух аргументов. Пусть функция $F(x, v)$ двух аргументов x и v имеет разложение следующего вида:

$$F(x, v) = \sum_{k=0}^n \sum_{m_2=0}^k \sum_{m_1=0}^k \alpha_{m_1, m_2, k} R^k \psi_{m_1, m_2, k}, \quad (3.1.1)$$

$$\psi_{m_1, m_2, k} = y^{k-m_1} (1-y)^{m_1} z^{k-m_2} (1-z)^{m_2}, \quad (3.1.2)$$

где $\alpha_{m_1, m_2, k}$ - действительные числа;

R - целое положительное число, параметр;

$y=y(x)$ - некоторая функция аргумента x ;

$z=z(v)$ - некоторая функция аргумента v ;

m_1, m_2, k, n - целые положительные числа или нули.

Имеет место следующее соотношение:

$$R \psi_{m_1, m_2, k} = \begin{bmatrix} \psi_{m_1, m_2, k+1} + \psi_{m_1+1, m_2, k+1} + \\ \psi_{m_1, m_2+1, k+1} + \psi_{m_1+1, m_2+1, k+1} \end{bmatrix} \quad (3.1.3)$$

Определение 3.1.1. Выражение (3.1.1) называется разложением функции $F(x,v)$ по основаниям u и z с параметром R .

По аналогии с определением 1.1.1 для разложения (3.1.1) также вводятся понятия (m_1, m_2, k) -разряда, веса (m_1, m_2, k) -разряда, величины (m_1, m_2, k) -разряда.

Определение 3.1.2. Объемная матрица, имеющая вид четырехугольной пирамиды и составленная из величин $\alpha_{m_1, m_2, k}$ разложения (3.1.1) таким образом, что

- все величины α_{00k} располагаются на одной горизонтали,
- все величины $\alpha_{m_1, m_2, k}$ при $k=\text{const}$ располагаются в одной вертикальной плоскости и смещены относительно величины α_{00k} на m_1 шагов влево и m_2 шагов вверх,

называется пирамидальным кодом функции $F(x,v)$ по основаниям u и z с параметром R . Этот код обозначается как $PRK(F(x,v))$ - см рис. 3.1.1.

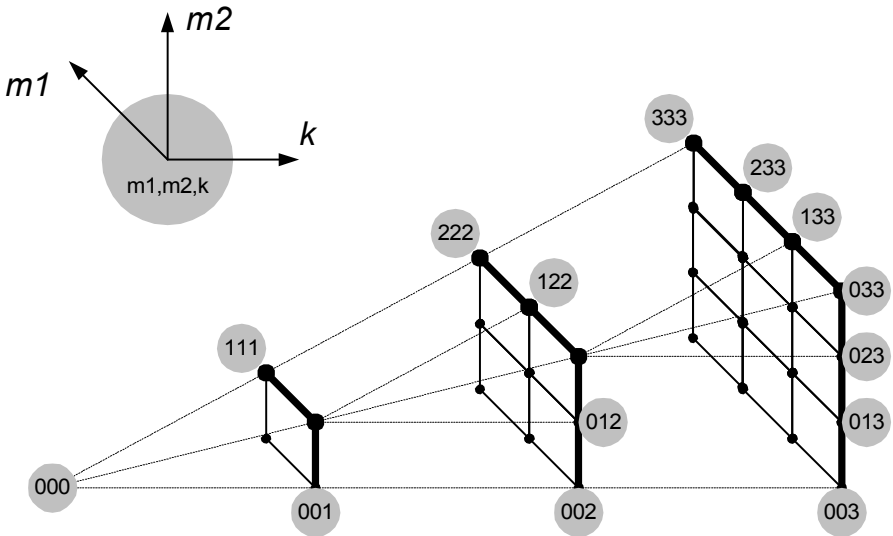
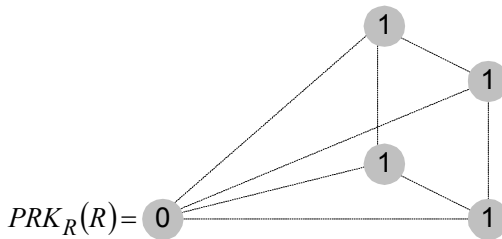


Рис. 3.1.1. Пирамидальный код

Несомненно, допускается любой поворот пирамидального кода относительно того положения, которое зафиксировано в этом определении. По аналогии с предыдущим выделим среди этих кодов R -ые пирамидальные коды $PRK_R(F(x,v))$, у которых разряды

$\alpha_{m_1, m_2, k} \in D_R$ - см. определение 1.1.3. Из (3.1.3) находим:



то есть перенос в рассматриваемом случае распространяется в четыре разряда и поэтому $R \geq 7$. Для $PRK_R(F(x, v))$ также соблюдается общий для всех позиционных кодов принцип выполнения арифметических операций.

Очевидно, структура пирамидальных кодов и алгоритмы операций с ними не накладывают ограничений на вид функций y и z . Но по соображениям, изложенным выше для кодов функций одного аргумента, и в этом случае желательно выбрать основания вида $\sin^2(x)$ и $\sin^2(v)$, поскольку они допускают кодирование тригонометрических рядов функций двух аргументов.

Тригонометрический ряд функции двух аргументов имеет следующий вид:

$$\Phi(x, v) = \sum_{\omega_1=0}^n \sum_{\omega_2=0}^n \begin{bmatrix} a_{\omega_1 \omega_2} \cos(\omega_1 x) \cos(\omega_2 v) + \\ b_{\omega_1 \omega_2} \sin(\omega_1 x) \cos(\omega_2 v) + \\ c_{\omega_1 \omega_2} \cos(\omega_1 x) \sin(\omega_2 v) + \\ d_{\omega_1 \omega_2} \sin(\omega_1 x) \sin(\omega_2 v) \end{bmatrix} \quad (3.1.4)$$

Рассуждая аналогично предыдущему, можно показать, что такой ряд представим в виде следующей суммы:

$$\Phi(x, v) = \sum_{k=0}^n \sum_{m_2=0}^k \sum_{m_1=0}^k \alpha_{m_1, m_2, k} \psi'_{m_1, m_2, k}, \quad (3.1.5)$$

где

$$\psi'_{m_1, m_2, k} = R^{k/2} \sin^{k-m_1}(x) \cos^{m_1}(x) \sin^{k-m_2}(v) \cos^{m_2}(v). \quad (3.1.6)$$

Определение 3.1.3. Объемная матрица, составленная из величин $\alpha_{m_1, m_2, k}$ разложения (3.1.5) по правилам определения 3.1.2, называется тригонометрическим пирамидальным кодом функции

$\Phi(x, v)$. Этот код обозначается как $TPRK(\Phi(x, v))$. Код, в котором $\alpha_{m_1, m_2, k} \in D_R$, называется R -ым и обозначается как $TPRK_R(\Phi(x, v))$.

Из последней формулы следует, что

$$\psi'_{m_1, m_2, k} = \left[\begin{array}{l} \psi'_{m_1, m_2, k+2} + \psi'_{m_1+2, m_2, k+2} + \\ \psi'_{m_1, m_2+2, k+2} + \psi'_{m_1+2, m_2+2, k+2} \end{array} \right] \quad (3.1.7)$$

В свою очередь, из этого соотношения следует, что **TPRK** состоит из десяти частей, причем

- каждый разряд кода принадлежит только одной из этих частей,
- индексы m_1, m_2, k разрядов, принадлежащих одной части, отличаются только на четное число и, таким образом, разряды одной части образуют пирамидальный код некоторой функции по основаниям $\sin^2(x)$ и $\sin^2(v)$,
- угловые разряды пирамидальных кодов десяти частей имеют индексы 000, 001, 011, 101, 111, 012, 102, 112, 122, 212.

Таким образом, при выполнении коротких операций с R -**TPRK** их можно рассматривать состоящими из десяти независимых кодов и выполнять операции независимо с каждой из этих частей. В остальном свойства **TPRK** и правила выполнения различных операций с ними аналогичны тому, что имеет место для кодов **ТТК** и **PRK**.

2. Гиперпирамидальные коды

При кодировании функций нескольких переменных рассматривается функция, зависящая от a аргументов x_i и имеющая разложение следующего вида:

$$F(x_1, \dots, x_i, \dots, x_a) = \sum_{k=0}^n \sum_{m_a=0}^k \dots \sum_{m_1=0}^k \left(\alpha_{m_1, \dots, m_a, k} \prod_{i=1}^a y_i^{k-m_i} (1-y_i)^{m_i} \right) \quad (3.2.1)$$

где

$\alpha_{m_1, \dots, m_a, k}$ - действительные числа;

R - целое положительное число, параметр;

$y_i = f_i(x_i)$ - некоторая функция аргумента x_i ;

m_i, k, n - целые положительные числа или нули.

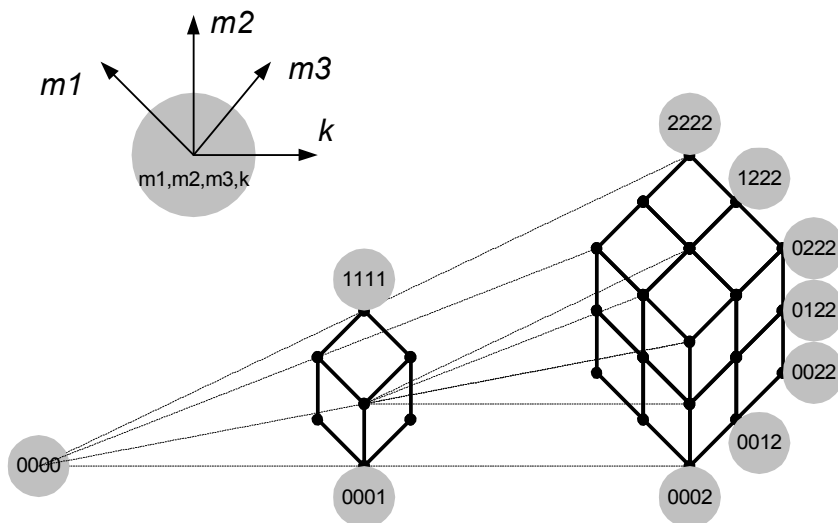


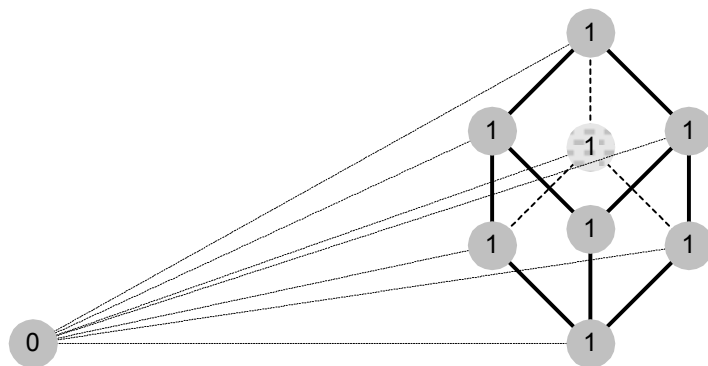
Рис. 3.2.1. Гиперпирамидальный код.

Естественно называть формулу (3.2.1) разложением функции $F(x_1, \dots, x_i, \dots, x_a)$ по основаниям y_i , $i = \overline{1, a}$ с параметром R ; объемную матрицу, составленную из величин $\alpha_{m_1, \dots, m_a, k}$

разложения (3.2.1) - гиперпирамидальным кодом этой функции и так далее. Очевидно, разложение (3.2.1) при $a=1$ превращается в разложение (1.1.1), а гиперпирамидальный код - в треугольный. При $a=2$ разложение (3.2.1) превращается в разложение (3.1.1), а гиперпирамидальный код - в пирамидальный. При дальнейшем увеличении структура кода усложняется: вместо k -столбца в треугольном коде и k -квадрата в пирамидальном коде появляется a -мерный k -куб. В качестве иллюстрации на рис. 3.2.1 изображен гиперпирамидальный код при $a = 3$.

Очевидно, что все сказанное относительно пирамидального кода обобщается и на гиперпирамидальные коды. Дополнительно отметим только два факта.

В гиперпирамидальном коде перенос распространяется в a -мерный куб, содержащий 2^a разрядов. Следовательно параметр $R \geq (2^{a-1} - 1)$. Ниже представлен код параметра R :



Гиперпирамидальные коды тригонометрических рядов - тригонометрические гиперпирамидальные коды функций нескольких переменных строятся по аналогии с предыдущим и состоят из нескольких независимых частей - гиперпирамидальных кодов по основаниям $\sin^2(x_i)$, $i = \overline{1, a}$. Число таких частей равно $(1 + 3^a)$.

Глава 4. Четверичные тригонометрические треугольные коды

В данной главе подробно рассматриваются алгоритмы операций с четверичными тригонометрическими треугольными кодами, описанные в общем случае в предыдущих главах. Целью этой главы является конкретизация теории кодирования функций до такой степени, которая допускает постановку задачи технического проектирования соответствующих компьютеров. Предпочтение, отдаваемое четверичным кодам, вызвано следующими обстоятельствами:

1. не существует двоичных треугольных кодов,
2. схемы для операций с четверичными кодами легко реализуются на двоичных элементах,
3. большинство формул, используемых в алгоритмах кодирования и декодирования тригонометрических треугольных кодов, приобретают для четверичных кодов более простой вид.

В связи с тем, что далее будут рассматриваться исключительно четверичные коды, символ R в обозначениях кода будет опускаться. Условимся далее, что множество значений величин разрядов $D_R = \{-2, -1, 0, 1\}$.

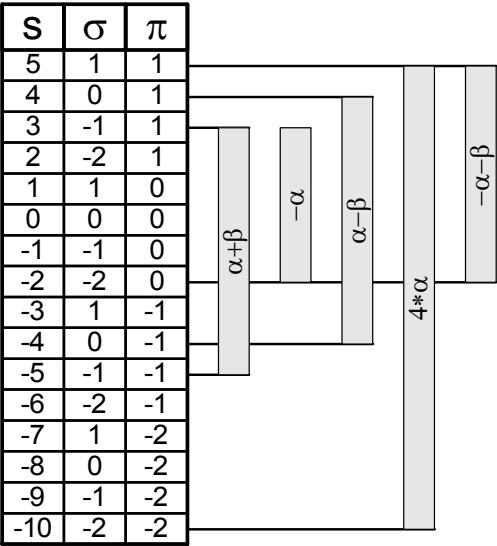
1. Арифметические операции с четверичными треугольными кодами

В общем виде арифметические операции с треугольными кодами описаны выше. Здесь будут указаны лишь особенности четверичных кодов.

1.1. Алгебраическое сложение.

Вначале рассмотрим алгебраическое сложение четверичных кодов чисел $K_4(A)$, которые составляют прямоугольный код. Вообще говоря, такие коды при $R=4$ и $D_R=\{-2,-1,0,1\}$ эквивалентны двоичным кодам вещественных чисел по основанию (-2) . Тем не менее мы рассмотрим (для дальнейших аналогий) табл. 1.1, которая описывает алгебраическое сложение кодов $K_4(A)$ при $R=4$ и $D_R=\{-2,-1,0,1\}$. Эта таблица построена с использованием формул (1.2.1), (1.2.3) и удовлетворяет условиям полноты.

Таблица 1.1.

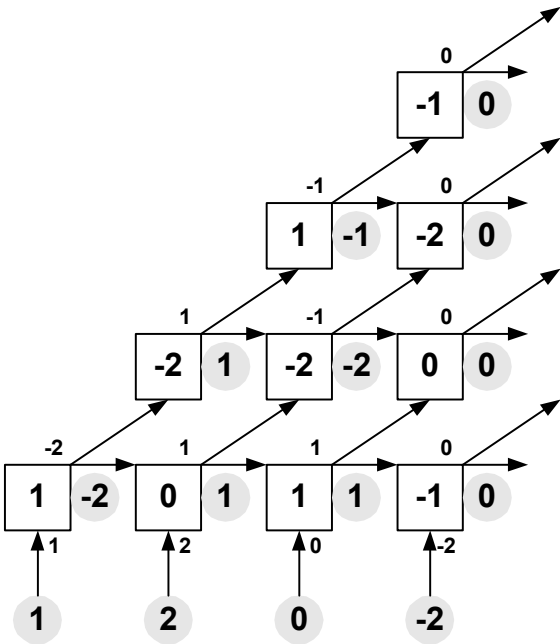


В табл. 1.2 перечислены варианты алгебраического сложения треугольных кодов при $R=4$ и $D_R=\{-2,-1,0,1\}$. Эта таблица

S	σ	π
8	0	2
7	-1	2
6	-2	2
5	1	1
4	0	1
3	-1	1
2	-2	1
1	1	0
0	0	0
-1	-1	0
-2	-2	0
-3	1	-1
-4	0	-1
-5	-1	-1
-6	-2	-1
-7	1	-2
-8	0	-2
-9	-1	-2
-10	-2	-2
-11	1	-3
-12	0	-3
-13	-1	-3
-14	-2	-3

$$SK_4(f(x)) = \begin{pmatrix} 0 \\ -1 \\ 0 \\ 1 \\ -2 \\ 0 \\ 1 \\ -2 \\ -2 \\ 0 \end{pmatrix}$$

По табл. 1.2 строим схему распространения переносов, где в кругах записаны разряды α_{mk} исходного кода, в квадратах - разряды σ_{mk} произведения, а рядом со стрелками - переносы π_{mk} из разряда, откуда выходят стрелки.



Следовательно, код произведения

$$\begin{matrix} & & & -1 \\ & & 1 & -2 \\ & -2 & -2 & 0 \\ \text{TK}_4(4f(x)) = & 1 & 0 & 1 & -1 \end{matrix}$$

1.2. Деление на параметр.

В данном случае $R=4$, $D_R=\{-2,-1,0,1\}$ и $(\alpha_{mk}, \pi'_{mk}, \pi''_{mk}) \in D_R$.

При этом по формуле (1.4.1) находим: $\frac{-11}{4} \leq S_{mk} \leq \frac{7}{4}$. Каждое из этих значений представляется в виде удовлетворяет условию (1.4.2), где σ_{mk} - целое число. В табл. 1.3 перечислены все возможные значения и соответствующие им значения.

Важно отметить, что в этой таблице $-3 \leq \sigma_{mk} \leq 2$. Это означает, что разряды частного должны иметь 2 бита (а не 2, как в других случаях) и после деления должно выполняться распространение переносов из разрядов, не удовлетворяющих условию $\sigma_{mk} \in D_R$.

Таблица 1.3.

