Компьютерная арифметика геометрических фигур
Алгоритмы и аппаратура

Соломон Ицкович Хмельник

Россия Израиль
2004

Computer Arithmetic of Geometrical Figures

Algorithms and Hardware Design

(in Russian)

Solomon I. Khmelnik

Technical Editor, Cover Designer – Inna S. Doubson

Published by "MiC" - Mathematics in Computer Comp.
BOX 15302, Bene-Ayish, Israel, 79845
Fax: ++972-3-9471301
http://geo.mic34.com
Printed in United States of America, Lulu Inc.

Russia Israel
2004

Summary

This book describes various versions of processors, designed for affine transformations of many-dimensional figures – planar and spatial. This processors is oriented to affine transformation of unstructured geometrical figures with arbitrary points distribution. The type of data presentation used in this book is non-conventional, based on a not well-known theory of vectors and geometrical figures coding. The problems of affine transformation are used widely in science and engineering. The examples of their application are computer tomography and data compression for telecommunication systems.

The book covers the figures coding theory – the codes structure, algorithms of coding and decoding for planar and spatial figures, arithmetical operations with planar and spatial figures. The theory is supplemented by numerous examples. The arrangement of several versions of geometrical processor is considered – data representation, operating blocks, hardwares realization of coding, decoding and arithmetic operations algorithms. The processor's internal performance is appraised.

The book is designed for students, engineers and developers, who intend to use the computer arithmetic of geometrical figures in their own research and development in the field of specialized processors. With that in view the book includes

- Theory of coding,
- Operations algorithms,
- Examples of coding, decoding and computations,
- Description of several versions of processors,
- A system of commands for them,
- Schemes of operational units,
- Comparative analysis.

Algorithms and units described in this book are developed into models in VHDL and FPGA. We shall welcome any kind of cooperation proposals sent to the address:

solik@netvision.net.il

Аннотация

В книге 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 ипользуется нетрадиционная форма представления данных, основанная на малоизвестной теории кодирования векторов и геометрических фигур. Задачи аффинного преобразования пространства широко используются в науке и технике. Примерами их применения могут служить компьютерная томография и сжатие информации для телекоммуникационных систем.

В книге 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.

Книга ориентирована на студентов, инженеров и разработчиков, которые намерены применять компьютерную арифметику геометрических фигур в собственных разработках специализированных процессоров. С этой целью книга включает

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

Предлагаемые в книге алгоритмы и устройства разрабатываются в виде моделей на VHDL и FPGA. Любые предложения о сотрудничестве посылайте по адресу

solik@netvision.net.il

Оглавление

1. Введение

В книге 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, [1]) и сжатие информации для телекоммуникационных систем (см., например, [2]).

В книге 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.

Решение указанных задач 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 [3, 4] и рассмотрены также в [5-11]. При построении геометрического кода используется метод представления комплексных чисел и векторов единым двоичным кодом [11, 12, 13].

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 геометричекая фигура описывается случайным набором точек и не имеет какой-либо структуры, что характерно для растровых изображений.

В общем, применение GC 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.

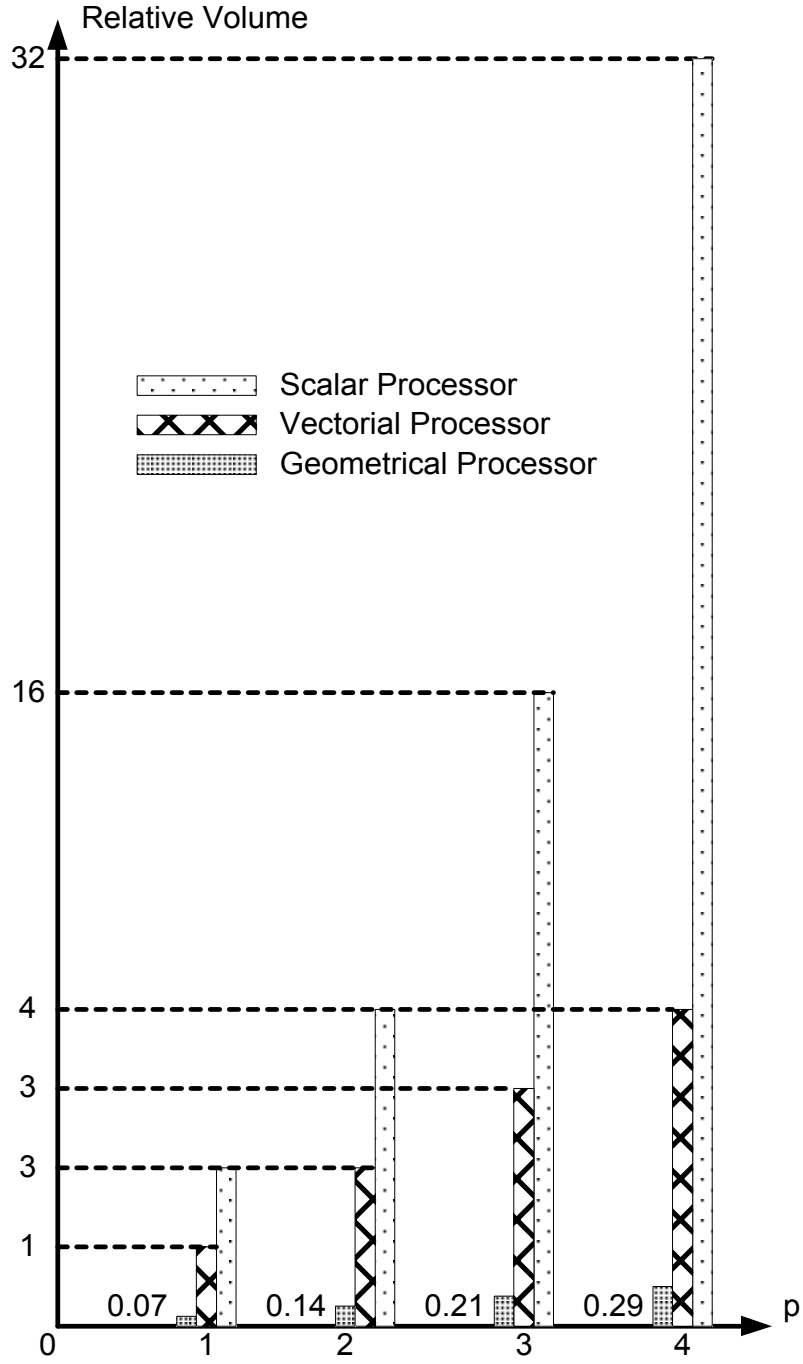

Рис. 1.1. Гистограмма относительного объема арифметических устройств.

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.

Книга содержит 7 глав, включая данное введение.

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 фигурам.

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 ревномерной 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_a, 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

$$a = p\,n + p^2\,r. \qquad (2.1.1)$$

В памяти процессора для каждой точки хранятся пары *«координаты-атрибут»*. Очевидно, максимальное количество кодируемых точек

$$M = 2^{pn}. \qquad (2.1.2)$$

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.