$4S_{mk}$	π_{mk}	σ_{mk}
-11	1	-3
-10	-2	-2
-9	-1	-2
-8	0	-2
-7	1	-2
-6	-2	-1
-5	-1	-1
-4	0	-1
-3	1	-1
-2	-2	0
-1	-1	0
0	0	0
1	1	0
2	-2	1
3	-1	1
4	0	1
5	1	1
6	-2	2
7	-1	2

Пример 1.2. Деление на параметр. Рассмотрим код

$$\begin{array}{r}
 1 \\
 0 \ 0 \\
 0 \ 0 \ 0 \\
 \text{TK}_4 = 0 \ 0 \ 0 \ 0
 \end{array}$$

После деления первой строки получаем:

$$\begin{array}{r}
 1/4 \\
 0 \ 0 \\
 0 \ 0 \ 0 \\
 \text{TK}_4 = 0 \ 0 \ 0 \ 0
 \end{array}$$

После передачи переноса из третьей строки получаем:

$$\begin{array}{cccc}
 & & 0 & \\
 & & 1 & (-1/4) \\
 & 0 & 0 & 0 \\
 \text{TK}_4 = & 0 & 0 & 0 & 0
 \end{array}$$

После передачи переноса из второй строки получаем:

$$\begin{array}{cccc}
 & & 0 & \\
 & & 1 & 0 \\
 & 0 & -1 & (1/4) \\
 \text{TK}_4 = & 0 & 0 & 0 & 0
 \end{array}$$

После передачи переноса из первой строки получаем:

$$\begin{array}{cccc}
 & & 0 & \\
 & & 1 & 0 \\
 & 0 & -1 & 0 \\
 \text{TK}_4 = & 0 & 0 & 1 & (-1/4)
 \end{array}$$

После передачи переноса из нулевой строки получаем частное

$$\begin{array}{cccc}
 & & 0 & \\
 & & 1 & 0 \\
 & 0 & -1 & 0 \\
 \text{TK}_4 = & 0 & 0 & 1 & 0
 \end{array}$$

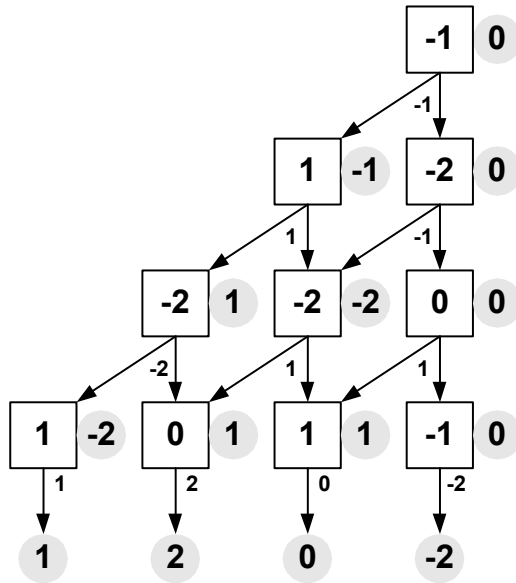
и остаток

$$\Lambda K_4 = 0 \ 0 \ 0 \ (1/4)$$

Пример 1.3. Деление на параметр кода функции

$$\begin{array}{cccc}
 & & & -1 \\
 & & & 1 & -2 \\
 & & -2 & -2 & 0 \\
 \text{TK}_4(F(x)) = & 1 & 0 & 1 & -1
 \end{array}$$

По табл. 1.3 строим схему распространения переносов, где в квадратах записаны разряды α_{mk} делимого, в кругах – разряды частного σ_{mk} , между стрелками – переносы π_{mk} из разряда, расположенного над стрелками.



Следовательно, смешанный код частного

$$SK_4(F(x)/R) = \begin{matrix} & & & 0 \\ & & -1 & 0 \\ & 1 & 1 & 0 \\ -2 & 0 & 2 & 0 \\ 1 & -2 & -2 & 0 \end{matrix}$$

Сравнивая примеры 1.2 и 1.3 замечаем, что в них описаны обратные действия – коды на рисунках совпадают, рисунки различаются направлением стрелок-переносов и

$$F(x) = Rf(x), \quad SK_4(F(x)/R) = SK_4(f(x)), \quad TK_4(R \cdot f(x)) = TK_4(F(x))$$

1.3. Умножение.

Умножение четверичных треугольных кодов не имеет особенностей по сравнению с общим случаем.

Пример 1.4. Умножение. Найдем код $TK_4(z(x))$ произведения $\mathcal{Z}=\mathcal{W}$ функций, коды $TK_4(u(x))$ и $TK_4(v(x))$ которых приведены в примерах 2.3 и 2.4 соответственно.

									0
								0	0
							0	0	0
						0	0	0	1
				0	-2	0	1	-1	1
			0	-2	0	0	-1	-1	1
		-2	-1	-1	0	-2	-2	1	
$TK_4(z(x))$		-2	1	1	-1	0	-1	0	0
	-2	1	1	-1	0	0	-1	0	0

В примере 2.5 этот код декодирован и тем самым найдено произведение $\mathcal{Z}=\mathcal{W}$ функций, определенных в примерах 2.3 и 2.4.

2. Кодирование и декодирование четверичных треугольных кодов

В общем виде кодирование и декодирование четверичных треугольных кодов описано в разделе 2.4. Здесь будут указаны лишь особенности четверичных кодов.

2.1. Кодирование и декодирование четверичных треугольных кодов

Рассмотрим примеры.

Пример 2.1. Кодирование и декодирование при $R = 4$, $D_R = \{-2, -1, 0, 1\}$. Вначале рассмотрим декодирование треугольного кода $TK(F(x))$ из примера 1.3. При делении кода $TK(F_0(x)) = TK(F(x))$ на 4 образуется код частного $TK(F_1(x))$ и код остатка $AK(f_0(x))$. При делении кода $TK(F_1(x))$ образуется код частного $TK(F_2(x))$ и код остатка $AK(f_1(x))$. При делении кода $TK(F_2(x))$ образуется код частного $TK(F_3(x))$ и код остатка $AK(f_2(x))$. Последнее деление кода $TK(F_3(x))$ дает нулевое частное и остаток $TK(f_3(x)) = TK(F_3(x))$. В обратном направлении этот же пример иллюстрирует процесс кодирования прямоугольного кода последовательным умножением его на параметр.

$TK(F_0(x)) = \begin{matrix} & & & 0 \\ & & 0 & 0 \\ & 0 & -1 & 0 \\ 1 & 0 & 1 & -1 \end{matrix}$					$TK(F_1(x)) = \begin{matrix} & & & 0 \\ & & -1 & 0 \\ & 1 & -1 & 1 \\ -2 & -2 & -2 & 1 \end{matrix}$				
$AK(f_0(x)) = \begin{matrix} & & & 0 \\ & & -1 & 0 \\ & 1 & -1 & 1 \\ -2 & 1 & -2 \end{matrix}$					$AK(f_1(x)) = \begin{matrix} & & & 0 \\ & & 0 & 0 \\ -1 & 0 & 0 \\ 1 & 0 & 1 \end{matrix}$				

Следовательно,

$$\begin{matrix} -1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ -2 & 1 & -2 & 0 \\ 1 & -2 & 0 & -2 \end{matrix}$$
$$A_{\psi} = \begin{matrix} -55 & 2 & 8 & -2 \end{matrix}$$

$$\begin{aligned} &= \Lambda K(f_3(x)) \\ &= \Lambda K(f_2(x)) \\ &= \Lambda K(f_1(x)) \\ &= \Lambda K(f_0(x)) \end{aligned}$$

Итак, $F(x) = -55 + 2\psi_{01} + 8\psi_{02} - 2\psi_{03}$.

Пример 2.2. Преобразование кода $TK(F(x))$ в код $PK(F(x))$ последовательным делением на параметр.

								0
							0	1
						-1	-2	1
					1	0	-1	1
				-1	1	0	0	1
			-1	-1	1	-2	1	1
		-1	-1	-1	1	-1	-1	1
	0	-1	-1	1	1	0	1	0
TK =	0	-2	0	-1	0	1	0	0

								0
							0	1
						-1	-2	0
					1	1	1	1
				-1	0	-1	-1	1
			-1	0	1	-1	-2	1
		-1	0	-1	1	1	1	0
	0	0	-1	-2	-2	1	0	0
CK1 =	0	-2	0	0	-2	0	0	0

								0
							0	0
						0	1	0
					-1	-1	0	1
				1	-2	0	-2	1
			-1	-1	-2	1	-1	1
		-1	1	-2	-1	-1	-1	1
	-1	1	-2	-1	-1	0	-2	1
	0	1	-2	0	-1	-2	1	-2
CK2 =	0	-2	0	0	-2	0	0	0

						0	
						0	0
					0	1	0
				-1	-1	-1	1
			1	-1	1	-1	0
		-1	-1	-1	0	0	1
	-1	-1	0	0	-1	-1	0
	-1	-2	0	-1	-1	1	-1
	0	1	-2	0	-1	-2	1
CK3=	0	-2	0	0	-2	0	0

						0	
						0	0
					0	0	0
				0	1	0	1
			-1	-1	-1	-2	1
		1	0	-2	-2	0	1
	-1	-2	0	-2	-1	-2	1
	-1	0	-2	1	1	0	-2
	-1	-2	0	-1	-1	1	-1
	0	1	-2	0	-1	-2	1
CK4=	0	-2	0	0	-2	0	0

						0	
						0	0
				0	0	0	
			0	1	0	1	
		-1	-1	-2	-2	1	
	1	0	-1	0	-2	1	
	-1	1	-1	-1	-1	0	0
	-1	0	-2	1	1	0	-2
	-1	-2	0	-1	-1	1	-1
	0	1	-2	0	-1	-2	1
CK5=	0	-2	0	0	-2	0	0

						0	
						0	0
			0	0	0		
		0	1	-1	1		
	-1	-1	1	-2	0		
	1	1	0	-1	0	1	0
	-1	1	-1	-1	-1	0	0
	-1	0	-2	1	1	0	-2
	-1	-2	0	-1	-1	1	-1

	0	1	-2	0	-1	-2	1	-2
СК6=	0	-2	0	0	-2	0	0	0

	0	1	-1	1	0	0	0	0
	-1	-1	0	-1	-1	0	0	0
	1	1	0	-1	0	1	0	0
	-1	1	-1	-1	-1	0	0	0
	-1	0	-2	1	1	0	-2	0
	-1	-2	0	-1	-1	1	-1	1
	0	1	-2	0	-1	-2	1	-2
PK=	0	-2	0	0	-2	0	0	0

A0	-3408							
A1	13538							
A2	-16776							
A3	11056							
A4	-4310							
A5	1032							
A6	-140							
A7	8							

Пример 2.3. Кодирование функции

$u(x)=-4+7 \psi_{01}+12 \psi_{02}-6 \psi_{03}+\psi_{04}:$

					0
					0
			0	1	1
TK ₄ (u(x))		-1	-2	1	1
	0	-2	-2	-2	1

Пример 2.4. Кодирование функции

$v(x)=10+48 \psi_{01}-36 \psi_{02}+8 \psi_{03}:$

					0
				0	1
			1	0	1
TK ₄ (v(x))		-1	-2	-2	1
	-2	-1	1	-2	1

Пример 2.5а. Декодирование кода

TK ₄ =			-1
		1	-1
	1	0	-2

Выполним последовательное деление кода TK_4 на параметр:

			0
		-1	0
	1	-1	0
$\Delta K1=$	1	-1	-2
			0
		0	0
	-1	0	0
$\Delta K2=$	1	1	0
			0
		0	0
	0	0	0
$\Delta K3=$	-1	0	0

Формируем прямоугольный код:

	-1	0	0	$\Delta K3$
PK_4	1	1	0	$\Delta K2$
	1	-1	-2	$\Delta K1$

Декодирование прямоугольного кода PK_4 дает функцию $(-11 + 3\psi_{01} - 2\psi_{02})$

Пример 2.5. Декодирование кода $TK_4(z(x))$ из примера 1.4.

Выполним последовательное деление кода $TK_4(z(x))$ на параметр:

				0	0	0	1
			0	0	1	-1	0
		0	-2	1	-2	0	1
		-2	0	-2	1	0	-2
	-2	0	0	-2	1	-1	-2
$\Delta K1=$	0	-2	0	0	-2	0	-1

				0	0	0	1	0
			0	0	1	-2	-1	0
		0	-2	1	1	1	1	0
	-2	1	0	0	-1	1	-2	1
$\Delta K2=$	-2	-2	-2	-2	1	0	1	-2

			0	0	1	1	0	0
		0	0	-2	-1	-1	-1	0
	0	-2	-2	1	-1	0	0	0
$\Delta K3=$	-2	1	-2	-1	-1	-2	-1	1

	0	0	1	1	0	1	0	
	0	0	-2	-1	-2	-2	1	0
AK4=	0	-2	-2	-1	1	-2	-1	0

	0	0	1	1	-1	1	0	0
ΛK5=	0	0	-2	-2	1	-2	0	0

Формирование прямоугольного кода:

	0	0	1	1	-1	1	-1	0	ΛK6
	0	0	-2	-2	1	-2	0	0	ΛK5
	0	-2	-2	-1	1	-2	-1	0	ΛK4
	-2	1	-2	-1	-1	-2	-1	1	ΛK3
PK ₄ (z(x))	-2	-2	-2	-2	1	0	1	-2	ΛK2
	0	-2	0	0	-2	0	0	-1	ΛK1

Декодирование прямоугольного кода PK₄(z(x)):

$$z = \begin{bmatrix} -40 - 122\psi_{01} + 344\psi_{02} + 424\psi_{03} \\ -718\psi_{04} + 352\psi_{05} - 76\psi_{06} + 7\psi_{07} \end{bmatrix}$$

2.2. Кодирование и декодирование четверичных тригонометрических треугольных кодов

Рассмотрим вначале функции γ_{ω}^i – см. теорему 2.1.3. Можно убедиться в следующих свойствах функций γ_{ω}^i при $R=4$ и $n>1$:

$$\psi_{01} \cdot \gamma'_{\omega} = \gamma_{\omega} + \gamma_{\omega-1},$$

$$\gamma_{\omega} = \gamma'_{\omega} + \gamma'_{\omega-1},$$

$$\psi_{01} \cdot \gamma'''_{\omega} = \gamma''_{\omega} + \gamma''_{\omega-1},$$

$$\gamma''_{\omega} = \gamma'''_{\omega} + \gamma'''_{\omega-1}.$$

Из этих формул при $n>2$ следует:

$$\gamma_{\omega}^i = (\psi_{01} - 2)\gamma_{\omega-1}^i + \gamma_{\omega-2}^i.$$

Четверичные коды этих функций упрощаются по сравнению с общим случаем. Коды $\text{TK}_4(\gamma_{\omega}^i)$ для младших значений n приведены в табл. 2.1. На использовании заранее рассчитанных таблиц кодов $\text{TK}_4(\gamma_{\omega}^i)$ основан алгоритм 2.4.1 кодирования функций.

Таблица 2.1.

n	$TK_4(\gamma_\omega)$						$TK_4(\gamma'_\omega)$					
1		1						1				
	-2	0					-1	0				
2			0						0			
		1	-1					1	-1			
	-2	1	0				1	0	0			
3				0						0		
			0	1					0	1		
		0	-2	0				-2	-1	0		
	-2	1	0	0			1	0	0	0		
4					0						0	
				0	0					0	0	
			0	-1	1				1	-1	1	
		1	0	-1	0			-2	-2	0	0	
	-2	1	0	0	0		1	0	0	0	0	
5						0						0
					0	0					0	0
				0	1	0				0	1	0
			0	-2	-1	1			-1	-1	-2	1
		0	-2	-1	0	0		1	0	-1	0	0
	-2	1	0	0	0	0	1	0	0	0	0	0
n	$TK_4(\gamma''_\omega)$						$TK_4(\gamma'''_\omega)$					
1		0						1				
	1	-1					-2	0				
2			0						0			
		0	-1					1	-1			
	1	1	0				-1	1	0			
3				0						0		
			0	1					0	1		
		0	-2	0				-1	-2	0		
	-1	-2	1	0			0	1	0	0		
4					0						0	
				0	0					0	0	
			0	0	1				0	-1	1	
		0	-2	-1	0			1	-1	-1	0	
	1	-2	1	0	0		1	1	0	0	0	
5						0						0
					0	0					0	0
				0	0	0				0	1	0
			0	1	-2	1			0	-2	-1	1
		0	0	1	-1	0		-1	1	-2	0	0
	-1	-1	1	0	0	0	-2	-2	1	0	0	0

Рассмотрим примеры кодирования и декодирования четверичных кодов. Некоторые числа, которые будут использоваться в этих примерах, сведены в табл. 2.2-4.

Таблица 2.2. К примерам кодирования и декодирования.

i	0	1	2	3	3	3
ω	0	1	1	0	1	2
$TK_4(\gamma_{\omega}^i)$	$\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}$	$\begin{smallmatrix} 1 & 0 \\ -1 & 0 \end{smallmatrix}$	$\begin{smallmatrix} 0 & 1 \\ 1 & -1 \end{smallmatrix}$	$\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}$	$\begin{smallmatrix} 1 & 1 \\ -2 & 0 \end{smallmatrix}$	$\begin{smallmatrix} 1 & 1 \\ -1 & 0 \end{smallmatrix}$
D_{ω}^i	2	2	-2	-4	-2	-2
E_{ω}^i	2	-1	1	-2	1	-1
$TK_4(E_{\omega}^i)$	$\begin{smallmatrix} 1 & 1 \\ -2 & 1 \end{smallmatrix}$	$\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}$	$\begin{smallmatrix} -1 & 1 \\ -1 & 1 \end{smallmatrix}$	$\begin{smallmatrix} -2 & 1 \\ -2 & 1 \end{smallmatrix}$	$\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}$	$\begin{smallmatrix} -1 & 1 \\ -1 & 1 \end{smallmatrix}$

Таблица 2.3. К примерам кодирования и декодирования.

i	0	1	1	2	2	3	3
v	0	0	1	0	1	0	1
A_v^i	2	-3	1	2	-2	3	-2

Таблица 2.4. К примерам кодирования и декодирования.

i	0	1	3	3
$PK_4(f^i(x))$	$\begin{smallmatrix} 1 & 0 \\ -2 & 0 \end{smallmatrix}$	$\begin{smallmatrix} -1 & 0 \\ 1 & 1 \end{smallmatrix}$	$\begin{smallmatrix} 0 & 0 \\ 1 & -1 \end{smallmatrix}$	$\begin{smallmatrix} 1 & 0 \\ -1 & -2 \end{smallmatrix}$
$TK_4(f^i(x))$	$\begin{smallmatrix} 1 & 1 \\ -2 & 1 \end{smallmatrix}$	$\begin{smallmatrix} -1 & 1 \\ 1 & 0 \end{smallmatrix}$	$\begin{smallmatrix} 0 & 1 \\ -1 & -1 \end{smallmatrix}$	$\begin{smallmatrix} -1 & 1 \\ 1 & -1 \end{smallmatrix}$

Пример 2.6. Кодирование функции по алгоритму 2.4.1 при $R = 4$ и $\alpha \in \{-2, -1, 0, 1\}$.