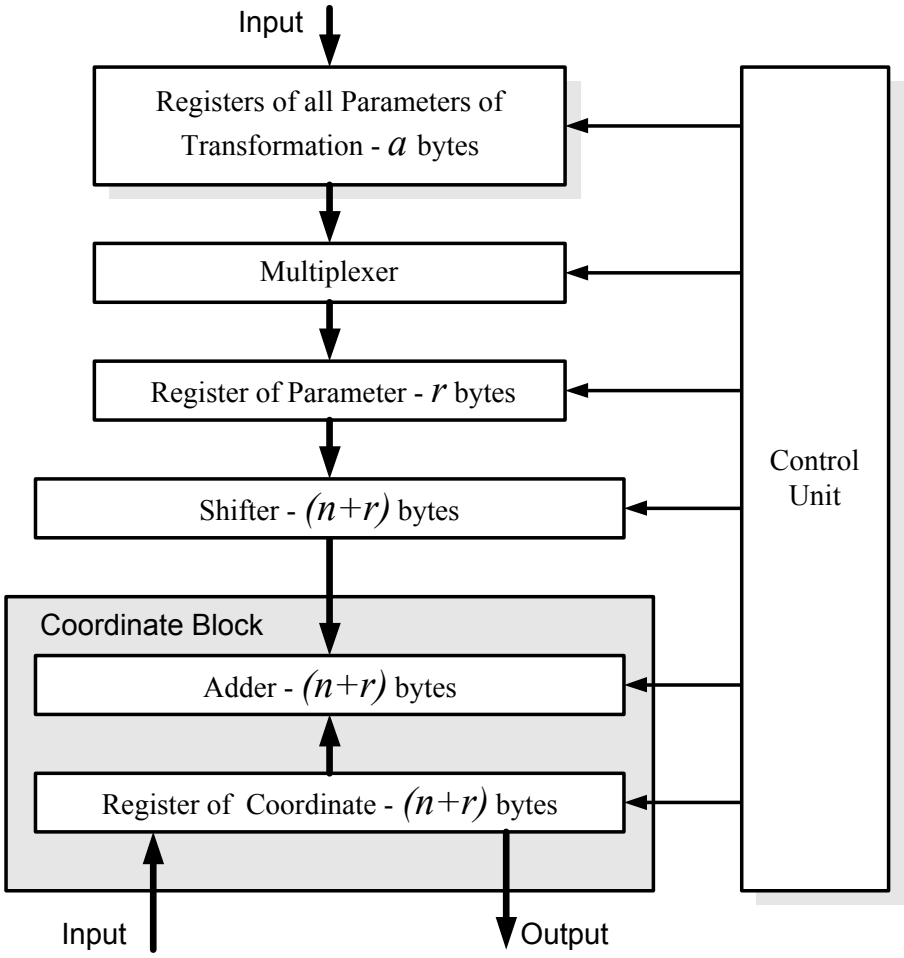

Рис. 2.2.1. Простейшее арифметическое устройство.

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

$$D = p(p-1) = 2,\ 6,\ 12,...\ if\ \ p = 2,\ 3,\ 4,... \quad (2.2.1)$$

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*.

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.

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.

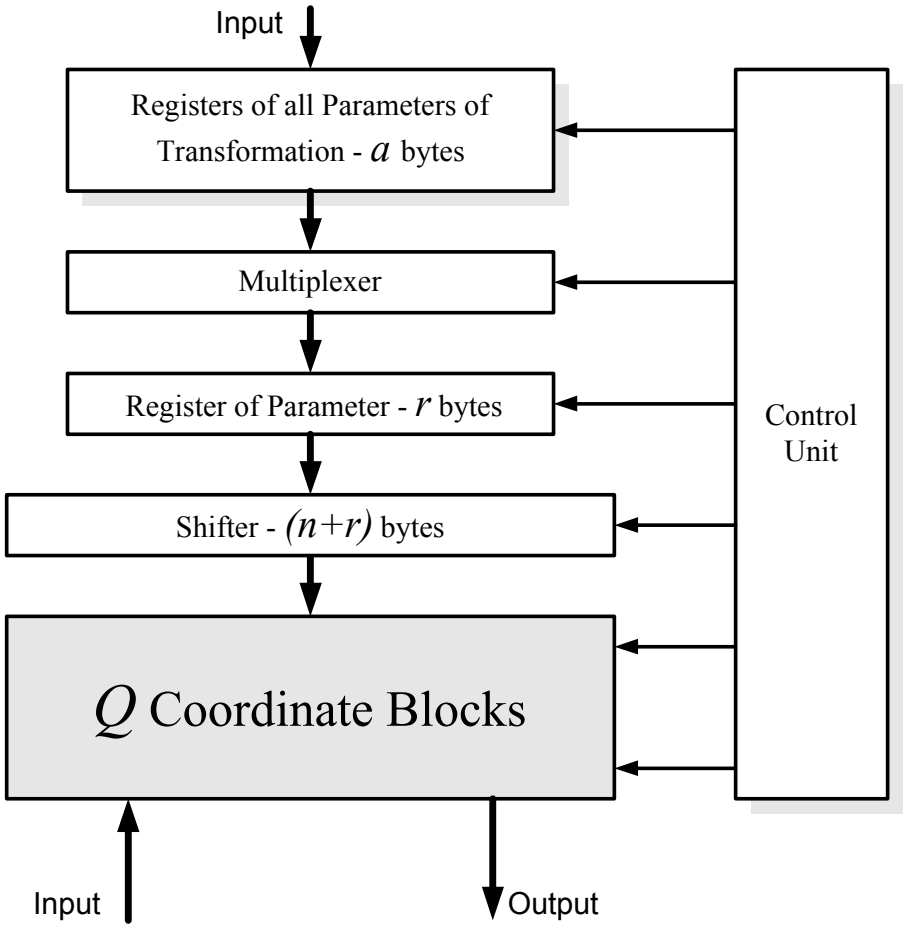

Рис. 2.2.2. Арифметическое устройство с фрагментарными прямоугольными кодами.

FSAU выполняет групповые операции с координатами точек фрагмента фигуры. На нем аффинное преобразование фигуры содержит Qp^2 групповых умножений и QD групповых сложений.

3. Основы компьютерной арифметики комплексных чисел и векторов

3.1. Метод кодирования комплексных чисел

В качестве основания кодирования линейных кодов могут использоваться комплексные числа. Такой код соответствует тому, что комплексное число Z представляется разложением вида

$$Z = \sum_m \alpha_m f\left(\rho,\ m\right),$$

где m –номер разряда разложения,

$\alpha_m = \left\{0,\ 1\right\}$ - значение разряда разложения,

ρ - основание разложения,

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

Двоичный позиционный код комплексного числа Z, соответствующий этому разложению, имеет вид

$$K(Z) = \ldots\alpha_m\ldots$$

В табл. 3.1.1 перечислены все возможные базовые функции, описанные в [11, 12]. Приведем еще для иллюстрации двоичные коды чисел во всех указанных системах кодирования, включая системы кодирования по действительному (положительному и отрицательному) и комплексному основаниям - см. табл. 3.1.2.

Таблица 3.1.1. Системы кодирования комплексных чисел

№	Basic Function	Base
1	$f(\rho,\ m)=\begin{cases}\rho^{m/2} \text{ if } m-\text{even}\\ j\cdot\rho^{m-1/2} \text{ if } m-\text{odd}\end{cases}$	$\rho=-2$
2	$f(\rho,\ m)=\rho^m$	$\rho = j\sqrt{2}$
3	$f(\rho,\ m)=\rho^m$	$\rho = -j\sqrt{2}$
4	$f(\rho,\ m)=\rho^m$	$\rho=(-1+j)$
5	$f(\rho,\ m)=\rho^m$	$\rho=(-1-j)$
6	$f(\rho,\ m)=\rho^m$	$\rho = \frac{1}{2}(-1+j\sqrt{7})$

Таблица 3.1.2. Двоичные системы кодирования.

	ρ	2	ρ	$\overline{\rho}$	- 2	- 1
1	$\begin{cases}(-2)^{m/2} \text{ if } m-\text{even}\\ j(-2)^{m-1/2} \text{ if } m-\text{odd}\end{cases}$	10100	10	1010	100	101
2	$j\sqrt{2}$	10100	10	1010	100	101
3	$-j\sqrt{2}$	10100	10	1010	100	101
4	$-1+j$	1100	10	110	11100	11101
5	$-1-j$	1100	10	110	11100	11101
6	$\frac{1}{2}(-1+j\sqrt{7})$	1010	10	101	110	111
7	**-2**	110	10		10	11
8	**2**	10	10			

3.2. Специальная алгебра в векторном пространстве

3.2.1. Алгебра в трехмерном векторном пространстве

Рассмотрим некоторую алгебру в трехмерном векторном пространстве. Пусть ***i, j, k*** – база трехмерного векторного пространства. Определим для этой базы табл. 3.2.1 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 3.2.1. Умножение трехмерных векторов

*	**i**	**j**	**k**
i	*i*	*j*	*k*
j	*j*	*k*	*-i*
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. 3.2.1 для ортов ***i, j, 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. 3.2.1 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.

3.2.2. 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, $U_1 * [U_2, U_2, U_2] = U_1 * U_2$.

3.2.3. Векторное произведение

$$U_1 \times U_2 = U_1 * [(-jz_2 + ky_2), (-jx_2 - kz_2), (jy_2 - kx_2)].$$

3.2.4. 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 $i(U_1 \bullet U_2)$.