Рассмотрим функцию

$$\Phi(x) = 2 - 4\sin 2x - 2\sin 3x + 2\cos 3x + 2\sin 4x - 2\sin 6x$$

1. выделяем из ряда $\Phi(x)$ функции $\eta^i f^i(x)$, равные при $i = 0, 1, 2, 3$ соответственно (2) , $(2\sin 3x)$, $(-2\cos 3x)$, $(-4\sin 2x - 2\sin 4x - 2\sin 6x)$; отсюда находим числа D_{ω}^i , приведенные в табл. 2.2;

2. находим числа E_{ω}^i при известных числах D_{ω}^i по формуле (2.1.19) - см. табл. 2.2;
3. кодируем числа E_{ω}^i и получаем коды $TK_4(E_{\omega}^i)$;
4. пользуясь формулой (2.4.2), известными кодами $TK_4(\gamma_{\omega}^i)$ и вычисленными кодами $TK_4(E_{\omega}^i)$, вычисляем коды $TK_R(\eta^i f^i(x))$, равные при $i = 0, 1, 2, 3$ соответственно:

$$\begin{Bmatrix} 1 \\ -2 \end{Bmatrix}, \begin{Bmatrix} -1 \\ 1 \end{Bmatrix}, \begin{Bmatrix} 0 \\ 1 \end{Bmatrix}, \begin{Bmatrix} 0 \\ -2 \end{Bmatrix} + \begin{Bmatrix} 1 \\ -2 \end{Bmatrix} + \begin{Bmatrix} -1 \\ 1 \end{Bmatrix} = \begin{Bmatrix} -1 \\ 1 \end{Bmatrix}$$
 - см. также табл. 2.4;
5. компонуем код

$$\begin{array}{ccccccc}
 & & & & & & 0 \\
 & & & & & & 1 & -1 \\
 & & & & & 1 & -1 & 0 \\
 & & & & -2 & 1 & -1 & -1 \\
 TTK_4(\Phi(x)) = & -2 & 1 & 1 & 0 & 0
 \end{array}$$

Пример 2.7. Кодирование функции по алгоритму 2.4.2 при условиях примера 2.6

1. находим числа D_{ω}^i - см. п. 1 примера 2.6
2. при известных числах D_{ω}^i и $S^i(v, \omega)$ находим числа A_v^i по формуле (2.1.18) - см. табл. 2.3;
3. кодируем функции $f^i(x)$, заданные числами A_v^i , то есть формируем коды $PK_4(f^i(x))$, а затем - коды $TK_4(f^i(x))$ - см. табл. 2.4; эти коды совпадают с теми, которые получены в п. 4 примера 2.6;
4. компонуем код $TK_4(\Phi(x))$ - см. п. 5 примера 2.6.

Пример 2.8. Декодирование кода по алгоритму 2.4.3.

Рассмотрим код $TK_4(\Phi(x))$, полученный в примере 2.6.

1. декомпируем код $TK_4(\Phi(x))$, что приводит к образованию кодов $TK_4(f^i(x))$ - см. табл. 2.4;

2. декодируем полученные коды для получения чисел A_v^i - см. табл. 2.3;
3. при известных числах A_v^i и $L^i(v, n)$ вычисляем числа D_ω^i по формуле (2.1.17) - см. табл. 2.2;
4. формируем тригонометрический ряд функции $\Phi(x)$ при известных числах D_ω^i - см. условие примера 2.6.

Пример 2.9. Кодирование функции в ТТК по алгоритму 2.4.1.

Рассмотрим функцию из примера 2.2.2 с использованием известных кодов $TK_4(\gamma_\omega^i)$, приведенных в табл. 2.1. Применяя формулу (2.1.19) для преобразования ряда (2.1.8) в ряд (2.1.10), из примера 2.2.2 находим:

$$F(x) = 32 + 21\gamma_1^0 - 4\gamma_2^0 + 2\gamma_3^0 - \gamma_4^0,$$

$$F'(x) = 2\text{Sin}x(128 - 5\gamma_1^1 + \gamma_2^1),$$

$$F'(x) = 2\text{Cos}x(128 + 5\gamma_1^2 + \gamma_2^2)$$

$$F'''(x) = 2\text{Sin}x\text{Cos}x(42 - 16\gamma_1^3 + 12\gamma_2^3 - 8\gamma_3^3)$$

	32				
$21\gamma_1^0$		0			
	-42	21			
$-4\gamma_2^0$			0		
		-4	4		
	8	-4	0		
$2\gamma_3^0$				0	
			0	2	
		0	-4	0	
	-4	2	0	0	
$-\gamma_4^0$					0
				0	0
			0	1	-1
		-1	0	1	0
	2	-1	0	0	0
$F^0(x)$					0
				0	0
			0	3	-1
		-5	0	1	0
	-4	18	0	0	0

	128		
$-5\gamma_1^1$		5	
	-5	0	
γ_2^1			0
		1	-1
	1	0	0
$\frac{F^1(x)}{2\text{Sin}x}$			0
		6	-1
	124	0	0

	128		
$5\gamma_1^2$		0	
	-5	5	
γ_2^2			0
		0	-1
	1	1	0
$\frac{F^2(x)}{2\text{Cos}x}$			0
		0	-1
	124	6	0

	42			
$-16\gamma_1^3$		0		
	32	-16		
$12\gamma_2^3$			0	
		12	-12	
	-12	12	0	
$-8\gamma_3^3$				0
			0	-8
		8	16	0
	0	-8	0	0
$\frac{F^3(x)}{2\text{Sin}2x}$				0
			0	-8
		20	4	0
	62	-12	0	0

Кодируя полученные $\text{TK}\left(F^i(x)/\eta^i\right)$ в $\text{TK}_4^i = \text{TK}_4\left(F^i(x)/\eta^i\right)$,
находим:

					0
				0	1
			-1	0	1
TK ₄ ⁰ =		-2	-1	-1	1
	0	1	0	1	0

								0
							0	0
						0	0	0
					1	1	1	1
			-2	-1	-2	-2	1	
		1	-2	0	-2	1	0	
TK ₄ ¹ =		1	0	-2	-1	1	0	0
	0	-1	0	-2	1	0	0	0

								0
							0	0
						0	0	0
					1	1	1	1
			-2	-1	-2	-2	1	
		0	-2	0	-2	1	0	
TK ₄ ² =		-1	0	-2	-1	1	0	0
	0	1	1	-2	1	0	0	0

					1
				-2	1
			1	-2	0
TK ₄ ³ =		0	-2	0	1
	2	0	1	0	0

Расширяя коды TK₄ⁱ, получаем тригонометрические четверичные коды:

TTK₄(F(x))=

								0
							0	0
						0	0	1
					0	0	0	0
			-1	0	0	0	0	1
		0	0	0	0	0	0	0
	-2	0	1	0	-1	0	1	
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0

$$\text{ТТК}_4(F''(x)) =$$

								1	0	1	0	1	0	1
								0	0	0	0	0	0	0
						-2	0	-1	0	-2	0	-2	0	1
					0	0	0	0	0	0	0	0	0	0
				0	0	-2	0	0	0	-2	0	1	0	0
			0	0	0	0	0	0	0	0	0	0	0	0
		-1	0	0	0	-2	0	-1	0	1	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	-2	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

$$\text{ТТК}_4(F'(x)) =$$

								0	1	0	1	0	1	0	1
								0	0	0	0	0	0	0	0
					0	-2	0	-1	0	-2	0	-2	0	1	
				0	0	0	0	0	0	0	0	0	0	0	0
			0	1	0	-2	0	0	0	-2	0	1	0	0	
		0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0	1	0	0	0	-2	0	-1	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	-1	0	0	0	-2	0	1	0	0	0	0	0	0

$$\text{ТТК}_4(F'''(x)) =$$

										0					
										0	1				
									0	0	0				
								0	-2	0	1				
						0	0	0	0	0	0				
				0	1	0	-2	0	1						
			0	0	0	0	0	0	0	0	0				
		0	0	0	0	-2	0	0	0	0	1				
	0	0	0	0	0	0	0	0	0	0	0	0			
0	0	-2	0	0	0	1	0	0	0	0	0	0			
0	0	0	0	0	0	0	0	0	0	0	0	0			

Пример 2.10. Коды функций ε_ω^i . Еще раз отметим, что функции ε_ω^i имеют простые коды $\text{ТТК}_4(\varepsilon_\omega^i)$. В частности, из

табл. 2.2.1 следует, что $\text{ТТК}_4(2\text{Sin}(x)) =$

0 0

1 0 0 1. Далее

$\underline{\text{ТТК}}_4(2\text{Cos}(x)) = 0 \ 0 \ 0 \ \underline{\text{ТТК}}_4(2\text{Sin}(2x)) = 0 \ 0 \ 0$

коды $\text{ТТК}_4(\varepsilon_{\omega}^i)$ могут быть получены по одному из вышеописанных алгоритмов кодирования. Другой способ – использование простых формул тригонометрии. Например, из

0

$2\text{Cos}(2x) = 2 - 4\text{Sin}^2(x),$ 0 0, следует,

$\text{ТТК}_4(4\text{Sin}^2(x)) = 0 \ 0 \ 1$

что $\underline{\text{ТТК}}_4(2\text{Cos}(2x)) = \begin{bmatrix} 0 \\ 0 \ 0 \\ 2 \ 0 \ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \ 0 \\ -2 \ 0 \ 0 \end{bmatrix}.$

Далее, из $2\text{Sin}(3x) = 6\text{Sin}(x) - 8\text{Sin}^3(x),$

$\text{ТТК}_4(6\text{Sin}(x)) = \begin{bmatrix} 0 \\ 0 \ 3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \ 0 \ 0 \\ 0 \ -1 \ 0 \ 1 \end{bmatrix},$ 0 0
0 0 0
0 0 0 1

$\text{ТТК}_4(8\text{Sin}^3(x)) = 0 \ 0 \ 0 \ 1$

следует, что $\underline{\text{ТТК}}_4(2\text{Sin}(3x)) = \begin{bmatrix} 0 \\ 0 \ 1 \\ 0 \ 0 \ 0 \\ 0 \ -1 \ 0 \ 0 \end{bmatrix}.$ Наконец, из

$2\text{Cos}(3x) = -6\text{Cos}(x) + 8\text{Cos}^3(x),$

$\text{ТТК}_4(6\text{Cos}(x)) = \begin{bmatrix} 3 \\ 0 \ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \ 0 \\ -1 \ 0 \ 1 \\ 0 \ 0 \ 0 \ 0 \end{bmatrix},$ 1
0 0
0 0 0
0 0 0 0

$\text{ТТК}_4(8\text{Cos}^3(x)) = 0 \ 0 \ 0 \ 0$

следует, что $\underline{\text{ТТК}}_4(2\text{Cos}(3x)) = \begin{bmatrix} 0 \\ 0 \ 0 \\ 1 \ 0 \ -1 \\ 0 \ 0 \ 0 \ 0 \end{bmatrix}.$

3. Математические операции с четверичными тригонометрическими треугольными кодами

В общем виде эти операции описаны в разделе 2.3. Здесь будут указаны лишь особенности четверичных кодов. Рассмотрим примеры.

Пример 3.1. Сдвиг оси ординат.

Проверим преобразованием ТТК справедливость равенства

$$F''(x) = F'(x + \pi/2)$$

где функции $F''(x)$, $F'(x)$ определены в примере 2.2.2. Из примера 2.9 находим $\text{ТТК}_4(F'(x))$. Применяя к нему формулу

$$\psi'_{mk}(x) = (-1)^m \psi_{k-m,k}(x)$$

получаем $\text{ТТК}_4(F''(x))$, приведенный в примере 3.3.

Пример 3.2. Умножение.

Найдем произведение $\Phi_1(x) = F(x) \cdot F'''(x)$, где коды функций $F(x)$, $F'''(x)$ определены в примере 2.9 – см. коды ТК_4^0 и ТК_4^3 соответственно. При этом

								0
							0	1
							-1	-2
						1	0	-1
					-1	1	0	0
				-1	-1	1	-2	1
			-1	-1	-1	1	-1	-1
			-1	-1	-1	1	-1	-1
$\text{ТК}_4(\Phi_1(x)) =$		0	-1	-1	1	1	0	1
	0	-2	0	-1	0	1	0	0

Пример 3.3. Дифференцирование.

Проверим дифференцированием ТТК справедливость равенства

$$F'''(x) = \frac{dF(x)}{dx}$$

где функции $F'''(x)$, $F(x)$ определены в примере 2.2.2. Из примера 2.9 находим $\text{ТТК}_4(F(x))$. Производя вычисление по формуле (2.3.1), получаем

									0
								0	2
							0	0	0
						0	0	0	-2
					0	0	0	0	0
				0	2	0	-4	0	2
			0	0	0	0	0	0	0
ТТК($F'''(x)$) =		0	6	0	2	0	8	0	-2
	0	0	0	0	0	0	0	0	0

Преобразуя этот код в четверичный, находим $\text{ТТК}_4(F'''(x))$, приведенный в примере 2.9.

4. Укорочение четверичных тригонометрических треугольных кодов

В общем виде укорочение четверичных тригонометрических треугольных кодов описано в разделе 2.6. Четверичные тригонометрические треугольные коды имеют особенность, состоящую в том, что функции вычисляются по упрощенным формулам и коды этих функций являются целыми. Действительно, из (2.6.6) следует:

$$\delta^i(n) = \sum_{v=0}^{n-1} \left[(-1)^{n+v+1} S^i(v,n) \cdot \psi_{0v} \right] \tag{4.1}$$

Но числа $S^i(v,n)$ являются целыми – см. теорему 2.1.1. Следовательно, коды $\text{ТК}_4(\delta^i(n))$ также являются целыми – см. теорему 1.6.1. Таким образом, функции $\delta^i(n)$ имеют целые коды (а

не смешанные, как это предполагалось для общего случая). Кроме того, коды $TK_4(\delta^i(n))$ имеют ровно $(n+1)$ значащих столбцов.

Смешанные коды $B_n^i = SK_4\left(\frac{\delta_n^i}{4}\right)$ функций $\frac{\delta_n^i}{4}$ могут быть получены (как и в общем случае) по формуле (2.6.4) при известных $\eta^i \gamma_n^i$. Но в данном случае эти смешанные коды имеют только одну строку в прямоугольной части кода (из-за того, что коды $TK(\delta^i(n))$ являются целыми). Все это существенно упрощающими выполнение приближенных операций с TTK_4 . Для иллюстрации в табл. 4.1 приведены коды функций, γ_n^0 , δ_n^0 , $\frac{\delta_n^0}{4}$ при $n < 5$. Для вычисления формула (2.6.4) была преобразована к виду

$$\delta_n^i = (-1)^{n+1} \gamma_n^i + \psi_{0n}.$$

Таблица 4.1.

n	$TK_4(\gamma_n^0)$	$TK_4(\delta_n^0)$	$B_n^0 = SK_4\left(\frac{\delta_n^0}{4}\right)$
0	1	0	0
1	1 -2 0	1 -2 1	1 ----- -2
2	1 -1 -2 1 0	0 1 -2 0 1	0 0 1 ----- -2 0
3	1 -2 1 -2 0 0 0	1 -2 1 -2 0 0 1	0 0 1 1 -2 -2 1 ----- -2 -1 -2 0
4	0 -1 1 1 0 -1 0 -2 1 0 0 0	0 1 -1 0 0 1 0 -2 0 0 0 1	0 0 0 1 -1 0 0 0 1 ----- -2 0 0 0

Как указывалось, укорочение сосоит в делении на параметр и округлении полученного смешанного кода, имеющего одну строку в прямоугольной части. Рассмотрим алгоритм округления такого смешанного кода, использующий указанные свойства кодов

$$B_n^i = SK_4 \left(\delta_n^i / 4 \right) \text{ и таблицы этих кодов. Этот алгоритм основан на}$$

замене разряда π_{ov} этого смешанного кода SK_4^{old} на код $\pi_{ov} B_v^i$. Новое значение округляемого смешанного кода вычисляется как сумма смешанных кодов:

$$SK_4^{new} = SK_4^{old} + \pi_{ov} (B_v^i - \psi_{-1,v}). \quad (4.2)$$

При этом могут встретиться случаи сложения с кодом B_v^i и вычитания кодов B_v^i или $2 B_v^i$, так как $\pi_{0v} \in \{-2, -1, 0, 1\}$.

Алгоритм 4.1. Округление кода $SK_4(f^i(x))$.

1. Присвоение значения $SK_4^{old} = SK_4(f^i(x))$.
2. Последовательный поиск старшего значащего разряда π_{ov} в младшей (-1)-строке округляемого кода SK_4^{old} . Если значащий разряд при $v > 0$ есть, то выполняется переход к следующему пункту. В противном случае выполняется переход к пункту 5.
3. Стирание разряда π_{ov} и алгебраическое сложение результата с кодом $\pi_{ov} B_v^i$, то есть реализация формулы (4.2).
4. Присвоение значения $SK_4^{old} = SK_4^{new}$ и переход к пункту 2.
5. Прекращение счета, поскольку SK_4^{old} не имеет (-1)-строки, то есть является треугольным кодом $SK_4(f_o^i(x))$, где $f_o^i(x)$ - округленная функция $f^i(x)$. При этом погрешность округления $\Delta = \Delta_n^i | \pi_{0v} |_{\max} / R$. Учитывая что $\Delta_n^i = 2$ и $| \pi_{0v} |_{\max} = 2$, находим $\Delta_{\max} = 1$

Пример 4.1. Укорочение.

Пусть $R=4$, $\alpha \in \{-2, -1, 0, 1\}$ и известен код, который необходимо укоротить на один столбец:

					0
				-1	-1
			1	1	-1
$TK_4(F^0(x)) =$		-2	0	-2	0
	1	0	-1	-2	0

После деления кода данной функции на параметр 4 получаем смешанный код функции $f^0(x) = F^0(x)/4$:

					0
				-1	0
		1	-2	0	
$SK_4(f^0(x)) =$	-2	0	0	0	
	1	-2	0	-2	

После округления этого кода, выполненного в примере 4.2, получаем

			-1
	1		-1
$TK_4(f_0^0(x)) =$	1	0	-2

Пример 4.2. Округление.

Округлим код $SK_4(f^0(x))$, полученный в примере 4.1, по алгоритму 4.1.

1. Присваиваем $SK_4^{old} = SK_4(f^0(x))$.
2. Находим $\pi_{03} = -2$.