3.2.5. 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

$$r_o = iCos\alpha + jCos\beta + kCos\gamma,$$

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 $0 \le \varphi \le \pi$, 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

$$U_2 = \begin{Bmatrix} i\left(Cos\varphi \cdot Sin^2\alpha + Cos^2\alpha\right) + \\ j\left(Cos\gamma \cdot Sin\varphi + Cos\alpha \cdot Cos\beta \cdot (1 - Cos\varphi)\right) + \\ k\left(-Cos\beta \cdot Sin\varphi + Cos\alpha \cdot Cos\gamma \cdot (1 - Cos\varphi)\right) \end{Bmatrix}$$

$$U_3 = \begin{Bmatrix} i\left(Cos\varphi \cdot Sin^2\beta + Cos^2\beta\right) + \\ j\left(Cos\alpha \cdot Sin\varphi + Cos\beta \cdot Cos\lambda \cdot (1 - Cos\varphi)\right) + \\ k\left(Cos\lambda \cdot Sin\varphi - Cos\alpha \cdot Cos\beta \cdot (1 - Cos\varphi)\right) \end{Bmatrix}$$

$$U_4 = \begin{Bmatrix} i\left(Cos\varphi \cdot Sin^2\gamma + Cos^2\gamma\right) + \\ j\left(-Cos\beta \cdot Sin\varphi - Cos\alpha \cdot Cos\lambda \cdot (1 - Cos\varphi)\right) + \\ k\left(Cos\alpha \cdot Cos\varphi - Cos\beta \cdot Cos\lambda \cdot (1 - Cos\varphi)\right) \end{Bmatrix}$$

3.2.6. Центроаффинное преобразование

Центроаффинное преобразование эквивалентно покомпонентному умножению на тройку векторов, построенных из элементов матрицы центроаффинного преобразования:

$$U_2 = U_1^T \cdot \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} == U_1 * \begin{bmatrix} (ia_{11} - ja_{12} - ka_{13}), \\ (ia_{21} - ja_{22} - ka_{23}), \\ (ia_{311} - ja_{32} - ka_{33}) \end{bmatrix}. \quad (3.2.1)$$

3.2.7. Многомерное пространство

Обобщим теперь полученные результаты на *n*-мерное пространство. Выберем в нем базу $E_1, E_2, ..., E_n$, элементы которой удовлетворяют условию

$$E_a E_b = -E_c \text{ при } d>n+1,$$
$$E_a E_b = E_c \text{ при } d<n+2,$$

где $d=(a+b-1)$, $c = d \bmod n$. (3.2.2)

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. 3.2.2 умножения комплексных чисел; при *n*=3 получаем алгебру с табл. 3.2.1, описанную выше; при *n*=4 последней формуле соответствует табл. 3.2.3 умножения четырех ортов и так далее.

Таблица 3.2.2. Умножение комплексных чисел.

*	**1**	**j**
1	*1*	*j*
j	*j*	*-1*

Таблица 3.2.3. 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*

3.3. Два способа синтеза кодов многомерных векторов

Как показано выше, в качестве основания кодирования линейных кодов могут использоваться комплексные числа - см. табл. 3.1.1. В системах 1 и 2 метод основан на построении некоторой композиции кодов действительных чисел по отрицательному основанию. Такой 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 ρ = - 2, то есть

$$X_i = \sum_{(m)} \alpha_m^i \rho^m, \qquad (3.3.1)$$

где *(i=1, 2,..., n)*. Каждому такому разложению соответствует код

$$K(X_i) = \ldots \alpha_m^i \ldots \qquad (3.3.2)$$

3.3.1. Способ 1.

Рассмотрим теперь *n*-мерный вектор

$$Z = E_1 X_1 + E_2 X_2 + \ldots + E_n X_n, \qquad (3.3.3)$$

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

$$K(Z) = \ldots \sigma_m \ldots, \qquad (3.3.4)$$

соответствующий разложению

$$Z = \sum_{(m)} r_m \rho^m,$$

где вектор

$$r_m = E_1 \alpha_m^1 + E_2 \alpha_m^2 + \ldots + E_i \alpha_m^i + \ldots + E_n \alpha_m^n \qquad (3.3.5)$$

изображается цифрой σ_m. В частности, при n=2 образуются коды комплексных чисел по основанию '-2', рассмотренные выше. При n=3 образуются коды трехмерных векторов, в которых разряды принимают одно из восьми значений:

$$r_m \in \{ 0, \mathbf{i, j, k, i+j, i+k, j+k, i+j+k} \}, \qquad (3.3.6)$$

где **i, j, k** - орты прямоугольных координатных осей.

Аналогично кодированию комплексных чисел для кодирования трехмерных векторов может быть введена векторная функция действительного целого аргумента

$$\vartheta_2(m) = \begin{Bmatrix} i(-2)^m \ \text{if} \ m = 3k \\ j(-2)^{m-1} \ \text{if} \ m = 3k+1 \\ k(-2)^{m-2} \ \text{if} \ m = 3k+2 \end{Bmatrix},$$

обозначаемая далее как ϑ_2^m. При этом рассматриваемый код трехмерного вектора по основанию (-2) с векторными значениями разрядов может рассматриваться как код трехмерного вектора по основанию (ϑ_2) с двоичными разрядами. Этому коду соответствует разложение вектора в виде $Z = \sum_m \left(\alpha_m \vartheta_2^m \right)$.

Аналогично, для кодирования n-мерных векторов может быть введена векторная функция действительного целого аргумента

$$\vartheta_n(m) = \begin{Bmatrix} i(-2)^m \ \text{if} \ m = nk \\ j(-2)^{m-1} \ \text{if} \ m = nk+1 \\ \dots \\ k(-2)^{m-n+1} \ \text{if} \ m = nk+n-1 \end{Bmatrix}, \qquad (3.3.7)$$

обозначаемая далее как ϑ_n^m.

3.3.2. Способ 2.

Построим теперь, как и для комплексных чисел, последовательность чередующихся двоичных разрядов α_m^i:

$$\dots \alpha_{m+1}^{2} \alpha_{m+1}^{1} \alpha_{m}^{n} \alpha_{m}^{n-1} \dots \alpha_{m}^{2} \alpha_{m}^{1} \alpha_{m-1}^{n} \alpha_{m-1}^{n-1} \dots \qquad (3.3.8)$$

В других обозначениях эта последовательность является двоичным кодом

$$K(Z) = \ldots \alpha_k \ldots, \qquad (3.3.9)$$

некоторого вектора Z. При этом основание кодирования также является вектором

$$\rho = E_2 \sqrt[n]{2}, \qquad (3.3.10)$$

где E_2 - второй орт базы $\{E_i\}$ n-мерного векторного пространства. Закодированный вектор Z определяется в данном случае по формуле

$$Z = X_1 + \rho \quad X_2 + \ldots + \rho^{\,i-1} X_i + \ldots + \rho^{\,n-1} X_n. \qquad (3.3.11)$$

С позиционными кодами векторов выполнимы операции алгебраического сложения, векторного, скалярного и покомпонентного умножения. Алгоритмы этих операций содержат циклы алгебраического сложения кодов чисел и сдвига кода вектора, то есть легко реализуются технически. Это может быть использовано при построении процессоров, оперирующих с векторами в целом. Такой процессор требует более простого алгоритма для решения задач с векторами, 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.

Для построения геометрических кодов наилучшими являются системы кодирования комплексных чисел 1, 2, 3. Последние две системы во многом аналогичны. Потому далее для комплексных чисел рассматриваются алгоритмы и устройства для операций только в системах 1 и 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». Такие коды образуются из традиционных кодов по основанию «2». Поэтому предварительно будут рассмотрены алгоритмы и устройства для арифметических операций, кодирования и декодирования действительных чисел по различным основаниям. Итак, далее будет рассмотрено несколько типов двоичных кодов:

- Традиционный прямой код по основанию «2» - так называемый **Р-код**,
- Код действительных чисел по основанию «-2» - так называемый **М-код**,
- Комплексный код по комплексному основанию - так называемый **С-код**.

Пределы изменения положительного действительного целого числа, представленного *n*-разрядным **Р-**кодом, таковы:

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

Пределы изменения действительного целого числа, представленного *n*-разрядным **М-**кодом, таковы:

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

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

3.4. Алгебраическое сложение М-кодов

Для построения геометрических кодов наилучшими являются системы кодирования 1, 2. В этих системах комплексный код представляется некоторой композицией кодов действительного числа по основанию «-2». Поэтому ниже предварительно будут рассмотрены алгоритмы и устройства для арифметических операций с М-кодами.