3. Выбираем из табл. 4.1 код B_3^0 и вычисляем код $\pi_{03}B_3^0 = -2B_3^0$:

		0	0	1
$B_3^0 =$	1	-2	-2	1
	-2	-1	-2	0

		0	1
$-2B_3^0 =$	-1	1	-2
	0	-2	0

4. Складываем полученный код с кодом SK_4^{old} , в котором стерт разряд π_{03} , т.е. вычисляем код SK_4^{new} :

$$\begin{array}{|c|c|c|} \hline & & -1 \\ \hline & 1 & -2 \\ \hline -2 & 0 & 0 \\ \hline 1 & -2 & 0 \\ \hline \end{array} + \begin{array}{|c|c|c|} \hline & & 0 \\ \hline & 0 & 1 \\ \hline -1 & 1 & -2 \\ \hline 0 & -2 & 0 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline & & -1 \\ \hline & 0 & -1 \\ \hline 1 & 0 & -2 \\ \hline 1 & 0 & 0 \\ \hline \end{array}$$

Это эквивалентно тому, что к функции $f^0(x)$, представленной кодом SK_4^{old} , прибавлена функция $(\delta_3^n - \psi_{0n})/4 = (2\gamma_3^0)/4$.

5. Присваиваем $SK_4^{old} = SK_4^{new}$.

6. Обнаруживая, что $\pi_{ov}=0$ при $v>0$, прекращаем счет. Это эквивалентно тому, что к функции, представленной кодом SK_4^{new} , прибавлена (-1). В результате получаем

$$TK_4(f_0^0(x)) = \begin{array}{|c|c|c|} \hline & & -1 \\ \hline & 1 & -1 \\ \hline 1 & 0 & -2 \\ \hline \end{array}$$

Таким образом, $4[f^0(x) - f_0^0(x)] = [2\gamma_3^0 - 1]$.

Пример 4.3. Оценка погрешности округления.

Для оценки погрешности округления в примере 4.2 заметим, что по существу вычислений $4[f^0(x) - f_0^0(x)] = [2\gamma_3^0 - 1]$. Из примера 2.1.2 следует, что $[2\gamma_3^0 - 1] = [-1 - 4\cos(6x)]$. Таким образом, абсолютная погрешность вычислений является функцией

$$[f^0(x) - f_0^0(x)] = -0.25 - \cos(6x),$$

то есть абсолютная погрешность коэффициентов тригонометрического ряда $\Delta = 1$. Для определения относительной погрешности округления кода в примере 4.1 декодируем результат округления $TK_4(f_0^0(x))$, как показано в примере 2.5а, и получаем функцию

$$f_0^0(x) = (-11 + 3\psi_{01} - 2\psi_{02})$$

Далее, преобразуем эту функцию, как показано в примере 2.1.3а, и получаем

$$f_0^0(x) = -17 - 10\cos(2x) - 4\cos(4x).$$

Далее находим:

$$f^0(x) = -17.25 - 10\cos(2x) - 4\cos(4x) - \cos(6x).$$

Таким образом, относительная погрешность округления

$$\Delta / \left| D_{\omega}^0 \right|_{\max} = \Delta / \left| D_2^0 \right|_{\max} = 1/17 \approx 6\%.$$

5. Погрешность кодирования четверичных тригонометрических треугольных кодов

Выше (в разделе 2.5) было показано, что для оценки максимальной относительной погрешности приближенных вычислений необходимо построить функцию M_{ω} максимально возможного модуля коэффициента D_{ω} . В качестве иллюстрации рассмотрим построение кривой для кодов ТТК($F(x)$) – см. свойство 2.2.3. Напомним, что числа $p(m, k, \omega)$ и $q(m, k, \omega)$ определены в теоремах 2.5.1 и 2.5.2 соответственно. Обозначим:

$\Pi_1(k, \omega)$ - сумма всех положительных чисел $p(m, k, \omega)$ при постоянных k и ω ;

$\Pi_2(k, \omega)$ - сумма всех отрицательных чисел $p(m, k, \omega)$ при постоянных k и ω ;

$Q_1(k, \omega)$ - сумма всех положительных чисел $q(m, k, \omega)$ при постоянных k и ω ;

$Q_2(k, \omega)$ - сумма всех отрицательных чисел $q(m, k, \omega)$ при постоянных k и ω ;

Числа $\Pi_1(k, \omega)$ и $\Pi_2(k, \omega)$ приведены в табл. 2.5.1 раздела 2.5. Из формулы (2.5.8) при $R=4$ находим:

$$Q(k, \omega) = \begin{cases} \Pi(k, \omega) & \text{if } \omega = 0, \\ 2\Pi(k, \omega) & \text{if } \omega > 0. \end{cases} \quad (5.1)$$

Далее при $\alpha_{mk} \in \{-2, -1, 0, 1\}$ имеем:

$$\left[\sum_{m=0}^k \left(\alpha_{mk} \cdot q(m, k, \omega) \right) \right]_{\max} = Q_1(k, \omega) + 2Q_2(k, \omega),$$

$$\left[\sum_{m=0}^k \left(\alpha_{mk} \cdot q(m, k, \omega) \right) \right]_{\min} = 2Q_1(k, \omega) - Q_2(k, \omega).$$

Отсюда, учитывая (2.5.10), получаем:

$$|D_\omega > 0|_{\max} = \sum_{k=\omega}^n [Q_1(k, \omega) + 2Q_2(k, \omega)], \quad (5.2)$$

$$|D_\omega < 0|_{\max} = \sum_{k=\omega}^n [2Q_1(k, \omega) - Q_2(k, \omega)]. \quad (5.3)$$

Обозначим:

$$\mu_1(\omega, n) = \sum_{k=\omega}^n [\Pi_1(k, \omega)], \quad (5.4)$$

$$\mu_2(\omega, n) = \sum_{k=\omega}^n [\Pi_2(k, \omega)]. \quad (5.5)$$

Числа $\mu_1(\omega, n)$, $\mu_2(\omega, n)$ при $\omega \leq 5$, $n \leq 5$ приведены в табл. 5.1 и 5.2.

Таблица 5.1. Числа $\mu_1(\omega, n)$

$n \setminus \omega$	0	1	2	3	4	5
0	1					
1	5	1				
2	19	5	2			
3	67	21	14	2		
4	233	81	70	14	3	
5	817	307	310	75	27	3

Таблица 5.2. Числа $\mu_2(\omega, n)$

$n \setminus \omega$	0	1	2	3	4	5
0	0					
1	0	-1				
2	0	-5	-1			
3	0	-21	-5	-2		
4	0	-81	-17	-14	-2	
5	0	-307	-49	-75	-14	-3

Имея в виду формулу (5.3), а также то, что $p(m, k, \omega=0) > 0$ и, следовательно, $\Pi_2(k, \omega=0) = 0$, находим:

$$|D_{\omega} > 0|_{\max} = \begin{cases} \mu_1(\omega, n) & \text{if } \omega = 0, \\ (2\mu_1(\omega, n) - 4\mu_2(\omega, n)) & \text{if } \omega > 0, \end{cases} \quad (5.6)$$

$$|D_{\omega} < 0|_{\max} = \begin{cases} 2\mu_1(\omega, n) & \text{if } \omega = 0, \\ (4\mu_1(\omega, n) - 2\mu_2(\omega, n)) & \text{if } \omega > 0, \end{cases} \quad (5.7)$$

Числа (5.6) и (5.7) при $\omega \leq 5$, $n \leq 5$ приведены в табл. 5.3 и 5.4. В этих же таблицах приведены аналогичные величины, подсчитанные для n -разрядного четверичного кода числа по формулам:

$$|D > 0|_{\max} = \sum_{k=0}^n 4^k = \frac{1}{3}(4^{n+1} - 1),$$

$$|D < 0|_{\max} = 2 \sum_{k=0}^n 4^k = \frac{2}{3}(4^{n+1} - 1).$$

Таблица 5.3. Числа $|D_{\omega} > 0|_{\max}$.

$n \setminus \omega$	0	1	2	3	4	5	$ D > 0 _{\max}$
0	1						1
1	5	6					5
2	19	30	8				21
3	67	126	48	12			85
4	233	1842	208	84	14		341
5	817	6930	816	450	110	18	1365

Таблица 5.4. Числа $|D_{\omega} < 0|_{\max}$.

$n \setminus \omega$	0	1	2	3	4	5	$ D < 0 _{\max}$
0	2						2
1	10	6					10
2	38	30	10				42
3	134	126	66	12			170
4	466	486	314	84	16		682
5	1634	1842	1338	450	136	18	2730

Из табл. 5.3 и 5.4 видно, что $M_\omega = |D_\omega < 0|_{\max}$. Зависимости $\lg(M_\omega^0)$ от ω при различных значениях n приведены на рис. 5.1.

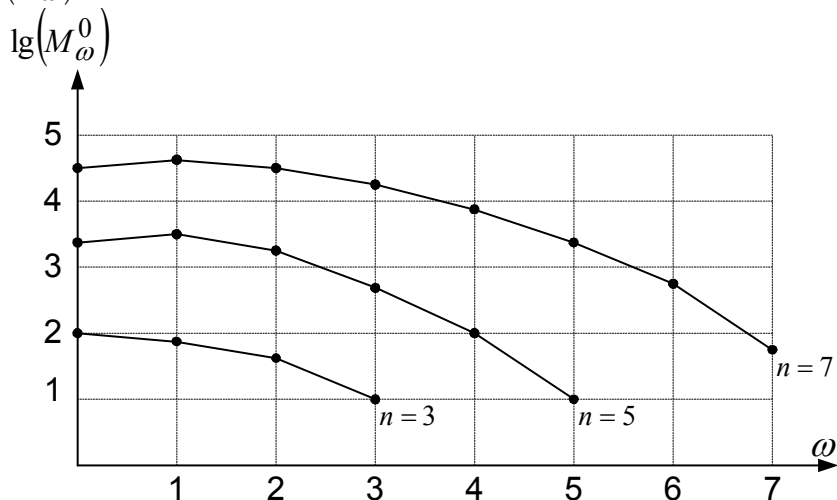


Рис. 5.1. Зависимость M_ω^0 от ω .

На рис. 5.2 изображены зависимости $\lg(M_\omega^0)$ от n при различных значениях ω . Там же для сравнения приведена зависимость максимального модуля действительного числа в системе кодирования $\langle \rho = 4, A_4 = \{-2, -1, 0, 1\} \rangle$ (в этом случае индекс ω теряет смысл).

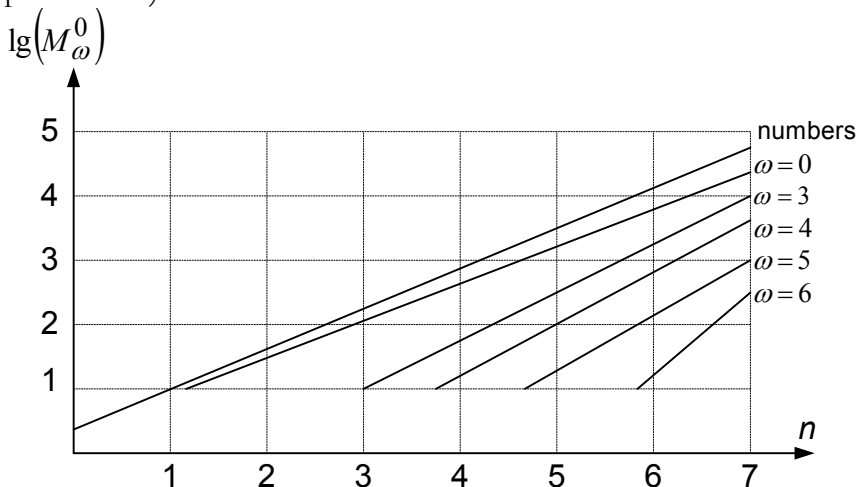


Рис. 5.2. Зависимость M_ω^0 от n .

Глава 5. Арифметическое устройство для операций с функциями

В данной главе подробно рассматривается арифметическое устройство для операций четверичными тригонометрическими треугольными кодами функций. Операционные устройства этого АУ должны оперировать с четверичными разрядами, принимающими значения из множества $D_R = \{-2, -1, 0, 1\}$. Для представления таких разрядов и описания операций с ними мы воспользуемся методом кодирования вещественных чисел по отрицательному основанию «-2» - см. часть 1, где введены понятия **М-кодов** и **Р-кодов**.

М-код некоторого числа a будем обозначать как $K(a)$. Например, $K(0)=00$, $K(1)=01$, $K(-1)=11$, $K(-2)=10$. Схемы алгебраического суммирования М-кодов строятся на основе т.н. обратного сумматора, реализующего формулу $c \equiv (-a-b)$. Объем обратного сумматора М-кодов равен объему обычного сумматора Р-кодов.

5.1. Одноразрядные схемы

5.1.1. Одноразрядный сумматор.

Из формул (1.3.1), (1.3.2) и табл. 4.1.2 следует, что числа α_{mk} , β_{mk} , π'_{mk} , π''_{mk} , π_{mk} , σ_{mk} имеют двухразрядные М-коды, а число S_{mk} имеет четырехразрядный М-код. Кроме того, число S_{mk} может быть представлено в виде:

$$S_{mk} = \sigma_{mk} + (-2)^2 \pi_{mk} = (\alpha_{mk} + \pi'_{mk}) + (\beta_{mk} + \pi''_{mk}).$$

Схема одноразрядного сумматора четверичных кодов, в которой учтены эти особенности, приведена на рис. 5.1.1, где

\sum_n - n -разрядный сумматор М-кодов,

\xrightarrow{n} - n -канал передачи разрядов.

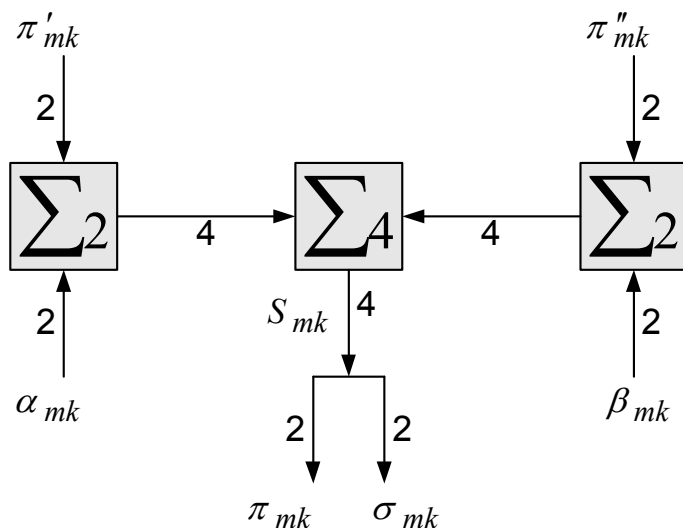


Рис. 5.1.1. Одноразрядный сумматор четверичных кодов

Заметим, что объем этого сумматора равен объему 8-ми одноразрядных сумматоров чисел. Поскольку этот сумматор складывает двухразрядные коды, можно сказать, что относительный (отнесенный к разрядности) объем сумматора ТК в 4 раза больше относительного объема сумматора чисел.

Задержка в этом сумматоре в 6 раз больше задержки в одноразрядном сумматоре чисел.

5.1.2. Одноразрядный вычитатель.

По аналогии с одноразрядным сумматором строится одноразрядный вычитатель четверичных кодов. В этом случае число S_{mk} может быть представлено в виде:

$$S_{mk} = \sigma_{mk} + (-2)^2 \pi_{mk} = (\alpha_{mk} + \pi'_{mk}) + (-\beta_{mk} + \pi''_{mk}).$$

Схема одноразрядного вычитателя четверичных кодов приведена на рис. 5.1.2 и отличается от схемы сумматора только тем, что в ней один из сумматоров заменен на вычитатель, обозначенный так:

$\overline{\Sigma}_n$ - n -разрядный вычитатель М-кодов.

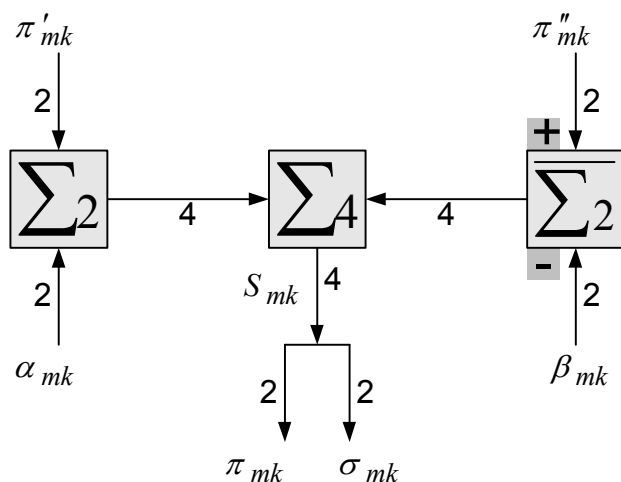


Рис. 5.1.2. Одноразрядный вычитатель четверичных кодов

5.1.3. Одноразрядный инвертор.

По аналогии с вычитателем строится одноразрядный инвертор четверичных кодов. В этом случае числа α_{mk} , σ_{mk} имеют двухразрядные М-коды, числа π'_{mk} , π''_{mk} , π_{mk} имеют двухразрядные М-коды, число S_{mk} имеет трехразрядный М-код и может быть представлено в виде:

$$S_{mk} = \sigma_{mk} + (-2)^2 \pi_{mk} = (\pi'_{mk} + \pi''_{mk}) - (\alpha_{mk}).$$

Схема одноразрядного инвертора четверичных кодов приведена – см. рис. 5.1.3.

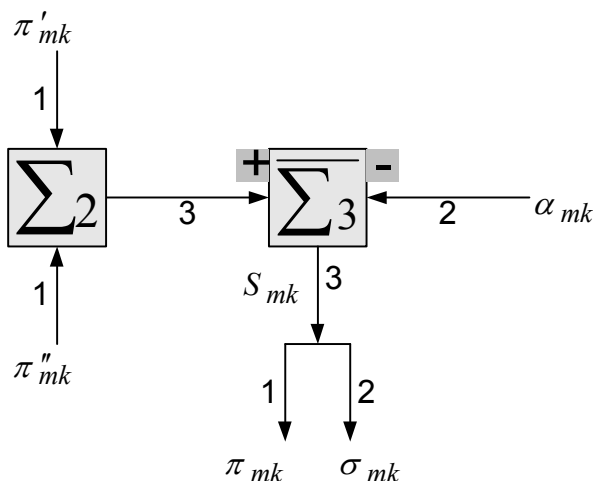


Рис. 5.1.3. Одноразрядный инвертор четверичных кодов

5.1.4. Одноразрядный учетверитель.

По аналогии с сумматором строится одноразрядный инвертор четверичных кодов. В этом случае числа α_{mk} , σ_{mk} имеют двухразрядные М-коды, числа π'_{mk} , π''_{mk} , π_{mk} имеют четырехразрядные М-коды, число S_{mk} имеет 6-тиразрядный М-код и может быть представлено в виде:

$$S_{mk} = \sigma_{mk} + (-2)^2 \pi_{mk} = (\pi'_{mk} + \pi''_{mk}) + (-2)^2 (\alpha_{mk}).$$

Схема одноразрядного учетверителя четверичных кодов приведена – см. рис. 5.1.4.