3.4.1. Многоразрядные схемы для М-кодов

Далее рассматриваются устройства для кодирования и декодирования. Основными блоками этих устройств являются линейные многоразрядные схемы алгебраического сложения. Они состоят из последовательно соединенных одноразрядных схем – см. рис. 3.4.1, где

N - разрядность кодов,

k={0,1,2,…,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.

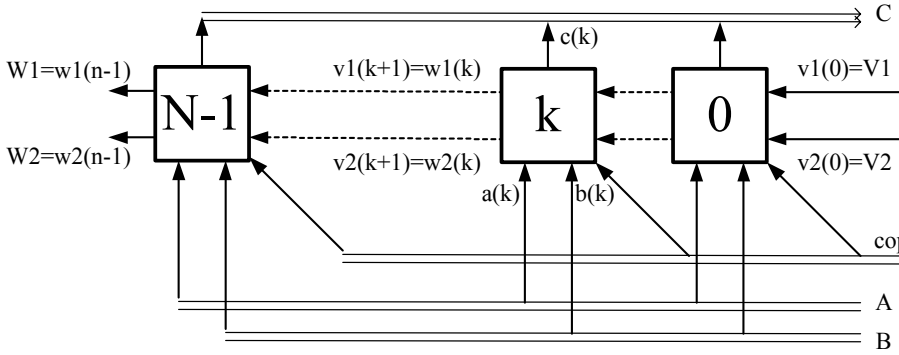

Рис. 3.4.1. Многоразрядная схема алгебраического сложения

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.
Ниже при рассмотрении конкретных схем алгебраического сложения описываются (как правило) только одноразрядные схемы.

3.4.2. Инвертор М-кода

На рис. 3.4.2 представлена одноразрядная схема инвертирования *Inv*. Ее функционирование описывается таблицей истинности табл. 3.4.2. Эта таблица вычисляет величину $c-2*w=-a+v$.

Таблица 3.4.2. Одноразрядная схема инвертирования

a	v	w	c
0	0	0	0
1	0	1	1
0	1	0	1
1	1	0	0

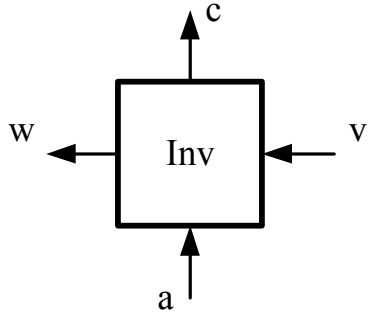

Рис. 3.4.2. Одноразрядная схема инвертирования.

3.4.3. Инверсный сумматор М-кодов

На рис. 3.4.3 представлена одноразрядная схема инверсного сумматора InvAdd. Ее функционирование описывается таблицей истинности табл. 3.4.3. Эта таблица вычисляет сумму *c-2*w=(-a–b+v).*

Таблица 3.4.3. Одноразрядная схема инверсного суммирования

a	b	v	w	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

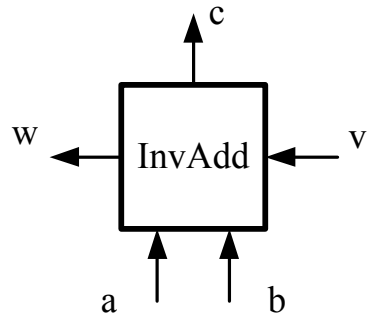

Рис. 3.4.3. Одноразрядная схема инверсного сумматора.

3.4.4. Сумматор М-кодов

На рис. 3.4.4 представлена одноразрядная схема сумматора **Add**. Ее функционирование описывается таблицей истинности табл. 3.4.4. Эта таблица вычисляет сумму $c\text{-}2^* w = (a + b + v)$

Таблица 3.4.4. Одноразрядная схема 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

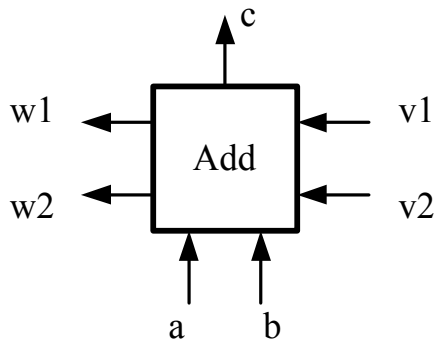

Рис. 3.4.4. Одноразрядная схема сумматора.

3.4.5. Вычитатель М-кодов

На рис. 3.4.5 представлена одноразрядная схема вычитателя **Sub**. Ее функционирование описывается таблицей истинности табл. 3.4.5. Эта таблица вычисляет сумму *c-2* w = (a - b + v).*

Таблица 3.4.5. Одноразрядная схема 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

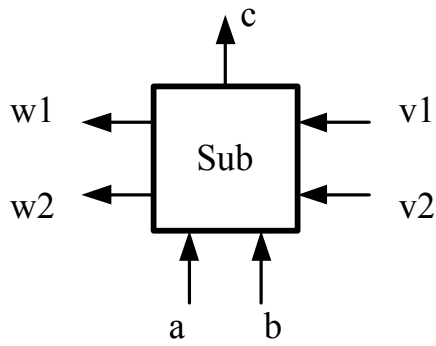

Рис. 3.4.5. Одноразрядная схема вычитателя.

3.4.6. Знакоопределитель М-кодов

Знакоопределитель определяет знак числа, представленного М-кодом. В нем используются одноразрядные знакоопределители. представленые на рис. 3.4.6.1 и имеющие две модификации:

Seven – одноразрядная схема знакоопределителя для разряда с четным номером,

Sodd - одноразрядная схема знакоопределителя для разряда с нечетным номером,

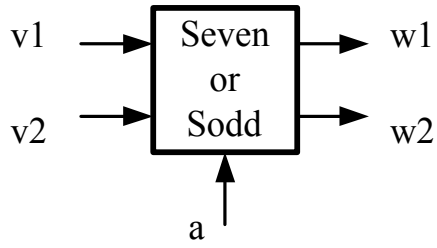

Рис. 3.4.6.1. Одноразрядная схема знакоопределителя.

Коды переносов в этих схемах интерпретируется следующим образом:

00 – код имеет нулевое значение,

01 – код имеет положительное значение,

10 – код имеет отрицательное значение.

Функционирование одноразрядных знакоопределителей *Seven* и *Sodd* описывается табл. 3.4.6.1 и 3.4.6.2 соответственно.

Таблица 3.4.6.1. Одноразрядная схема знакоопределителя для четного разряда

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**

Табл. 3.4.6.1 реализует правило:

‘w2, w1’ = ‘v2 v1’, если a = ‘0’,

‘w2, w1’ = ‘01’, если a = ‘1’.

Табл. 3.4.6.2 реализует правило:

‘w2, w1’ = ‘v2 v1’, если a = ‘0’,

‘w2, w1’ = ‘11’, если a = ‘1’.

Таблица 3.4.6.2. Одноразрядная схема знакоопределителя для нечетного разряда

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**

Знакоопределитель **nSign** в целом представлен на рис. 3.4.6.2, где показана схема соединения одноразрадных 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>=0.

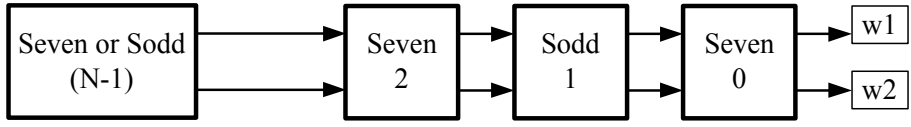

Рис. 3.4.6.2. Знакоопределитель

3.5. Умножение многомерных векторов

3.5.1. Метод умножения многомерных и векторов

Необходимо найти произведение векторов *C=A*B*, где множитель и множимое имеют соответственно разложения

$$A = \sum_h \alpha_h f(\rho\ ,h), \qquad B = \sum_k \beta_k f(\rho\ ,k).$$

Код произведения определяется как $C = \sum_h \left[B\alpha_h f(\rho\ ,h)\right]$.

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.

3.5.2. 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.

3.5.3. 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:

$$B_i = E_i B, \ i = \overline{1,n} \qquad (3.5.1)$$

- см. также (3.3.3). Например, при *n=2*, имеем:

$$B_1 = B, \ B_2 = jB;$$

при *n=3*, имеем:

$$B_1 = iB, \ B_2 = jB, \ B_3 = kB;$$

при *n=3*, имеем:

$$B_1 = iB, \ B_2 = jB, \ B_3 = kB, \ B_4 = mB$$

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)

$$B = E_1 b_1 + E_2 b_2 + \ldots + E_n b_n, \qquad (3.5.2)$$

В частности, для комплексных чисел используется табл. 3.2.2. Имеем:

$$B_1 = B, \ B_2 = j(b_1 + jb_2) = -b_2 + jb_1.$$

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.

3.5.4. 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.

3.5.5. 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.n. 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.

3.5.6. 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:

$B = X$, *если* $m = 3k$,
$B = Y$, *если* $m = 3k+1$.

Если выполняется *покомпонентное умножение трехмерного вектора*

C = A * (X, Y, V),

то множимое *B* определяется следующим образом:

$B = X$, *если* $m = 3k$,
$B = Y$, *если* $m = 3k+1$,
$B = V$, *если* $m = 3k+2$.

Если выполняется *покомпонентное умножение четырехмерного вектора*

C = A * (X, Y, V, W),

то множимое *B* определяется следующим образом:

$B = X$, *если* $m = 3k$,
$B = Y$, *если* $m = 3k+1$,
$B = V$, *если* $m = 3k+2$,
$B = W$, *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.

3.6. 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 (3.3.7), предложенные в [13].

В этой системе код вектора имеет вид (3.3.4), а его разряды принимают значения (3.3.6). 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:

$v_m = \{\alpha', \beta', \gamma'\}$, $w_m = \{\alpha'', \beta'', \gamma''\}$.

3.6.1. Скалярное произведение

По формуле для скалярного произведения

$$v \bullet w = \alpha'\alpha'' + \beta'\beta'' + \gamma'\gamma'' \qquad (3.6.1)$$

строится табл. 3.6.1, в которой для произведений - чисел указаны коды по основанию ρ = -2.

Таблица 3.6.1. 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 ρ = -2.

3.6.2. Векторное произведение

Формула векторного произведения имеет вид: $Z = V \times W$, причем $z_m = \{\alpha, \beta, \gamma\}$, где

$$\begin{aligned} \alpha &= \beta'\gamma'' - \gamma'\beta'', \\ \beta &= \gamma'\alpha'' - \alpha'\gamma'', \\ \gamma &= \alpha'\beta'' - \beta'\alpha''. \end{aligned} \qquad (3.6.2)$$

Координаты вектора ***Z,*** вычисленные по формуле (3.6.2), могут принимать значения -1, 0, 1. Следовательно, вектор ***Z*** всегда можно представить как разность двух векторов, каждый из которых имеет код по основанию (3.3.7). Используя, далее, правила алгебраического сложения этих векторов, можно построить табл. 3.6.2, описывающую векторное умножение. В отличие от предыдущей таблицы эта таблица должна быть заполнена полностью, так как является несимметричной относительно диагонали (векторное произведение некоммутативно).

Таблица 3.6.2. Одноразрядное векторное умножение.

×	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 $\boldsymbol{Z=V\times W}$ 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 $\boldsymbol{Z}$ по основанию (3.3.7).

3.6.3. 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 неодноименных частей друг на друга. Рассмотрим этот вопрос подробнее сначала для скалярного умножения.

Скалярное умножение описывается формулой (3.6.1), но в случае многоразрядного кода необходимо еще учитывать перенос p из младшего разряда. При этом разрядный результат должен вычисляться по формуле

$$S=\alpha'\alpha''+\beta'\beta''+\gamma'\gamma''+p. \qquad (3.6.3)$$

В величина S должна быть представлена в виде

$$S=\sigma+P\rho, \qquad (3.6.4)$$

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. 3.6.3.

Таблица 3.6.3. 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

$$S = \alpha'\alpha'' + \beta'\beta'' + \gamma'\gamma'' + p + q\,. \qquad (3.6.5)$$

и представляться в виде

$$S = \sigma + P\rho + Q\rho^{2}\,. \qquad (3.6.6)$$

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. 3.6.4.

Таблица 3.6.4. 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**

На рис. 3.6.1 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 – входной перенос,

Sum – одноразрядный сумматор.

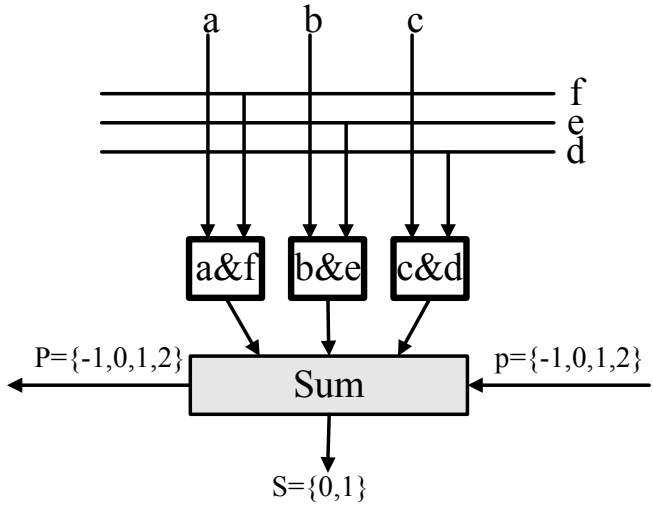

Рис. 3.6.1. Сумматор в блоке скалярного умножения

3.6.4. Переносы при векторном умножении.

Векторное умножение описывается формулами (3.6.2). С учетом переносов она приобретает следующий вид:

$$\begin{aligned} \rho P_\alpha + \alpha &= p_\alpha + \beta'\gamma'' - \gamma'\beta'', \\ \rho P_\beta + \beta &= p_\beta + \gamma'\alpha'' - \alpha'\gamma'', \\ \rho P_\gamma + \gamma &= p_\gamma + \alpha'\beta'' - \beta'\alpha''. \end{aligned} \qquad (3.6.7)$$

где *p, P* - значения переносов в данный и в старший разряды. В формуле (3.6.7) переносы принимают только два значения (0,1). Рассмотрим первую из этих формул. Для нее операция описывается табл. 3.6.5.

Таблица 3.6.5. 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. 3.6.2 приведена схема сумматора в блоке векторного умножения на один разряд множителя. На этой схеме

a, b, c – разряды кода множимого,
d, e, f – разрад кода множителя,
pG, pH, pM – входные переносы,
PG, PH, PM – выходные переносы,
SumG, SumH, SumM – одноразрядные сумматоры.

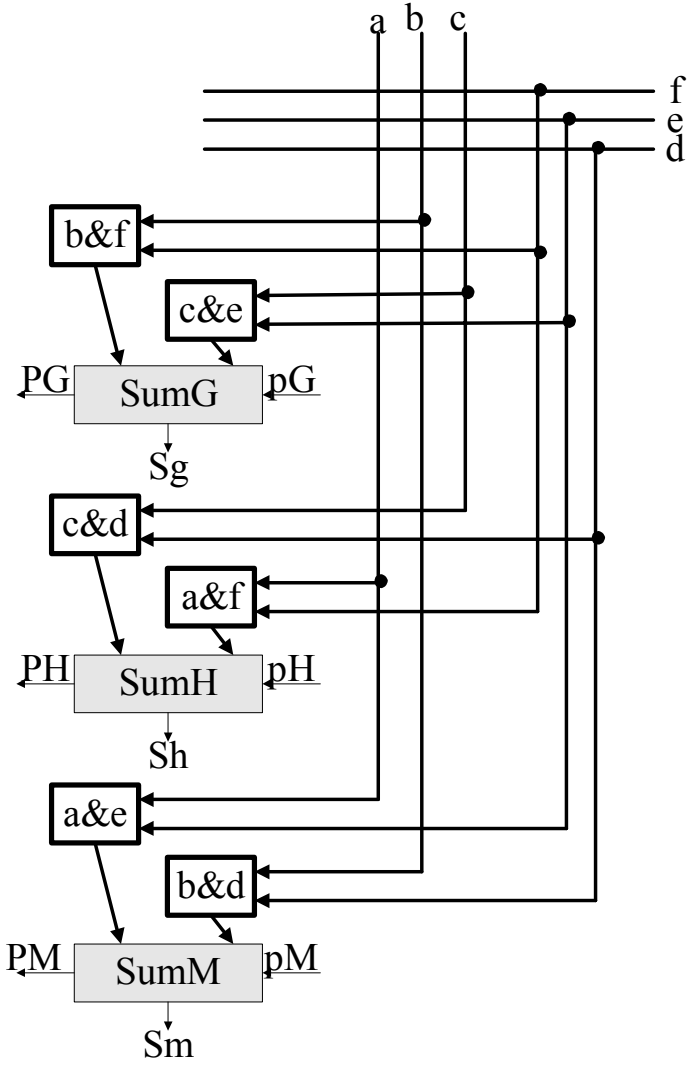

Рис. 3.6.2. Сумматор в блоке векторного умножения

3.7. Алгоритмы и устройства для кодирования и декодирования комплексных чисел и векторов

Для построения геометрических кодов наилучшими являются системы кодирования 1, 2. Последние две системы во многом аналогичны. Потому далее рассматриваются алгоритмы и устройства для кодирования и декодирования только в системах 1 и 2. В этих системах комплексный код представляется некоторой композицией кодов действительного числа по основанию «-2». Такие коды образуются из традиционных кодов по основанию «2». Поэтому предварительно будут рассмотрены алгоритмы и устройства для кодирования и декодирования действительных чисел по различным основаниям.