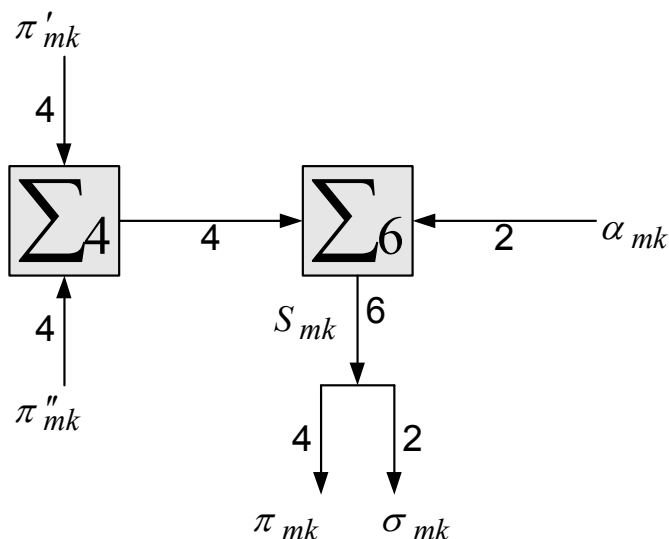


Рис. 5.1.4. Одноразрядный учетверитель четверичных кодов

5.1.5. Одноразрядный делитель на 4.

Из формул (1.4.1), (1.4.2) и табл. 4.1.3 следует, что числа α_{mk} , π'_{mk} , π''_{mk} , π_{mk} имеют двухразрядные М-коды, число σ_{mk} имеет четырехразрядный М-код, число $4S_{mk}$ имеет 6-тиразрядный М-код и может быть представлено в виде:

$$4S_{mk} = ((-2)^2 \sigma_{mk}) + (\pi_{mk}) = (\alpha_{mk} - \pi'_{mk}) + ((-2)^2 \pi''_{mk}),$$

Схема одноразрядного сумматора четверичных кодов, в которой учтены эти особенности, приведена на рис. 5.1.5.

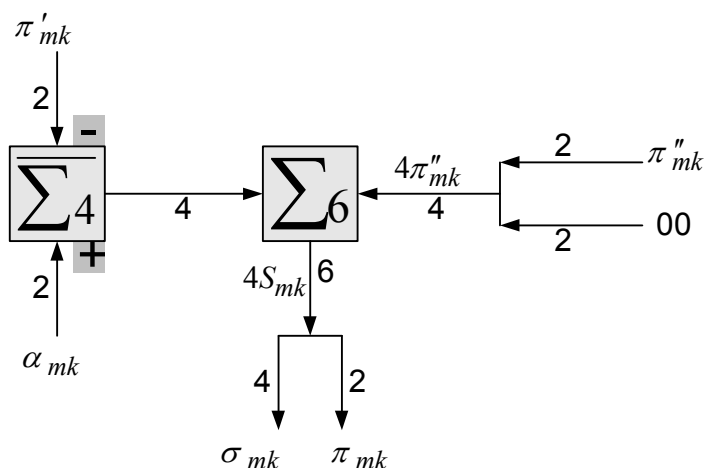


Рис. 5.1.5. Одноразрядный делитель на 4 четверичных кодов

5.2. Многоразрядные схемы

Многоразрядные схемы для операций с треугольными кодами строятся на основе описанных выше одноразрядных схем. Можно предложить несколько типов многоразрядных схем, различающихся количеством параллельно работающих одноразрядных схем и порядком чередования групп разрядов, обрабатываемых параллельно. Кроме того, тип схемы зависит от направления распространения переносов и по этому критерию схемы разбиваются на две группы:

- алгебраические сумматоры, в которых перенос распространяется вверх и влево,
- делители на параметр, в которых перенос распространяется вниз и вправо.

В дальнейшем из первой группы мы рассмотрим только сумматор, поскольку остальные схемы этой группы полностью ему аналогичны.

5.2.1. Столбцовый сумматор.

Этот сумматор изображен на рис. 5.2.1. Он складывает одноименные столбцы α и β слагаемых кодов. При этом одноименные разряды столбцов складываются параллельно. При сложении образуется столбец результата σ и столбец выходного переноса $\bar{\pi}$ в следующий столбец. На входе сумматора

присутствует столбец входного переноса $\underline{\pi}$ из предыдущего столбца. Очевидно, разряд входного переноса используется как входной перенос π' для одного из одноразрядных сумматоров и как входной перенос π'' для другого из одноразрядных сумматоров.

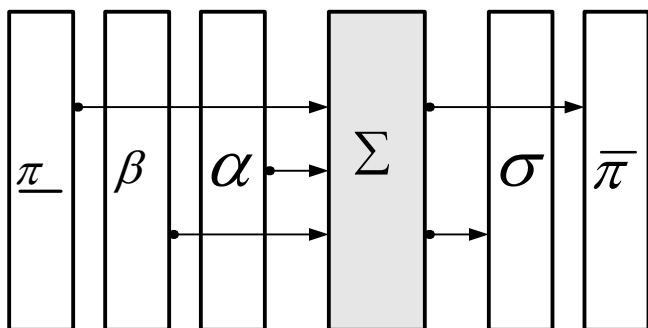


Рис. 5.2.1. Столбцовый сумматор.

Сложение кодов в целом состоит в последовательном сложении столбцов слева направо. Обозначим:

τ длительность сложения на одноразрядной схеме,

t' длительность передачи из регистра в регистр,

t'' длительность записи в регистр,

N количество столбцов и строк в треугольном коде.

Тогда полная длительность сложения на столбцовом сумматоре

$$t_1 = N(\tau + t' + t'')$$

5.2.2. Строчный сумматор.

Этот сумматор изображен на рис. 5.2.2. Он складывает одноименные строки α и β слагаемых кодов. При этом выполняется распространение переносов π' через одноразрядные схемы. При сложении образуется строка результата σ и строка выходного переноса $\bar{\pi}$ в следующую строку. На входе сумматора присутствует строка входного переноса $\underline{\pi}$ из предыдущей строки. Очевидно, разряд входного переноса используется как входной перенос π'' для одного из одноразрядных сумматоров.

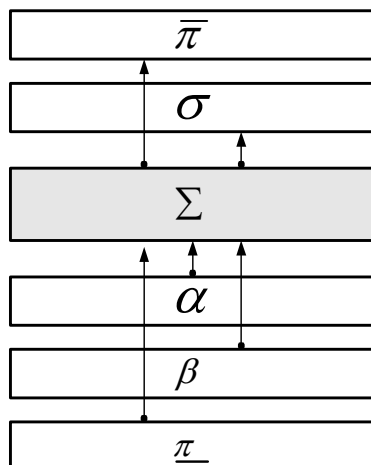


Рис. 5.2.2. Строчный сумматор.

Сложение кодов в целом состоит в последовательном сложении строк снизу вверх. Полная длительность сложения на строчном сумматоре

$$t_2 = N(N\tau + t' + t'')$$

5.2.3. Столбцовый делитель.

Этот делитель изображен на рис. 5.2.3. Он делит столбец α исходного кодов. При этом выполняется распространение переносов π' через одноразрядные схемы. При делении образуется столбец результата σ и столбец выходного переноса $\bar{\pi}$ в следующий столбец. На входе делителя присутствует столбец входного переноса $\underline{\pi}$ из предыдущего столбца. Очевидно, разряд входного переноса используется как входной перенос π'' для одной из одноразрядных схем.

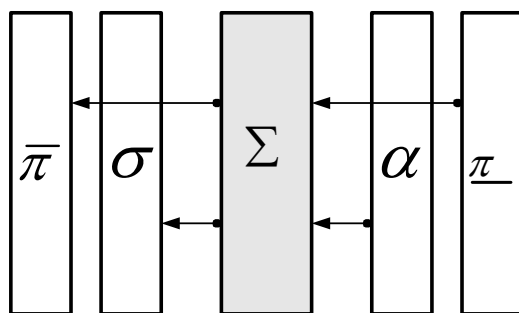


Рис. 5.2.3. Столбцовый делитель.

Деление кода в целом состоит в последовательном делении столбцов справа налево. Полная длительность деления на столбцовом делителе

$$t_3 = N(N\tau + t' + t'')$$

5.2.4. Строчный делитель.

Этот делитель изображен на рис. 5.2.4. Он делит строку α исходного кодов. При делении образуется строка результата σ и строка выходного переноса $\bar{\pi}$ в следующую строку. На входе сумматора присутствует строка входного переноса $\underline{\pi}$ из предыдущей строки. Очевидно, разряд входного переноса используется как входной перенос π' для одной из одноразрядных схем и как входной перенос π'' для другой из одноразрядных схем.

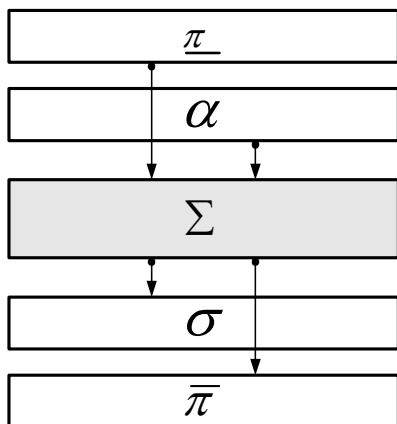


Рис. 5.2.4. Строчный делитель.

Деление кода в целом состоит в последовательном делении строк сверху вниз. Полная длительность деления на столбцовом делителе

$$t_4 = N(\tau + t' + t'')$$

5.2.5. Параллельный сумматор.

Этот сумматор изображен на рис. 5.2.5, где показаны связи одноразрядных сумматоров цепями переносов. Здесь указаны также цепи переносов из прямоугольной составляющей в смешанном коде. Полная длительность сложения на этом сумматоре

$$t_5 = N\tau + t''$$

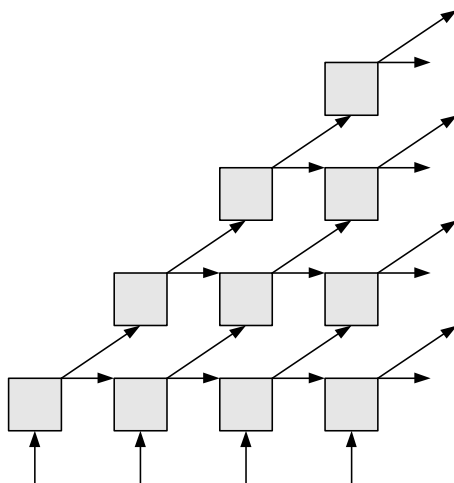


Рис. 5.2.5. Параллельный сумматор.

5.2.6. Параллельный делитель.

Этот делитель изображен на рис. 5.2.6, где показаны связи одноразрядных схем цепями переносов. Здесь указаны также цепи переносов в прямоугольную составляющую смешанного кода. Полная длительность деления на этом делителе

$$t_6 = N\tau + t''$$

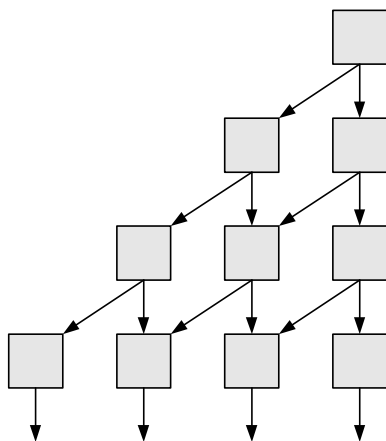


Рис. 5.2.6. Параллельный делитель.

5.3. Вариант арифметического устройства

5.3.1. Структура арифметического устройства.

Прежде всего, рассмотрим место арифметического устройства для операций с функциями в центральном процессоре – см. рис. 5.3.1, где

CPU - центральный процессор,

TAU - традиционное арифметическое устройство,

KDM - кодер-декодер М-кодов,

FAU - арифметическое устройство для операций с функциями

Как будет ясно из дальнейшего, все эти устройства должны участвовать в операциях кодирования и декодирования функций.

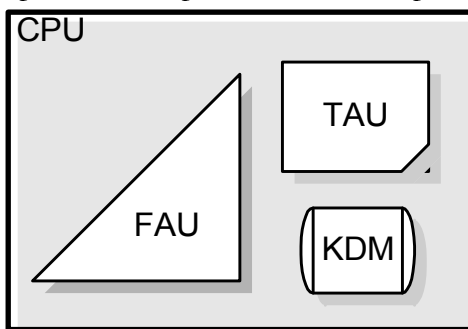


Рис. 5.3.1. Центральный процессор

Ниже рассматривается один из возможных вариантов построения FAU - арифметического устройства, оперирующего с четверичными тригонометрическими кодами функций. Блок-схема этого устройства приведена на рис. 5.3.2, где

RegTTK – регистр кода TTK,

RegTK – регистр кода TK,

RegAK – регистр кода AK,

RegSK – регистр кода SK, который объединяет регистры RegTK и RegAK,

RegPK – регистр кода PK,

ROM – таблицы кодов,

ADDSK – сумматор кодов SK,

DIVIS – делитель на 4 кода TK,

QUADR – умножитель на 4 кода TK,

ANL – анализатор разрядов множителя,

SHIFT – сдвигатель.

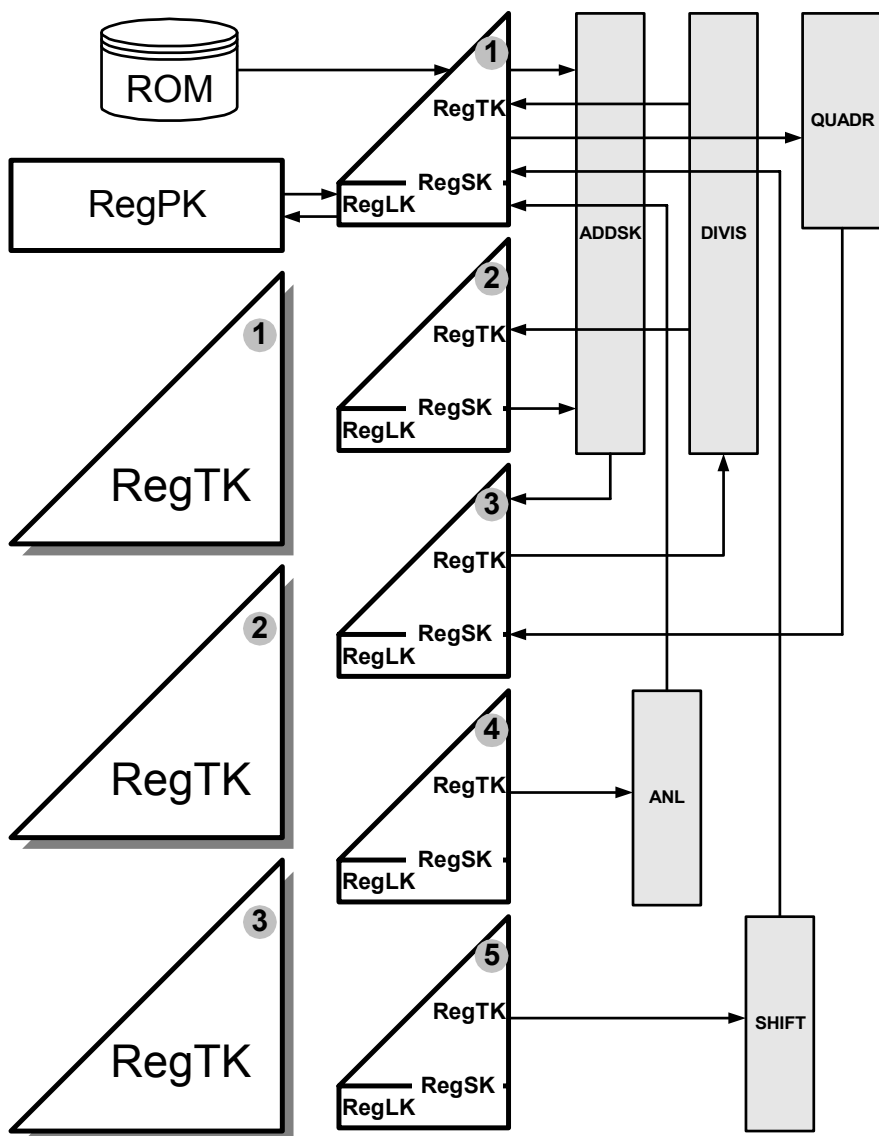


Рис. 5.3.2. FAU

Рассмотрим последовательность выполнения различных операций на этом FAU.

Как было показано, арифметические операции с четверичными кодами эквивалентны операциям с М-кодами, а в состав процессора, оперирующего с функциями, должен быть включен KDM - кодер-декодер М-кодов. Эти операции и KDM рассмотрены в части 1 этой книги.

5.3.2. Операции с треугольными кодами.

1. Алгебраическое сложение смешанных кодов.

Оно выполняется в устройстве ADDSK, причем исходные коды находятся в регистрах RegSK1 и RegSK2, а результат формируется в регистре RegSK3.

2. Учетверение смешанного кода.

Оно выполняется в устройстве QUADR, причем исходный код находится в регистре RegSK1, а результат формируется в регистре RegTK3.

3. Деление треугольного кода на 4.

Оно выполняется в устройстве DIVIS, причем исходный код находится в регистре RegTK3, а результат – смешанный код формируется в регистрах RegSK1 и RegSK2 (напомним, что результат деления формируется в двух регистрах). После деления выполняется сложение кодов в регистрах RegSK1 и RegSK2 на устройстве ADDSK. Результат формируется в регистре RegSK3, т.е. там же, где находился исходный код.

4. Округление смешанного кода.

Оно выполняется согласно алгоритму 4.4.1. В устройстве RAM-ТК хранятся коды aB_v^i , $a \in \{1, -1, -2\}$. Алгоритм реализуется следующим образом:

1. Округляемый код $SK_4 = SK_4(f^i(x))$ передается в регистр RegSK3.
2. Округляемый код SK_4 , находящийся в регистре RegSK3, передается в регистр RegSK2.

3. В младшей (-1)-строке регистра RegSK2 отыскивается старший значащий разряд π_{ov} округляемого кода. Если такой разряд отсутствует, то округление заканчивается с результатом в регистре RegTK3.
4. В регистр RegSK1 передается код $\pi_{ov}B_v^i$, а в регистре RegSK2 стирается разряд π_{ov} , т.е. в этом регистре образуется код $SK_4 \leftarrow SK_4 - \pi_{ov}\psi_{-1,v}$
5. На устройстве ADDSK выполняется сложение: $RegSK3 = RegSK1 + RegSK2$ или $SK_4 \leftarrow SK_4 + \pi_{ov}B_v^i$. Два последних пункта реализуют формулу (4.4.2).
6. Выполняется переход к п. 2.