Алгоритмы и устройства для кодирования и декодирования для векторов не рассматриваются, т.к. они полностью аналогичны алгоритмам и устройствам для 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.

3.7.1. 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*. Затем следует на *инверторе М-кодов* вычислить числа $|X_\alpha| \cdot sign(X_\alpha)$, $|X_\beta| \cdot sign(X_\beta)$.

2. Формирование С-кода $K(Z) = \ldots\beta_m\alpha_m\ldots\beta_1\alpha_1\beta_0\alpha_0,\beta_{-1}\alpha_{-1}\beta_{-2}\alpha_{-2}\ldots$ комплексного числа $Z = X_\alpha + jX_\beta$, который в дальнейшем представляется в виде $K(Z) = \ldots\gamma_m\ldots$, где $\left\{\begin{matrix}\gamma_{2m} = \alpha_m \text{ if } m-\text{even}\\ \gamma_{2m+1} = \beta_m \text{ if } m-\text{odd}\end{matrix}\right\}$. Для этого используется *распределитель*.

3.7.2. Декодирование комплексного числа в системе 1.

1. Выделение из С-кода комплексного числа $Z = X_\alpha + jX_\beta$ четных и нечетных разрядов по правилу $\left\{\begin{matrix}\alpha_{m/2} = \gamma_m \text{ if } m-\text{even}\\ \beta_{(m-1)/2} = \gamma_m \text{ if } m-\text{odd}\end{matrix}\right\}$ и формирование из разрядов α_m и β_m М-кодов действительных чисел $X_\alpha,\ X_\beta$ соответственно. Для этого используется *прекодер*
2. Декодирование действительных чисел $X_\alpha,\ X_\beta$ (положительных или отрицательных) чисел $X_\alpha,\ X_\beta$ из М-кода в Р-код. Для этого используется *полный декодер М-кода в Р-код* .

3.7.3. Кодирование комплексного числа в системе 2.

1. Вычисление $\overline{X}_\beta = \mu\cdot X_\beta$ при $\mu = 1/\sqrt{2}$. Это вычисление выполняется в традиционной системе двоичного кодирования.
2. Кодирование действительных (положительных или отрицательных) чисел $X_\alpha,\ \overline{X}_\beta$ из Р-кода в М-код аналогично п.1 алгоритма 3.7.1.
3. Формирование С-кода комплексного числа $Z = X_\alpha + j\overline{X}_\beta$ аналогично п.2 алгоритма 3.7.1.

3.7.4. Декодирование комплексного числа в системе 2.

1. Выделение из С-кода комплексного числа $Z = X_\alpha + jX_\beta$ четных и нечетных разрядов по правилу $\left\{\begin{matrix}\alpha_{m/2} = \gamma_m \text{ if } m - \text{even} \\ \beta_{(m-1)/2} = \gamma_m \text{ if } m - \text{odd}\end{matrix}\right\}$ и формирование из разрядов α_m и β_m М-кодов действительных чисел $X_\alpha,\ \overline{X}_\beta$ соответственно, где $\overline{X}_\beta = \mu \cdot X_\beta$ при $\mu = 1/\sqrt{2}$. Для этого используется *прекодер*.
2. Декодирование действительных (положительных или отрицательных) чисел $X_\alpha,\ \overline{X}_\beta$ из из М-кода в Р-код аналогично п.2 алгоритма 3.7.2.
3. Вычисление $X_\beta = \overline{X}_\beta\sqrt{2}$. Это вычисление выполняется в традиционной системе двоичного кодирования.

3.7.5. 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. 3.7.5.1, 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.

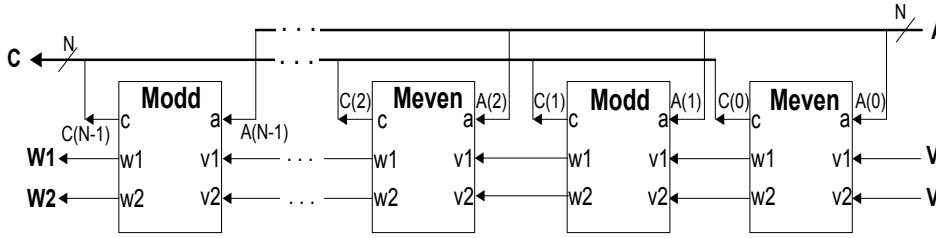

Рис. 3.7.5.1. Кодер положительного Р-кода в М-код

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. 3.7.5.2. Их функционирование описывается таблицей истинности табл. 3.7.5.1 и 3.7.5.2 соответственно.

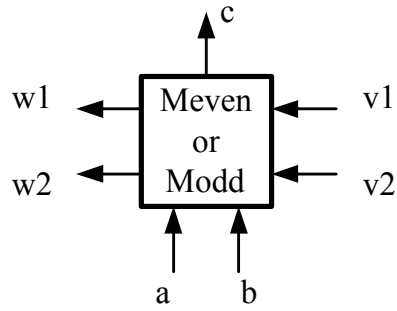

Рис. 3.7.5.2. Одноразрядная схема кодера.

Таблица 3.7.5.1. Одноразрядная схема кодера для четного разряда

Input			Output		
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 3.7.5.2. Одноразрядная схема кодера для нечетного разряда

Input			Output		
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

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

Этот декодер преобразует М-код некоторого числа в Р-код этого числа. Его схема представлена на рис. 3.7.6.1, 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.

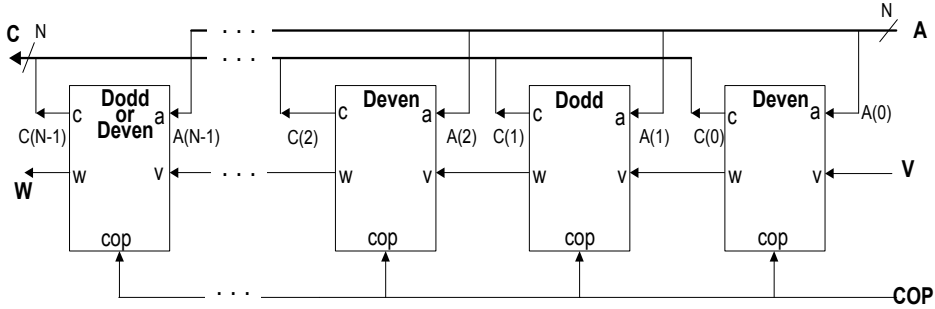

Рис. 3.7.6.1. Декодер М-кода в Р-код

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 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 cop=1), а вычитание выполняется по правилам вычитания Р-кодов.

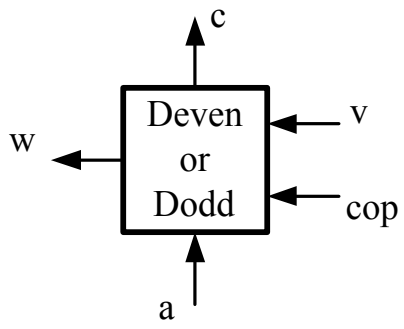

Рис. 3.7.6.2. Одноразрядная схема декодера.

Одноразрядные схемы *Deven* и *Dodd* представлены на рис. 3.7.6.2. Их функционирование описывается таблицей истинности табл. 3.7.6.1 и 3.7.6.2 соответственно.

Табл. 3.7.6.1 вычисляет

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

Табл. 3.7.6.2 вычисляет

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

Таблица 3.7.6.1. Одноразрядная схема декодера для четного разряда

Input				**Output**	
	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

Таблица 3.7.6.2. Одноразрядная схема декодера для нечетного разряда

Input				Output	
	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

3.7.7. Полный декодер М-кода в Р-код – mDecoderMP

Декодер *DecoderMP* управляется кодом операции *Cop*. В связи с этим полный декодер должен (кроме декодера *DecoderMP*) содержать еще и знакоопределитель *nSign*. При этом схема блока принимает вид, представленный на рис. 3.7.7.

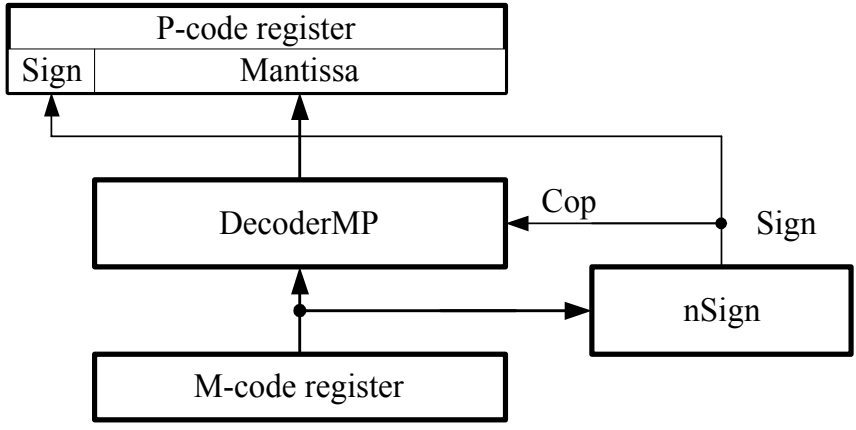

Рис. 3.7.7. Полный декодер.

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

Прекодер Р-кода в М-код распределяет разряды Р-кода 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 коды A_{even} и A_{odd}. Заметим, что эти коды в дальнейшем поступают на два входа алгебраического сумматора М-кодов. Этот сумматор вычисляет либо $\left(A_{even} - A_{odd}\right)$, либо $\left(A_{odd} - A_{even}\right)$ в зависимости от значения некоторого управляющего сигнала $s = \{0,1\}$ соответственно. Прекодер представлен в табл. 3.7.8, где для кода указаны номера разрядов, а для кодов A_{even} и A_{odd} знаком «**=**» указаны разряды, значения которых совпадают со значением соответствующего разряда кода . Цифрой «**0**» указано определенное значение разряда этих кодов.

Таблица 3.7.8. Прекодер Р-кода в М-код

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

3.7.9. Распределитель частей кода – Partitioning.

Распределитель частей кода преобразует действительную и мнимую части комплексного кода *A* в два М-кода действительных чисел Re *A* и Im *A*. При этом мнимая часть передается со сдвигом на 1 разряд влево. Распределитель кода представлен в табл. 3.7.9, где для кода *A* указаны номера разрядов, а для кодов Re *A* и Im *A* указаны номера разрядов кода *A*, которые перемещаются распределителем в данный разряд.

Таблица 3.7.9. Распределитель частей кода

A	…	7	6	5	4	3	2	1	0
Re *A*	…	14	12	10	8	6	4	2	0
Im *A*	…	15	13	11	9	7	5	3	1

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 кодов 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

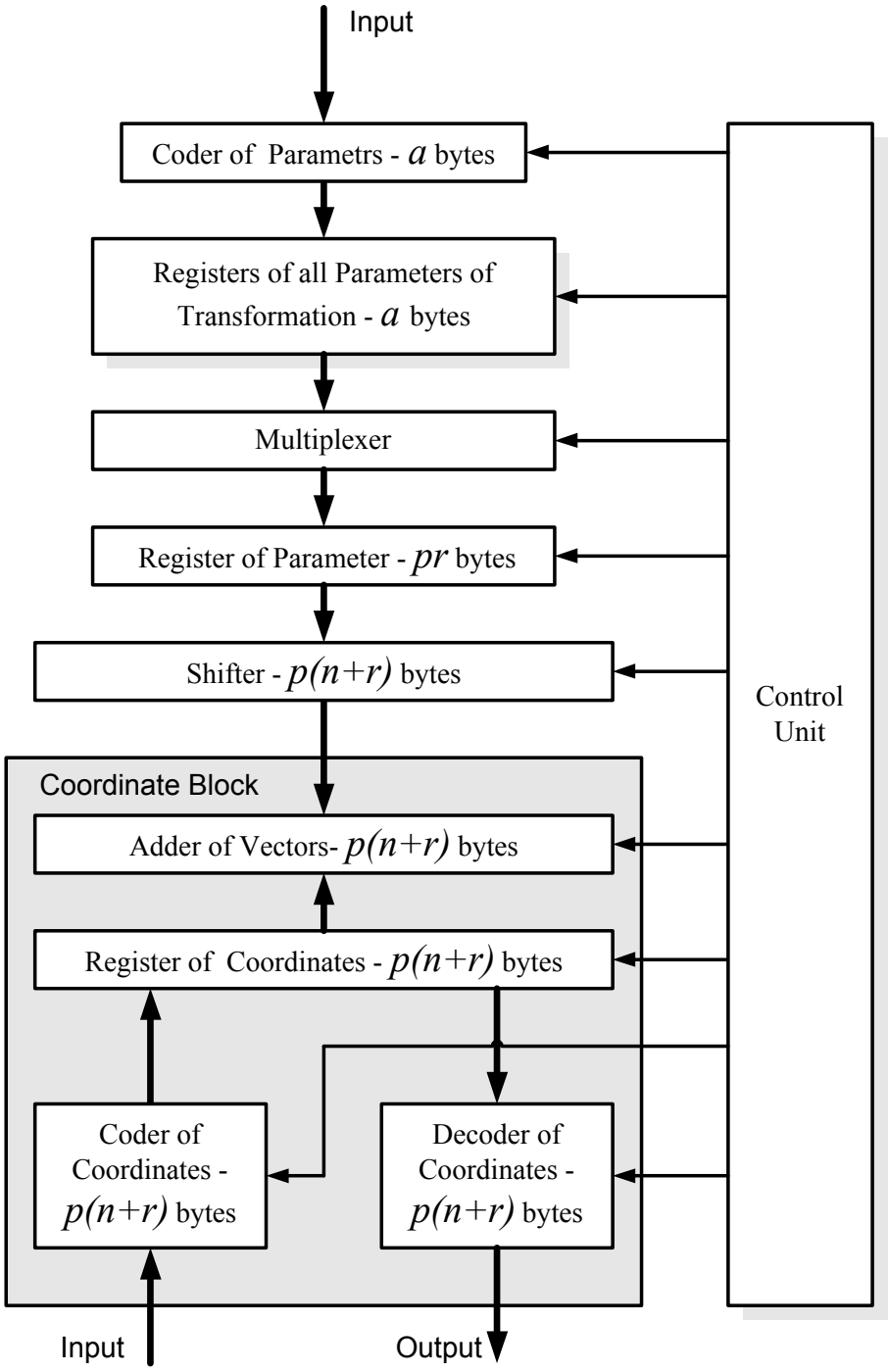

Рис. 4.1.1. Векторное арифметическое устройство

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.