6. Укорочение треугольного кода.

Оно выполняется следующим образом:

1. Укорачиваемый код передается в регистр RegTK3.
2. Выполняется деление на 4 кода, находящегося в регистре RegTK3. Смешанный код результата оказывается в регистре RegSK3.
3. Если в младшей (-1)-строке регистра RegSK3 все разряды равны нулю, то укорочение заканчивается с результатом в регистре RegTK3. В противном случае выполняется округление кода, находящегося в регистре RegSK3. Результат образуется в регистре RegTK3.
4. Выполняется переход к п. 2.

7. Преобразование прямоугольного кода в треугольный код.

Оно выполняется следующим образом:

1. Прямоугольный код $PK(F(x))$ передается в регистр RegPK, а регистр RegSK1 обнуляется.
2. В регистр RegAK1 (являющийся частью регистра RegSK1) записывается очередная i -строка прямоугольного кода — линейный код $LK(f_i(x))$. Если перебор строк закончился, то преобразование заканчивается с результатом в регистре RegTK1.
3. Выполняется учетверение кода, находящегося в регистре RegSK1. Результат формируется в регистре RegTK3.

4. Выполняется перепись из регистра RegTK3 в регистр RegTK1.
5. Выполняется переход к п. 2.

Заметим, что в частности прямоугольный код может иметь только один столбец, т.е. представлять *число*. В этом случае данная операция соответствует кодированию числа в треугольный код.

8. Преобразование треугольного кода в прямоугольный код.

Оно выполняется следующим образом:

1. Преобразуемый треугольный код $TK(F(x))$ передается в регистр RegTK3.
2. Выполняется деление кода, находящегося в регистре RegTK3. Результат формируется в регистре RegSK1.
3. В очередную i -строку прямоугольного кода, находящегося в регистре RegPK, записывается линейный код $LK(f_i(x))$, образовавшийся в регистре RegLK1 (являющимся частью регистра RegSK1) Если перебор все строки записались, то преобразование заканчивается с результатом в регистре RegPK.
4. Выполняется перепись из регистра RegTK1 в регистр RegTK3.
5. Выполняется переход к п. 2.

9. Умножение треугольных кодов.

Оно состоит (как указывалось) из чередующихся операций «сдвиг-сложение». При этом:

1. Множитель находится в регистре RegTK4. Очередной разряд a множителя анализируется в устройстве ANL. При этом определяется вид алгебраического сложения частичного произведения и множимого: множимое может складываться ($a=1$), вычитаться ($a=-1$) или удваиваться и вычитаться ($a=-2$).
2. Множимое находится в регистре RegTK5 и сдвигается на устройстве SHIFT в регистр RegTK1.
3. Частичное произведение перед очередным сложением находится в регистре RegTK2.
4. Новое частичное произведение формируется в регистре RegTK3, а затем передается в регистр RegTK2.

5.3.3. Операции с тригонометрическими треугольными кодами.

В данном АУ принят последовательный способ операций с ТТК – отдельные составляющие кодов ТТК, находящихся в регистрах RegТТК1 и RegТТК2, последовательно передаются в регистры RegТТК1 и RegТТК2. Результат, образующийся в регистре RegТТК3, переписывается в регистр RegТТК3. Таким образом, АУ оперирует с ТК, разрядность которых в 4 раза меньше разрядности ТТК. Таким образом, во всех операциях с ТТК используется по определению 2.2.1

- компоновка кода $\text{ТТК}_R(\Phi(x))$ из кодов $\text{ТК}_R(f^i(x))$,
- декомпоновка кода $\text{ТТК}_R(\Phi(x))$ в коды $\text{ТК}_R(f^i(x))$.

1. Алгебраическое сложение ТТК.

Алгебраическое сложение ТТК распадается на 4 алгебраических сложения ТК.

2. Умножение ТТК.

Каждая составляющая результирующего ТТК образуется в результате четырех умножений. Поэтому умножение ТТК содержит 16 умножений ТК и 12 сложений ТК.

3. Дифференцирование ТТК.

В данном варианте АУ реализуется первый способ дифференцирования – сложение заранее вычисленных кодов

$\text{ТК}^{mk} = \text{ТК}\left(\frac{d}{dx}\psi'_{mk}\right)$, которые хранятся в устройстве RAM-ТК.

При этом каждая составляющая ТТК дифференцируется независимо. Рассмотрим дифференцирование треугольного кода очередной составляющей ТТК:

1. Дифференцируемый код находится в регистре RegТК4. Очередной разряд a этого кода анализируется в устройстве ANL. При этом определяется вид алгебраического сложения частичного результата и табличного кода ТК^{mk} : этот код может складываться ($a=1$), вычитаться ($a=-1$) или удваиваться и вычитаться ($a=-2$).

2. Очередной табличный код TK^{mk} переписывается из устройства в регистр RegTK1.
3. Частичный результат перед очередным сложением находится в регистре RegTK2.
4. Новое частичный результат формируется в регистре RegTK3, а затем передается в регистр RegTK2.

4. Интегрирование ТТК.

В данном варианте АУ реализуется первый способ интегрирования с использованием сложения заранее вычисленных кодов $TK_R(J_{mk}(x))$, которые хранятся в устройстве RAM-ТК. При этом интегрирование выполняется аналогично дифференцированию.

5. Кодирование тригонометрического ряда.

В данном варианте АУ реализуется алгоритм 2.4.1 кодирования ТТК. При этом используются заранее вычисленные коды $TK\gamma_{\omega}^i$, которые хранятся в устройстве RAM-ТК. При этом каждая i -составляющая тригонометрического ряда кодируется независимо. Итак, компоненты тригонометрического ряда представлены в виде (2.1.10) и известны числа E_{ω}^i . Кодирование i -составляющей заключается в следующем:

1. Вычисление чисел E_{ω}^i при известных коэффициентах D_{ω}^i тригонометрического ряда. Это вычисление выполняется по формуле (2.1.19) на **TAU**.
2. Представление чисел E_{ω}^i в виде М-кодов или, что одно и то же, в виде четверичных кодов. Это вычисление выполняется на **KDM**.
3. Кодирование четверичных кодов чисел E_{ω}^i , то есть образование кодов $TK E_{\omega}^i$. Это кодирование описано выше.
4. Вычисление кода i -составляющей $TK(\eta^i f^i(x))$ по известным кодам $TK E_{\omega}^i$ и $TK\gamma_{\omega}^i$ по формуле (2.4.2), т.е. суммирование произведений $(TK E_{\omega}^i) \cdot (TK\gamma_{\omega}^i)$.

6. Декодирование ТТК.

Декодирование выполняется по В данном варианте АУ реализуется алгоритму 2.4.3. Декодирование каждой i -составляющей ТТК выполняется независимо и заключается в следующем.

1. Треугольный код i -составляющей преобразуется в прямоугольный код, представляющий собой множество четверичных чисел A_v^i . Эта операция описана выше.
2. Четверичные числа A_v^i преобразуются в обычные двоичные коды. Это вычисление выполняется на **KDM**.
3. По (2.1.17) вычисляются коэффициенты D_ω^i тригонометрического ряда функций $f^i(x)$ в зависимости от значений чисел A_v^i . При этом используются числа $L^i(v, n)$. Это вычисление выполняется на **TAU**.

7. Укорочение ТТК.

После каждой операции может возникнуть переполнение разрядной сетки – увеличение количества столбцов свыше установленного числа N . Укорочение ТТК заключается в следующем:

1. Определение числа M столбцов ТТК.
2. Укорочение каждой составляющей ТК на $(N-M)$ столбцов.
3. Умножение экспоненты ТТК на $R^{(N-M)}$. Это вычисление выполняется на **TAU**.

5.4. Сравнительный анализ

В этом разделе будет выполнен сравнительный анализ различных компьютеров, предназначенных для операций с тригонометрическими рядами вида (2.1.8). Такой ряд имеет коэффициенты D_ω^i и представляет некоторую функцию $\Phi(x)$. Мы рассмотрим три компьютера:

- компьютер Т – традиционный компьютер, оперирующий (в данном случае) коэффициентами D_ω^i тригонометрических рядов,

- компьютер Р – компьютер, содержащий несколько арифметических устройств, оперирующих с коэффициентами D_{ω}^i *параллельно*,
- компьютер Е – компьютер, содержащий **FAU** и оперирующий с кодами $\text{ТТК}_4(\Phi(x))$.

Для удобства дальнейшего изложения условимся называть ряд (2.1.8) тригонометрическим рядом Ω -ранга. Очевидно, для решения любой практической задачи с рядами, требующей, по крайней мере, умножения рядов, необходимо ограничиться максимально допустимым рангом ряда. При отсутствии этого ограничения многократное умножение неизбежно выведет объем информации о ряде за пределы границ, обусловленных конструкцией компьютера или программы. Таким образом, в любом случае должна быть предусмотрена операция укорочения ряда, заключающаяся в отбрасывании старших его членов с одновременной коррекцией младших членов ряда. В связи с этим в качестве пробной для сравнения компьютеров рассмотрим задачу, содержащую лишь операции сложения, умножения и укорочения тригонометрических рядов.

5.4.1. Взаимосвязь между разрядностью ТТК, рангом ряда и разрядностью коэффициентов ряда.

Если n – максимальный номер старшего столбца треугольного кода четырех составляющих ТТК некоторой функции, то ранг тригонометрического ряда этой функции

$$\Omega = 4n. \quad (7.1)$$

Из требования об ограничении ранга рядов следует, что амплитуды

$|D_{\omega}^i|$ гармоник должны, как правило, убывать с увеличением их частоты ω . Именно такой характер имеет зависимость максимально возможного модуля $M_{\omega} = |D_{\omega}^i|$ от частоты ω , построенная в разделе 4.5 – см. рис. 4.5.1. Поэтому в дальнейшем будем полагать, что частота ω' , при которой для кодируемого ряда $|D_{\omega'}^i| = \max_{\omega} |D_{\omega}^i|$, и частота ω'' , при которой $M_{\omega''} = \max_{\omega} M_{\omega}$, совпадают, т.е.

$$\max_{\omega} |D_{\omega}^i| = \max_{\omega} M_{\omega}.$$

Отсюда следует, что разрядность b_D максимального коэффициента тригонометрического ряда и разрядность b_M максимального из чисел M_ω совпадают.

Найдем величину b_M . Диапазон изменения коэффициента D_ω в разложении функции, представленной треугольным кодом, имеющим $(n+1)$ столбцов, равен

$$\xi(\omega, n) = |D_\omega > 0|_{\max} + |D_\omega < 0|_{\max} + 1.$$

Модули чисел D_ω (присутствующие в этой формуле) и их свойства описаны в разделе 4.5. На основании этого можно рассчитать числа $\xi(\omega, n)$. В табл. 5.4.1 приведены эти числа $\xi(\omega, n)$ при $\omega \leq 5$, $n \leq 5$.

Таблица 5.4.1. Числа $\xi(\omega, n)$.

$n \setminus \omega$	0	1	2	3	4	5
0	4					
1	16	13				
2	58	61	19			
3	202	253	115	25		
4	700	973	523	169	31	
5	2450	3685	2155	901	247	37

Далее, в табл. 5.4.2 приведены для каждого n числа $\xi_{\max}(n) = \max_{\omega}(\xi(\omega, n))$ и разрядность b_M двоичного кода этих чисел.

Таблица 5.4.2. Числа $\xi_{\max}(n)$.

n	0	1	2	3	4	5
$\xi_{\max}(n)$	4	16	61	253	973	3685
b_M	2	4	6	8	10	12

Из этой таблицы следует, что

$$b_M = 2(n+1) \quad (7.2)$$

5.4.2. Разрядность.

Пусть допустимая относительная погрешность такова, что максимальный коэффициент ряда Ω -ранга должен представляться b_D -разрядным двоичным кодом. Количество коэффициентов D_ω^i ряда равно Ω . Из (7.1) и (7.2) имеем:

- разрядность компьютера Т равна $b_T = b_D$;
- разрядность массива кодов, представляющих функцию $\Phi(x)$ в компьютере Р, равна $b_P = 4nb_D$;
- разрядность компьютера F, т.е. разрядность ТТК (или разрядность 4-х ТК), равна $b_F = 2(n+1)(n+2)$.

Итак,

$$\begin{aligned} b_T &= b_D = 2(n+1), \\ b_P &= 4nb_D = 8n(n+1), \\ b_F &= 2(n+1)(n+2). \end{aligned}$$

5.4.3. Объем арифметического устройства.

Этот объем определяется, в основном, разрядностью регистров и типом сумматора (при данном наборе команд). Как показано в разделе 5.1.1, объем одnorазрядного сумматора ТК превышает объем обычного одnorазрядного сумматора в 4 раза. Но FAU содержит сумматор ТК, разрядность которых в 4 раза меньше разрядности ТТК. Таким образом, относительный (отнесенный к разрядности) объем сумматора FAU совпадает с относительным объемом сумматора в компьютере Т. Следовательно, объемы арифметических устройств сравниваемых компьютеров относятся как разрядности этих компьютеров - см. табл. 5.4.3.

Таблица 5.4.3. Относительный объем

Компьютер	Относительный объем
Т	(2)
Р	(4n)
F	(n+2)

5.4.4. Длительность элементарных операций.

Длительность алгебраического сложения пропорциональна длине цепочки распространения переносов. В сумматоре компьютера Т эта длительность равна $t_T^S = \tau \cdot b_D = 2\tau(n+1)$, где τ - время задержки в одном разряде. В сумматоре компьютера Р эта длительность такая же и, поскольку все сумматоры работают параллельно, длительность алгебраического сложения в компьютере Р равна $t_P^S = t_T^S$.

Длина цепочки распространения переносов в ТК равна сумме длин «катетов ТК», увеличенной вдвое, т.к. переполнение может вдвое увеличить количество столбцов суммарного кода. Поэтому длительность алгебраического сложения в компьютере F равна $t_F^S = 4\tau(n+1)$.

Длительность умножения пропорциональна количеству разрядов множителя. Поэтому в компьютере T длительность умножения равна $t_T^m = t_T^S b_D = 4\tau(n+1)^2$. В компьютере P длительность умножения всех коэффициентов на один из них такая же, т.е. равна $t_T^m = t_P^S$. Длительность умножения в компьютере F равна $t_F^m = t_F^S b_F = 8\tau(n+1)^2(n+2)$.

Длительность деления в компьютере T будем считать равным длительности умножения, т.е. $t_T^d = t_T^m$. Длительность деления в компьютере P всех коэффициентов на один из них такая же, т.е. равна, т.е. $t_P^d = t_P^m$.

Длительность деления на 4 в компьютере F равна $t_F^d = 2t_F^S = 8\tau(n+1)$, что следует из алгоритма этой операции.

Длительность округления в компьютере F равна $t_F^o = 2nt_F^S = 8\tau \cdot n(n+1)$, что следует из алгоритма этой операции.

Длительность укорочения в компьютере F равна $t_F^u = 2n(t_F^d + t_F^o) = 16\tau \cdot n(n+1)^2$, что также следует из алгоритма этой операции.

5.4.5. Взаимосвязь между элементарными операциями и операциями с функциями.

В компьютере F операции с функциями соответствуют элементарным операциям. В компьютере T операции с функциями выполняются по некоторым очевидным программам. В компьютере P используются групповые операции, а операции с функциями также выполняются по некоторым очевидным программам, Включающим и групповые операции. Кроме того, эти программы содержат некоторое количество операций управления и доступа к данным, которыми мы в дальнейших расчетах пренебрегаем (хотя это и ухудшает расчетные характеристики компьютере F).

В табл. 5.4.4 и 5.4.5 приведено количество и длительность операций в сравниваемых компьютерах.

Таблица 5.4.4. Длительность умножения

Комп	Операции	Длительность	
Т	$16n^2$ умножений $16n^2$ сложений	$16n^2(t_T^s + t_T^m)$	$32\tau \cdot n^2(n+1)(2n+3)$
Р	$4n$ умножений $4n$ сложений	$4n(t_P^s + t_P^m)$	$8\tau \cdot n(n+1)(2n+3)$
Ф	1 умножение 1 укорочение	$(t_F^m + t_F^u)$	$8\tau \cdot (n+1)^2(3n+2)$

Таблица 5.4.5. Длительность сложения

Комп.	Операции	Длительность	
Т	$4n$ сложений	$4nt_T^s$	$8\tau \cdot n(n+1)$
Р	1 сложение	t_P^s	$2\tau \cdot (n+1)$
Ф	1 сложение	(t_F^s)	$4\tau \cdot (n+1)$

5.4.6. Выводы.

Результаты сравнения компьютеров сведены в табл. 5.4.6, где приведены их сравнительные характеристики. Они показывают, что компьютеры Р и Ф сравнимы по быстродействию, но компьютер Р в 4 раза больше компьютера Ф по объему. Следовательно, дальнейшему сравнению подлежат только компьютеры Ф и Т.

Учитывая (7.1), замечаем, что $B_F/B_T \approx n/2 = \frac{\Omega}{8}$ и $U_T/U_F \approx 4n = \Omega$.

Таблица 5.4.6. Сравнение компьютеров.

Комп	Относительный объем (В)	Длительность умножения	Относительная длительность умножения (U)
Т	(2)	$32\tau \cdot n^2(n+1)(2n+3)$	$\approx (4n)$
Р	(4n)	$8\tau \cdot n(n+1)(2n+3)$	≈ 1
Ф	(n+2)	$8\tau \cdot (n+1)^2(3n+2)$	≈ 1

Таким образом,

у компьютера F по сравнению с компьютером T
быстродействие увеличивается в Ω раз за счет
увеличения объема в $\frac{\Omega}{8}$ раз.

Следовательно, быстродействие компьютера F увеличивается в 8 раз быстрее увеличения объема.

Обозначения

Add - сумматор М-кодов,

A_v - коэффициент функционального ряда по функциям $\psi_{ok}(x)$ – см. теорему 2.1.5,

a_{ω}^i - числа, определенные в теореме 2.1.3,

c^i - числа, определенные в теореме 2.1.4,

CoderPM - кодер положительного Р-кода в М-код,

D_{ω}^i - коэффициент функционального ряда по функциям $\mathcal{E}_{\omega}^i(x)$ – см. теорему 2.1.5,

DecoderMP - декодер Р-кода в М-код,

Deven – одноразрядная схема декодирования для разряда с четным номером,

Dodd - одноразрядная схема декодирования для разряда с нечетным номером,

E_{ω}^i - коэффициент функционального ряда по функциям $\gamma_n^i(x)$ – см. теорему 2.1.5,

$[F \parallel h_k]$ - множество коэффициентов h_k функционального ряда для функции F ,

FAU - арифметическое устройство для операций с функциями,

H_{ω}^i - коэффициент функционального ряда по функциям $\lambda_{\omega}^i(x)$ – см. теорему 2.1.5,

Inv - инвертор М-кода,

InvAdd - инверсный сумматор М-кодов,

$K_R(A)$ - код числа A по основанию R ,

$L^i(v, n)$ - числа, определенные в теореме 2.1.2,

Meven – одноразрядная схема кодирования для разряда с четным номером,

Modd - одноразрядная схема кодирования для разряда с нечетным номером,

mDecoderMP - полный декодер Р-кода в М-код,

nSign - знакоопределитель М-кода,

$S^i(v, \omega)$ - числа, определенные в теореме 2.1.1,

М-код - код вещественного числа по отрицательному основанию «-2»,

Р-код - прямой код положительного числа по основанию «2»,

$PK_R(F(x))$ - прямоугольный код функции $F(x)$,

$PRK(F(x, v))$ - пирамидальный код функции $F(x, v)$,

Seven – одноразрядная схема знакоопределителя для разряда с четным номером,

$SK_R(F(x))$ - смешанный код функции $F(x)$,

Sodd - одноразрядная схема знакоопределителя для разряда с нечетным номером,

$STK_R(F(x))$ - ступенчатый код функции $F(x, v)$,

Sub – вычитатель М-кодов,

$TK_R(F(x))$ - треугольный код функции $F(x)$,

$TPRK_R(\Phi(x, v))$ - тригонометрический пирамидальный код функции $\Phi(x, v)$,

$TTK_R(F(x))$ - тригонометрический треугольный код функции $F(x)$,

$\Lambda K_R(f(x))$ - линейный код функции $f(x)$.

$\mathcal{E}_\omega^i(x)$ - функции, определенные в теореме 2.1.4,

$\gamma_n^i(x)$ - функции, определенные в теореме 2.1.3,

$\eta^i(x)$ - функции, определенные в теореме 2.1.4,

$\lambda_\omega^i(x)$ - функции, определенные в теореме 2.1.4,

ψ_{mk} - вес mk -разряда в треугольного кода по основанию $y = \sin^2(x)$,

ψ'_{mk} - вес mk -разряда в тригонометрического треугольного кода.

Хмельник С. И.

Специализированная ЦВМ для операций с комплексными числами.

Вопросы радиоэлектроники, серия XII, выпуск 2, 1964 *(в конце статьи указано, что поступила в редакцию в марте 1962 г.)*

Это – исторически первая в мире статья по кодированию комплексных чисел

Специализированная ЦВМ

1. Представление комплексных чисел в ЦВМ в виде единого кода, что должно упростить программирование.

2. Выполнение операций с комплексными числами (по крайней мере арифметических и выделения \Im и \Re комплексного числа) непосредственно в арифметическом устройстве машины без спецпрограммы.

3. Представление в запоминающем устройстве машины как кодов комплексных чисел, так и обычных кодов действительных чисел для осуществления внешних связей машины.

Машина, отвечающая указанным требованиям, потенциально имеет значительные достоинства. Действительно, возможность производить операции непосредственно с комплексными числами, а не сводить их к операциям с мнимыми и действительными частями (или с модулями и аргументами) приводит к сокращению общего числа операций, необходимых для решения задачи с комплексными числами. Последнее облегчает программирование и уменьшает время решения задачи на цифровой машине. Зависимости между количеством элементарных действий для некоторых операций с комплексными числами, представленными в общем виде или в алгебраической форме, приведены в табл. I.

Недостатком подобной машины является неизбежное усложнение конструкции. Соотношение между указанными преимуществами и

Т а б л и ц а I

Исходные числа		Операция	Формулы для определения результата		Соотношение между числом операций
$Z_1 = U_1 + i V_1$	$Z_2 = U_2 + i V_2$		в общем виде	в алгебраической форме	
Слагаемое	Слагаемое	Алгебраическое сложение	$Z_1 \pm Z_2$	$U = U_1 \pm U_2$ $V = V_1 \pm V_2$	I : 2
Множимое	Множитель	Умножение	$Z_1 \cdot Z_2$	$U = U_1 U_2 - V_1 V_2$ $V = U_1 V_2 + V_1 U_2$	I : 6
Делимое	Делитель	Деление	$\frac{Z_1}{Z_2}$	$U = \frac{U_1 U_2 + V_1 V_2}{U_2^2 + V_2^2}$ $V = \frac{V_1 U_2 - U_1 V_2}{U_2^2 + V_2^2}$	I : II
Подынтегральная функция	Аргумент	Вычисление определенного интеграла	$\int_{a+bi}^{c+di} Z_1 \cdot dZ_2$	$U = \int_a^c u \cdot du - \int_b^d v_1 dv_2$ $V = \int_a^c v_1 du_2 + \int_b^d u \cdot dv_2$	I : 4

недостатком не может быть определено предварительно. Однако необходимость изучения вопроса о возможности построения машины для операций с комплексными числами не вызывает сомнений, если учесть широкое распространение задач с функциями комплексного переменного в различных областях науки и техники: энергетике, связи, теории упругости, гидродинамике и т.д. По-видимому, задачи, связанные с определением точки в трехмерном пространстве (задачи обнаружения, наведения, преследования), также можно свести к задачам с плоскими векторами, т.е. с комплексными числами.

1. КОДИРОВАНИЕ КОМПЛЕКСНЫХ ЧИСЕЛ

Представление чисел в ЦВМ основано на кодировании чисел различными методами (коды: прямой, обратный и дополнительный, в остатках, двоично-десятичные и т.д.). Наиболее широкое распространение получил метод позиционного кодирования, основанный на разложении числа по степеням другого числа, называемого основанием. Подобное разложение имеет вид

$$Z = \sum_{k=-\infty}^M \alpha_k \cdot \rho^k, \quad (I)$$

где Z — некоторое число;

ρ — основание разложения;

k — номер разряда разложения (целое положительное или отрицательное число);

$\alpha_k = 0, 1, 2, 3, 4, \dots, (R - 1)$ — коэффициент k -го разряда разложения (R — целое положительное число).

В дальнейшем выражение (I) будем называть R -м разложением числа Z по основанию ρ . В настоящее время известны два вида такого разложения:

а) разложение действительных положительных чисел по основанию $\rho = R$, применяемое для построения прямого, обратного и дополнительного кодов действительных чисел;

б) разложение относительных действительных чисел по основанию $\rho = -R$ [1], [2], [3].

Широкое распространение позиционного кодирования объясняется тем, что алгоритмы арифметических операций с разложениями чисел просты и легко реализуются в цифровых вычислительных устройствах. Так, если имеется разложение числа

$$R = \sum_{k=1}^m \alpha_k \rho^k, \quad (2)$$

то сложение нескольких R -х разложений заключается в поразрядном сложении коэффициентов α_k этих разложений с учетом переносов, имеющих вид разложения (2). Все другие арифметические действия сводятся к операциям сложения и умножения на степень основания ρ^k .

Это обстоятельство наводит на мысль о целесообразности применения метода позиционного кодирования для представления комплексных чисел в ЦВМ в виде единого кода. Возможность позиционного кодирования комплексных чисел показывается теоремами 4-7, доказательство которых построено на использовании теорем 1-3, приведенных в [1].

Т е о р е м а 1. Для любого действительного числа Z существует R -е разложение по основанию $\rho = -R$.

Т е о р е м а 2. Если разложение теоремы 1 конечно (k - ограничено), то оно единственно.

Т е о р е м а 3. Неединственным образом по теореме 1 разлагаются и имеют два отличающихся друг от друга бесконечных разложения все числа вида

$$V(R) = \frac{\pm R^k}{1+R} + CR^{k+1}, \quad (3)$$

где C, k - любые целые числа.

Т е о р е м а 4. Для любого комплексного числа $Z = u + i v$ существует R -е разложение по основанию $\rho = \pm i \sqrt{R}$, где $i^2 = -1$.

Д о к а з а т е л ь с т в о. Согласно теореме 1 имеем

$$u = \sum_{m=-\infty}^{M_1} \alpha'_m (-R)^m, \quad \alpha'_m = 0, 1, 2; \dots, (R-1)$$

или

$$u = \sum_{m=-\infty}^{M_1} \alpha'_m (\pm i \sqrt{R})^{2m}.$$

Очевидно,

$$iU = \pm i\sqrt{R} \left(\pm \frac{U}{\sqrt{R}} \right),$$

но действительное число

$$\pm \frac{U}{\sqrt{R}} = \sum_{m=-\infty}^{M_2} \alpha_m'' (\pm i\sqrt{R})^{2m}, \quad \alpha_m'' = 0, 1, 2, \dots, (R-1),$$

следовательно,

$$iU = \pm i\sqrt{R} \cdot \sum_{m=-\infty}^{M_2} \alpha_m'' (\pm i\sqrt{R})^{2m+1}$$

или

$$iU = \sum_{m=-\infty}^{M_2} \alpha_m'' (\pm i\sqrt{R})^{2m+1}.$$

Введем следующие обозначения:

$$\alpha_m' = \alpha_{2m};$$

$$\alpha_m'' = \alpha_{2m+1};$$

$$M = \max(M_1; M_2).$$

Тогда

$$Z = \sum_{k=-\infty}^M \alpha_k (\pm i\sqrt{R})^k, \quad (4)$$

где

$$\alpha_k = 0, 1, 2, \dots, (R-1),$$

что и требовалось доказать.

Т е о р е м а 5. Для любого комплексного числа $Z = U + iV$ существует R -е разложение по основанию

$$\rho = \sqrt{\frac{R}{2}} (-1 \pm i),$$

где $\sqrt{\frac{R}{2}}$ - целое положительное число.

Доказательство. Имеем

$$\rho = \sqrt{\frac{R}{2}}(-1 \pm i);$$

$$\rho^2 = \pm Ri;$$

$$\rho^3 = R\sqrt{\frac{R}{2}}(1 \pm i).$$

Покажем, что число R имеет разложение по ρ . Согласно (2)

$$R = \sum_{k=1}^3 \alpha'_k \rho^k,$$

где

$$\alpha'_3 = 1,$$

или

$$R = \alpha'_1 \sqrt{\frac{R}{2}}(-1 \pm i) + \alpha'_2 R(\pm i) + R\sqrt{\frac{R}{2}}(1 \pm i).$$

Преобразуем последнее уравнение в систему

$$\left. \begin{aligned} R &= \alpha'_1 \left(-\sqrt{\frac{R}{2}}\right) + R\sqrt{\frac{R}{2}}; \\ \pm \alpha'_1 \sqrt{\frac{R}{2}} \mp \alpha'_2 R \pm R\sqrt{\frac{R}{2}} &= 0, \end{aligned} \right\}$$

решив которую, найдем

$$\alpha'_1 = R - \sqrt{2R};$$

$$\alpha'_2 = \sqrt{2R} - 1.$$

Коэффициенты α'_k — числа целые и меньшие R , следовательно, число R имеет разложение по ρ в виде (2).

Согласно теореме 2 любое комплексное число Z имеет разложение по основанию ρ, ρ^2 , так как $\rho^2 = \pm i\sqrt{R^2}$.

Итак,

$$Z = \sum_{m=-\infty}^{M'} \beta_m (\rho^2)^m,$$

где

$$\beta_m = 0, 1, 2, \dots, (R^2 - 1).$$

Обозначим

$$2m = k;$$

$$\beta_m = \beta'_k;$$

$$2M' = M''.$$

Тогда

$$Z = \sum_{k=-\infty}^{M''} \beta'_k \rho^k.$$

Так как $\beta'_k < R^2$, то

$$\beta'_k = \gamma_k R + \epsilon_k,$$

где

$$\gamma_k = 0, 1, 2, \dots, (R-1)$$

и

$$\epsilon_k = 0, 1, 2, \dots, (R-1).$$

Следовательно,

$$Z = \sum_{k=-\infty}^{M''} (\gamma_k R + \epsilon_k) \rho^k$$

или

$$Z = R \sum_{k=-\infty}^{M''} \gamma_k \rho^k + \sum_{k=-\infty}^{M''} \epsilon_k \rho^k.$$

В последнюю формулу в качестве сомножителей и слагаемых входят R -е разложения по основанию ρ . Следовательно, число Z также имеет R -е разложение по основанию ρ , т.е.

$$Z = \sum_{K=-\infty}^M \alpha_K \left[\sqrt{\frac{R}{2}} (-1 \pm i) \right]^K, \quad (5)$$

где

$$\alpha_k = 0, 1, 2, \dots, (R-1);$$

что и требовалось доказать.

Единственность и конечность разложений комплексных чисел оцениваются теоремами 6 и 7, которые являются следствием вышеприведенных теорем.

Т е о р е м а 6. Если разложения теоремы 4 конечны, то они единственны; неединственным образом разлагаются и имеют два (или четыре) отличающихся друг от друга бесконечных разложения все числа $Z = u + i\bar{v}$, у которых u или \bar{v} (или u и \bar{v} одновременно) определяются формулами

$$u = V(R);$$

$$\bar{v} = V(R)\sqrt{R};$$

числа $V(R)$ находятся по формуле 3.

Д о к а з а т е л ь с т в о. Разложения вида (4) получают-ся путем ограниченного числа конечных преобразований разложений действительных чисел по основанию $\rho = -R$.

Каждая пара разложений чисел u и $\pm \frac{\bar{v}}{\sqrt{R}}$ по $\rho = -R$ образует одно разложение комплексного числа $Z = u + i\bar{v}$ по основанию $\rho = \pm i\sqrt{R}$, причем последнее разложение будет конечным тогда и только тогда, когда оба исходных разложения конечны. Последнее условие не выполняется, если

$$u = V(R);$$

$$\pm \frac{\bar{v}}{\sqrt{R}} = V(R).$$

Т е о р е м а 7. Если разложения теоремы 5 конечны, то они единственны; неединственным образом разлагаются и имеют два (или четыре) отличающихся друг от друга бесконечных разложения все числа $Z = u + i\bar{v}$, у которых u или \bar{v} (или u и \bar{v} одновременно) определяются формулами

$$u = V(R^2);$$

$$\bar{v} = V(R^2)R.$$

Доказательство совершенно аналогично доказательству теоремы 6. Следует только иметь в виду, что исходные разложения берутся в этом случае по основанию $\rho = \pm i\sqrt{R^2}$.

Приведенные теоремы дают несколько способов получения разложений комплексных чисел. Соответствующие позиционные коды строятся в виде

$$K(Z) = \alpha_M \dots \alpha_k \dots \alpha_0, \alpha_{-1}, \alpha_{-2}, \dots \quad (6)$$

Между R -м разложением (I) числа Z и R -м позиционным кодом (6) того же числа существует взаимно-однозначное соответствие. заключающееся в том, что коэффициенту α_k k -го разряда разложения соответствует цифра α_k , стоящая в k -м разряде кода (номера разрядов кода отсчитываются влево от запятой).

Для примера укажем, что существует четыре двоичных позиционных кода комплексного числа, соответствующих двоичному разложению по основаниям $\rho = -1 \pm i$ и $\rho = \pm i\sqrt{2}$. Например, при

$$\begin{aligned} \rho &= i - 1 \\ k(-1) &= \text{IIIOI}; \\ k(2) &= \text{II00}; \\ k(i) &= \text{II}; \\ k(4-i) &= \text{IIIOIOIII}; \\ k(-5+3i) &= \text{IIIOIOOIIIO} \quad \text{и т.д.} \end{aligned}$$

2. ОПЕРАЦИИ С ПОЗИЦИОННЫМИ КОДАМИ

Пусть Z_1 и Z_2 - некоторые комплексные числа, а Z - результат операции ∇ с этими числами

$$Z = Z_1 \nabla Z_2.$$

Рассмотрим R -е позиционные коды чисел Z_1, Z_2, Z по основанию ρ

$$K(Z_1) = \dots \alpha_k \dots; \quad (7)$$

$$K(Z_2) = \dots \beta_k \dots; \quad (8)$$

$$K(Z) = \dots \sigma_k \dots, \quad (9)$$

где k - номер разряда кода;
 $\alpha_k; \beta_k; \sigma_k$ - цифры k -го разряда кода, которые принимают целочисленные значения от 0 до $(R-1)$.

Операции с кодами (7) и (8), в результате которой получается код (9), будем также называть операцией ∇ с кодами

$$K(Z) = K(Z_1) \nabla K(Z_2). \quad (10)$$

Из всех операций с числами и соответствующими позиционными кодами выделим класс поразрядных операций Δ , которые определяются следующим образом.

Если над числами Z_1 и Z_2 , имеющими R -е разложение по основанию ρ , производится операция Δ , то имеет место равенство

$$\sum_{k=n}^M (\alpha_k \Delta \beta_k) \rho^k = \left[\sum_{k=n}^M \alpha_k \rho^k \right] \Delta \left[\sum_{k=n}^M \beta_k \rho^k \right]. \quad (11)$$

3. ПЕРВЫЙ АЛГОРИТМ ПОРАЗРЯДНЫХ ОПЕРАЦИЙ

Операция Δ с кодами (7), (8)

$$K(Z) = K(Z_1) \Delta K(Z_2)$$

производится над каждой парой чисел (α_k, β_k) с учетом некоторого переноса π_k по формуле

$$Q_k + \pi_k = S_k, \quad (12)$$

где $Q_k = \alpha_k \Delta \beta_k$ - результат операции Δ над k -ми разрядами кодов;

S_k - разрядный результат.

Позиционный R -й код разрядного результата по основанию ρ имеет вид

$$K(S_k) = \gamma_m \dots \gamma_i \dots \gamma_0. \quad (13)$$

Код (13), у которого $\gamma_i = 0$ при всех $i < 0$, назовем целым кодом. Обозначим

$$\gamma_0 = \sigma_k, \quad (14)$$

где σ_k - число, стоящее в k -м разряде кода (9).

Тогда код переноса выражается как

$$K(\pi_{k+1}) = \gamma'_{m-1} \dots \gamma'_i \dots \gamma'_0, \quad (15)$$

где

$$\gamma'_i = \gamma_{i+1}.$$

Таким образом,

$$S_k = \sigma_k + \pi_{k+1} \rho.$$

Итак, поразрядная операция над позиционными кодами выполняется в следующей последовательности.

А л г о р и т м I. Известны коды (7) и (8).

1. Определяется S_k по формуле (12).
2. Определяется $K(S_k)$ в виде кода (13).
3. Определяются σ_k по (14) и $K(\pi_{k+1})$ по (15).
4. Определяется число π_{k+1} по известному коду (15).
5. Операции 1-4 осуществляются для $(k+1)$ -го разряда.

Алгоритм I выполняется начиная с младшего разряда кодов (7) и (8). Последним разрядом является тот разряд M , для которого соблюдаются условия

$$\left. \begin{aligned} \pi_{M+1} &= 0; \\ \alpha_k &= 0 \quad \text{при } k > M; \\ \beta_k &= 0 \quad \text{при } k > M. \end{aligned} \right\} \quad (16)$$

Результатом выполнения алгоритма I является код (9).

Для осуществления поразрядной операции по алгоритму I, т. е. последовательно, разряд за разрядом, необходимо, чтобы код (13) всегда был целым. Естественно, код (15) при этом также будет целым.

Покажем, в каких случаях выполняется данное условие. Рассмотрим несколько чисел, имеющих R -разложение по основанию ρ , не содержащие отрицательных степеней числа ρ . Очевидно, и разложение суммы этих чисел также не будет содержать отрицательных степеней числа ρ .

Специализированная ЦВМ

Последнее утверждение равносильно следующему: сумма целых кодов также является целым кодом. Отсюда следует, что для того, чтобы в уравнении (12) число S_k имело целый код, необходимо и достаточно наличие целого кода у числа Q_k .

Итак, операция является поразрядной, если выполняются следующие условия:

- 1) действительно равенство (II);
- 2) код $K(\alpha_k \Delta \beta_k)$ — целый.

4. НЕКОТОРЫЕ ПОРАЗРЯДНЫЕ ОПЕРАЦИИ

К числу поразрядных операций относятся: сложение, умножение на -1 , вычитание и некоторые другие операции.

Сложение. Пусть

$$K(Z) = \sum_{i=1}^S K(Z_i).$$

Справедливость условия 1) в этом случае очевидна. Результат сложения k -х разрядов $\alpha_k^{(i)}$ кодов $K(Z_i)$

$$Q_k = \sum_{i=1}^S \alpha_k^{(i)}$$

имеет целый код, так как числа $\alpha_k^{(i)}$ имеют целый код.

Таким образом, выполняются оба условия. Следовательно, сложение — поразрядная операция.

Заметим, что для определения $K(S_k)$ в формуле (12) необходимо знать код числа R . Код $K(R)$ для различных оснований приведен в табл.2.

Т а б л и ц а 2

p	R	$K(R)$	$K(-1)$
$p = \pm i\sqrt{R}$	$R > 1$ целое	$K(R) = \delta_0^R \delta_1^R \delta_2^R \delta_3^R$ $\delta_0^R = 1$ $\delta_1^R = R-1$ $\delta_2^R = \delta_3^R = 0$	$K(-1) = \delta_0^R \delta_1^R \delta_2^R \delta_3^R$ $\delta_0^R = 1$ $\delta_1^R = 0$ $\delta_2^R = R-1$
$p = -1 \pm i$	$R = 2$	$K(R) = 1100$	$K(-1) = 1101$
$p = \sqrt{\frac{R}{2}}(1 \pm i)$	$\frac{\sqrt{R}}{2} \neq 2$ целое	$K(R) = \delta_0^R \delta_1^R \delta_2^R \delta_3^R$ $\delta_0^R = 1$ $\delta_1^R = \sqrt{2R} - 1$ $\delta_2^R = R - \sqrt{2R}$ $\delta_3^R = 0$	$K(-1) = \delta_0^R \delta_1^R \delta_2^R \delta_3^R$ $\delta_0^R = 1$ $\delta_1^R = \sqrt{2R}$ $\delta_2^R = R-1$

Умножение на постоянный коэффициент α , если $K(\alpha)$ — целый код. Пусть

$$K(Z) = K(\alpha) \cdot K(Z_1).$$

Результат умножения k -го разряда

$$Q_k = \alpha_k \cdot \alpha,$$

т.е. определяется α_k — кратным сложением числа α . Таким образом, данная операция сводится к операции I, т.е. является поразрядной.

Инвертирование (умножение на -1). Для доказательства поразрядности этой операции достаточно показать, что число -1 имеет целый код, т.е.

$$K(-Z_2) = -K(Z_2) = K(-1) \cdot K(Z_2).$$

Код $K(-1)$ для различных оснований также приведен в табл.2.

Вычитание.

$$K(Z) = K(Z_1) - K(Z_2)$$

или

$$K(Z) = K(Z_1) + [-K(Z_2)].$$

Таким образом, вычитание состоит из двух поразрядных операций и, следовательно, также является поразрядной операцией.

5. ВТОРОЙ АЛГОРИТМ ПОРАЗРЯДНЫХ ОПЕРАЦИЙ

Рассмотрим снова формулу (I2) и код (I3). Как и ранее, $y_0 = e_k$. Обозначим

$$y_i = p_{k+i}. \quad (I7)$$

Назовем p_{k+i} одnorазрядным переносом в $(k+i)$ -й разряд. Пусть число переносов в k -й разряд равно числу e_k . Определим численное значение переноса π'_k по формуле

$$\pi'_k = \sum_{j=1}^{e_k} p_{kj}. \quad (I8)$$

Оно всегда является действительным целым и имеет целый код

$$K(\pi'_k) = \delta_e \dots \delta_t \dots \delta_o.$$

Для всех поразрядных операций ($K(Q_k)$ - целый код) такая организация переноса не нарушает условия: $K(S_k)$ - целый код.

Таким образом, поразрядные операции можно выполнять также по следующему алгоритму.

А л г о р и т м 2. Известны коды (7) и (8).

1. Определяется число S_k по формуле (6).
2. Определяется код (13).
3. Определяются σ_k по (14) и p_{k+i} ($i = 1 \div m$) по (17).
4. Определяется число π_{k+1} по (18).
5. Операции 1-4 осуществляются для $(k + 1)$ -го разряда.

Алгоритм 2 выполняется начиная с младшего разряда кодов (7) и (8). Последним разрядом является тот разряд M , для которого выполняются условия (16). Результатом выполнения алгоритма 2 является код (9).

6. РЕАЛИЗАЦИЯ АЛГОРИТМОВ 1 И 2 В ЦВМ

Синтез схемы, выполняющей поразрядную операцию над позиционными кодами, разбивается на два этапа:

- составление одnorазрядных таблиц чисел по форме табл.3;
- разработка собственно схемы.

Составление таблиц чисел ведется по соответствующим алгоритмам, причем таблица для одного разряда является законченной

Т а б л и ц а 3

α_k	β_k	Q_k	π_k	S_k	$K(S_k)$	σ_k	π_{k+1}
α_k	β_k	$\alpha_k \Delta \beta_k$	π_k или π'_k	S_k	$\gamma_m \dots \gamma_i \dots \gamma_0$	γ_0	π_{k+1} или $p_{k+i} = \gamma_i$ ($1 \leq i \leq m$)

только в том случае, если удовлетворяет следующим двум условиям

а) если перенос Π_{k+1} принимает некоторое численное значение, то перенос Π_k также должен принимать это значение (или перенос Π'_k должен принимать все возможные значения, соответствующие любой комбинации чисел ρ_{k+i});

б) в табл.3 должна присутствовать любая комбинация чисел $\alpha_k, \beta_k, \Pi_k(\Pi'_k)$.

По табл.3 строится схема одноразрядной операции Δ , содержащая три группы входов: α_k, β_k и $\Pi_k(\rho_k)$ и две группы выходов: \mathcal{G}_k и $\Pi_{k+1}(\rho_{k+i})$. Строение схемы зависит от вида операции Δ , величин ρ, R , выбора исходных элементов схемы и т.д. и не может быть описано в общем виде.

На основе подобных одноразрядных схем легко синтезируются различные схемы для операций Δ с многоразрядными кодами: последовательные и параллельные, накапливающие и комбинационные. При этом для выполнения комбинационных схем удобно пользоваться одноразрядными схемами, основанными на применении алгоритма 2, а для накапливающих — основанными на применении алгоритма I. В последнем случае легко организуется сквозной перенос.

Составление таблиц чисел и логику схемы удобно рассмотреть на примерах с двоичным кодом по основанию $\rho = i-1 (R=2)$.

Пример I. Инвертор кода

Имеется код

$$K(Z) = \dots \beta_k \dots$$

Необходимо найти код

$$K(-Z) = \dots \mathcal{G}_k \dots$$

Пользуясь алгоритмом I, составляем табл.4, удовлетворяющую вышеуказанным условиям α и δ . На основе данных этой таблицы строим схему инвертора на триггерах со сквозным переносом (рис.1).

Работа схем сквозного переноса и счетного входа триггеров описывается следующими уравнениями алгебры логики, непосредственно вытекающими из данных таблицы:

$$\Delta_{k+1} = \bar{\beta}_k \Delta_k \vee [1]_k ;$$

$$[1+i]_{k+1} = \beta_k \Delta_k ;$$

Пример 2. Сумматор кодов

Слагаемые коды имеют вид

$$K(Z_1) = \dots \alpha_k \dots ;$$

$$K(Z_2) = \dots \beta_k \dots$$

Необходимо найти код суммы

$$K(Z_1 + Z_2) = \dots \sigma_k \dots$$

Для составления табл.5 воспользуемся алгоритмом 2. В этом случае формула (12) записывается так:

$$S_k = \alpha_k + \beta_k + \sum_{j=1}^{e_k} p_{kj}.$$

Легко проверить, что табл.5 удовлетворяет требованиям α и β . Характерным для данной таблицы является то, что любое p_{kj} может принимать значения 0 или 1.

Т а б л и ц а 5

S_k	$K(S_k)$	σ_k	p_{k+2}	p_{k+3}	p_{k+4}	p_{k+5}	p_{k+7}	p_{k+8}
0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0
2	1100	0	1	1	0	0	0	0
3	1101	1	1	1	0	0	0	0
4	1110100000	0	0	0	1	1	1	1
5	111010001	1	0	0	1	1	1	1
6	111011100	0	1	1	1	1	1	1
7	111011101	1	1	1	1	1	1	1
8	1110000000	0	0	0	0	1	1	1

Процесс составления таблиц по алгоритму 2 неоднозначен. Действительно, можно составить еще две таблицы (6 и 7), отвечающие этим требованиям.

Табл.6 и 7 имеют следующую особенность: код числа иногда записывается в виде

$$K(S_k) = -K(-S_k)$$

или

$$K(S_k) = -\gamma_m \dots \gamma_i \dots \gamma_0.$$

В этом случае

$$p_{k+i} = -y_i.$$

Таким образом, перенос p_{k+i} может принимать три различных значения: 0, 1, -1. Такое усложнение переноса приводит к упрощению таблиц, что видно непосредственно из сравнения табл. 5-7.

Т а б л и ц а 6

S_k	$K(S_k)$	b_k	p_{k+2}	p_{k+3}	p_{k+4}
0	0	0	0	0	0
-1	1 1 1 0 1	1	1	1	1
1	1	1	0	0	0
2	1 1 0 0	0	1	1	0
3	1 1 0 1	1	1	1	0
4 = -1-4/	-1 0 0 0 0	0	0	0	-1
5 = 1/4/ + 1	-1 0 0 0 0 + 1	1	0	0	-1

Т а б л и ц а 7

S_k	$K(S_k)$	b_k	p_{k+2}	p_{k+3}	p_{k+4}
0	0	0	0	0	0
1	1	1	0	0	0
2 = -1-2/	-1 1 1 0 0	0	-1	-1	-1
3 = -2/-2/ + 1	-1 1 1 0 0 + 1	1	-1	-1	-1
-1 = -1/2/ + 1	-1 1 0 0 + 1	1	-1	-1	0
-2 = -1/2/	-1 1 0 0	0	-1	-1	0
-3	1 0 0 0 1	1	0	0	1

Наиболее простой с точки зрения схемной реализации оказывается табл. 7. Перепишем ее в виде табл. 8, введя следующие обозначения:

$$p'_{k+i} = 1; \quad p''_{k+i} = 0 \quad \text{при} \quad p_{k+i} = 1;$$

$$p'_{k+i} = 0; \quad p''_{k+i} = 1 \quad \text{при} \quad p_{k+i} = -1;$$

$$p'_{k+i} = 0; \quad p''_{k+i} = 0 \quad \text{при} \quad p_{k+i} = 0.$$

Табл.8 описывает связь между числами $\alpha_k, \beta_k, \epsilon_k, p_{kj}$ как двоичными переменными и поэтому может быть использована для построения одноразрядного сумматора комбинационного типа обычными методами алгебры логики.

Т а б л и ц а 8

S_k	ϵ_k	$P_{k+2}'' = P_{k+3}''$	P_{k+4}'	P_{k+4}''
0	0	0	0	0
1	1	0	0	0
2	0	1	0	1
3	1	1	0	1
-1	1	1	0	0
-2	0	1	0	0
-3	1	0	1	0

На рис.2 изображена схема комбинационного последовательного сумматора, в котором используется одноразрядный сумматор, соответствующий табл.8.

Одноразрядный сумматор имеет шесть входов (два слагаемых α_k и β_k и четыре переноса из $(k-2), (k-3), (k-4)$ -го разрядов $p_{k1}, p_{k2}, p_{k3}, p_{k4}$) и четыре выхода $\epsilon_k, P_{k+2}'' = P_{k+3}'', P_{k+4}', P_{k+4}''$. Переносы p_{kj} образуются задержкой переносов p_{k+i} на время $i\tau$ (τ — длительность одного интервала). Величина S_k определяется по формуле

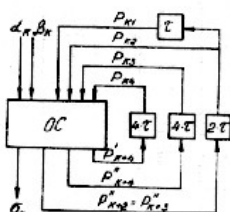


Рис.2

$$S_k = \alpha_k + \beta_k + p_{k4} - p_{k1} - p_{k2} - p_{k3}.$$

Перейдем к рассмотрению более сложных операций, не являющихся поразрядными.

7. УМНОЖЕНИЕ

Необходимо найти произведение

$$Z = Z_1 \cdot Z_2,$$

где числа Z_1 и Z_2 имеют соответственно коды

$$K(Z_1) = \alpha_M \dots \alpha_k \dots \alpha_m, \quad (19)$$

$$K(Z_2) = \beta_N \dots \beta_k \dots \beta_n. \quad (20)$$

Код произведения определяется из выражений

или
$$K(Z) = K(Z_1) \cdot K(Z_2)$$

или
$$K(Z) = \sum_{k=n}^N K(\beta_k \rho^k Z_1),$$

или
$$K(Z) = \sum_{k=n}^N K(\beta_k) K(\rho^k) K(Z_1),$$

где

$$K(\beta_k) = \beta_k;$$

$$K(\rho^k) = \underbrace{1000 \dots 000}_{k \text{ нулей}}.$$

Умножение кодов описывается следующим алгоритмом.

А л г о р и т м 3. Известны коды (19) и (20).

1. Определяется код

$$K(Z, \rho^k) = K(Z_1) K(\rho^k) = K(Z_1) \cdot 1000 \dots 000$$

сдвигом кода $K(Z_1)$ на k разрядов.

2. Определяется код

$$K(Z, \beta_k \rho^k) = K(Z, \rho^k) K(\beta_k) = \beta_k K(Z, \rho^k)$$

β_k - кратным сложением кода $K(Z, \rho^k)$.

3. Производится сложение $(k-1)$ -й частичной суммы

$$\sum_{k=n}^{k-1} K(\beta_k \rho^k Z_1)$$

с k -м частичным произведением

$$K(Z, \beta_k \rho^k),$$

т.е. определяется k -я частичная сумма.

4. Выполняются операции I-3 для $k_i = k+1$.

Алгоритм 3 выполняется последовательно для всех разрядов множителя от 17 до N (или в обратном порядке), причем первая частичная сумма равна 0, а последняя (N -я) является кодом произведения $k(Z)$.

8. ДЕЛЕНИЕ

Необходимо найти частное

$$Z = \frac{Z_1}{Z_2},$$

где числа Z_1 и Z_2 имеют соответственно коды (19) и (20).

Деление как действие, обратное умножению, производится путем последовательного вычитания делителя из делимого. Критерием правильности деления служит уменьшение номера старшего значащего разряда (СЗР) кода остатка, образующегося в результате вычитания.

А л г о р и т м 4. Известны коды (19) и (20).

1. Определяется код $K(Z_2 \rho^v)$ сдвигом делителя $K(Z_2)$ на $v = M - N - h$ разрядов.

2. Вычитается сдвинутый делитель $K(Z_2 \rho^v)$ из h -го остатка $K(Z_0^{(h)})$, т.е. определяется $(h+1)$ -й остаток;

3. Сравниваются номера СЗР h -го и $(h+1)$ -го остатков, обозначенных соответственно через H_h и H_{h+1} (табл.9).

Т а б л и ц а 9

	Условие	Результат		
		Частное	h	$K(Z_0)$
а	$H_{h+1} < H_h$	увеличивается на единицу h -го разряда	увеличивается на единицу	$K(Z_0^{(h+1)})$
б	$H_{h+1} = H_h$	то же	остается без изменений	то же
в	$H_{h+1} > H_h$	остается без изменений	увеличивается на единицу	$K(Z_0^{(h)})$

Специализированная ЦВМ

4. Выполняются операции 1-3 с новыми значениями числа h и остатка.

Деление производится, начиная с $h = 0$, при котором за остаток принимается делимое, и кончая получением заданного числа разрядов в частном. Частное образуется суммированием единиц v -го разряда, полученных операцией 3а (см. табл. 9), и имеет вид

$$K(Z) = \sigma_v \dots \sigma_v \dots \sigma_{(M-N)}.$$

Алгоритм 4 следующим образом связывает между собой коды $K(Z)$, $K(Z_1)$, $K(Z_2)$, $K(Z_0^{(v)})$:

$$K(Z_1) = \sum_{v=M-N}^v K(Z_2 \sigma_v \rho^v) + K(Z_0^{(v)}). \quad (21)$$

Откуда следует, что

$$Z = \frac{Z_1 - Z_0^{(v)}}{Z_2}. \quad (22)$$

Равенство (22) подтверждает справедливость алгоритма 4.

Алгоритмы 3 и 4 состоят из операций сложения, вычитания и сдвига и, следовательно, легко могут быть реализованы в ЦВМ.

ЗАКЛЮЧЕНИЕ

Все вышесказанное показывает возможность представления комплексных чисел в виде единых кодов и выполнения операций на ЦВМ непосредственно с этими кодами. Строение кодов комплексных чисел ничем не отличается от строения обычных кодов действительных чисел, что позволяет использовать общее запоминающее устройство для хранения тех и других кодов.

Таким образом, применение метода позиционного кодирования отвечает всем требованиям к ЦВМ, изложенным во введении.

ЛИТЕРАТУРА

1. Z. PAWLAK, A. WAKULICZ. Use of expansions with a negative basis in the arithmometer of a digital computer. "Bulletin de l'Academie polonaise des sciences". vol. 5, cl. 3, N 3, 1957.

Хмельник С.И.

2. Z.PAWLAK. An electronic digital computer based on the "-2" system."Bulletin de l'Academie polonaise des sciences"vol.7, cl.4, N 12, 1959.

3. IRE Transaction Electronic Computers, vol.6, 1957, p.123.

Статья поступила в редакцию в марте 1962 г.