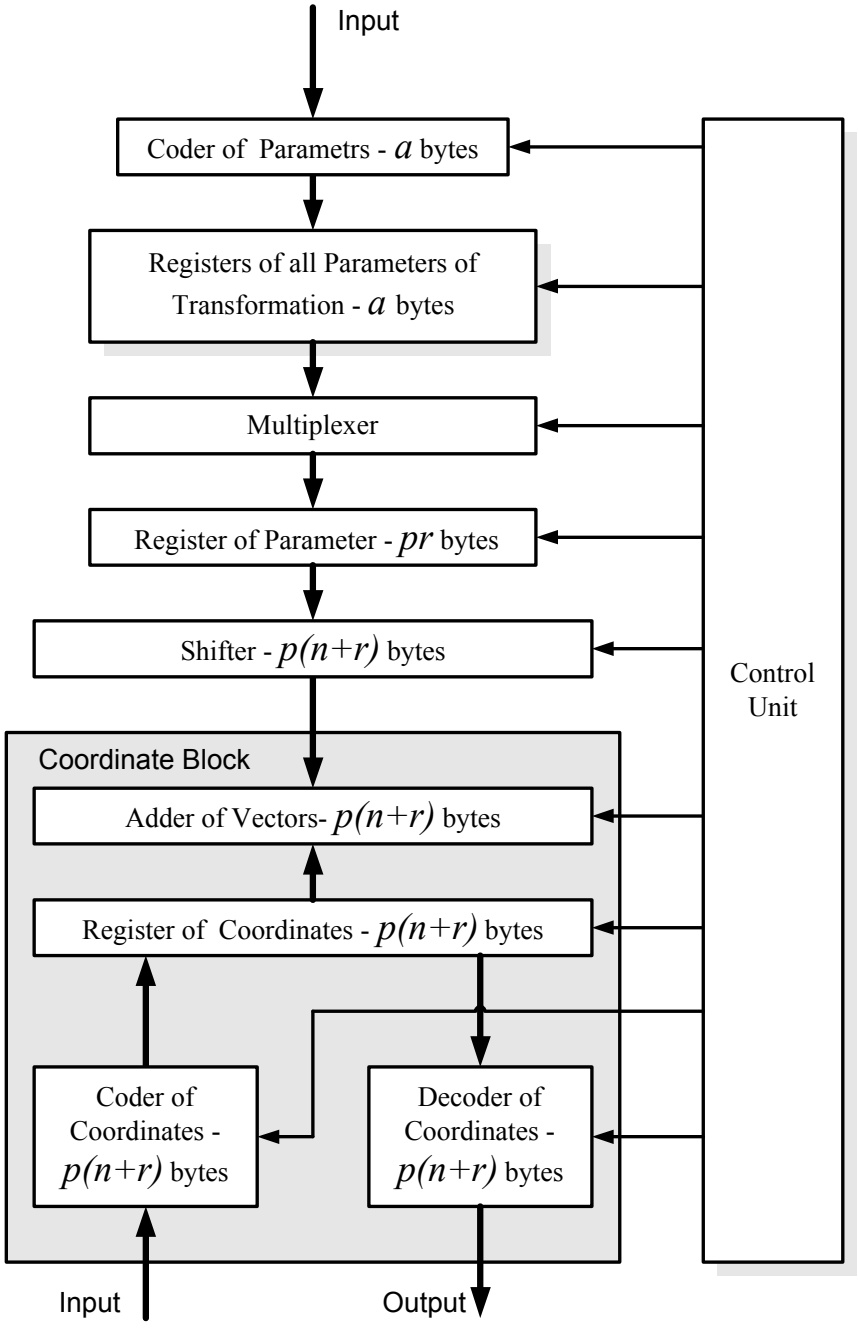

Рис. 4.1.2. Векторное арифметическое устройство с прямоугольными кодами.

4.2. Сравнения

В табл. 4.2.1a и 4.2.1b 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 4.2.1a. 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 4.2.1b. 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 пр 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

$$h_k = H_k \Big/ H_{k+1},$$

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 $Q >> 1$. Относительное качество h возрастает при $Q \to 1$. 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 значеие 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.

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.

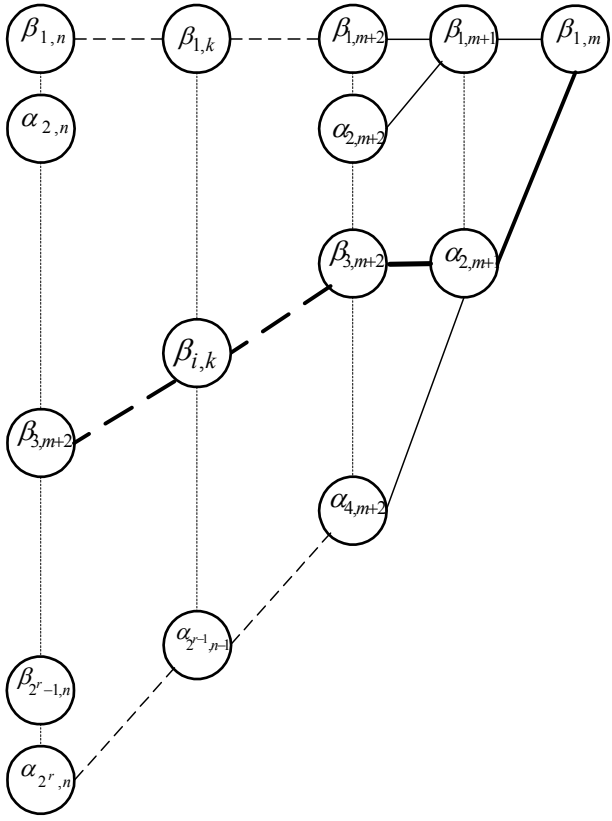

Рис. 5.1.1. Дерево геометрического кода.

Пусть m и n - номера крайнего правого и левого ярусов соответственно. Тогда $k=(n, n\text{-}1,..., m\text{+}1, m)$; $i=(1, 2,..., 2^{k-m})$; число ярусов $r=(n\text{-}m\text{+}1)$; число узлов дерева $u=(2^r\text{-}1)$; 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

$$\beta_{p,n}\ldots\beta_{i,k}\ldots\beta_{3,m+2}\alpha_{2,m+1}\beta_{1,m}.$$

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...0...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)

$$\alpha_{43}\alpha_{22}\beta_{11}\beta_{10} \qquad (K[4]=1100),$$
$$\beta_{53}\beta_{32}\alpha_{21}\beta_{10} \qquad (K[5]=0010),$$
$$\alpha_{63}\beta_{32}\alpha_{21}\beta_{10} \qquad (K[6]=1010),$$
$$\beta_{73}\alpha_{42}\alpha_{21}\beta_{10} \qquad (K[7]=0110),$$

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.

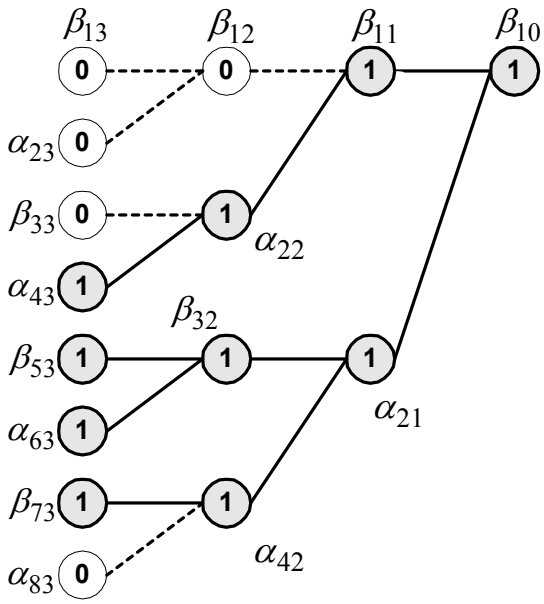

Рис. 5.1.2. Пример. Дерево бинарных разрядов

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 тоже число. Кроме того, геометрическим кодом удается (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.

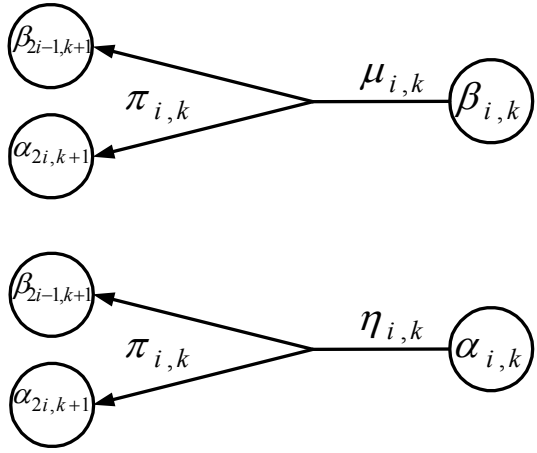

Рис. 5.1.2а. Схема распространения переносов в GC

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 ρ 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:

$$\mu = \pi \wedge \overline{\delta}, \qquad \eta = \pi \wedge \delta.$$

При μ=1 разряд β принимает значение “1” вне зависимости от его прежнего значения. Аналогично, при η=1 разряд α принимает значение “1”.

5.1.2.3. Транспонирование

*Транспонирование*м геометрического кода будем называть такое преобразование, при котором нижняя и верхняя половины геометрического кода меняются местами. Точнее говоря, разряды исходного кода связаны с разрядами траспонированного кода (помеченные верхней чертой) следующим образом:

$$\alpha_{i,k} = \overline{\alpha}_{j,k}, \quad \beta_{i,k} = \overline{\beta}_{j,k}, \quad j = rest(1 + 2^{k-i}) \bmod 2^{k-i+1}.$$

Например, код на рис. 5.1.2 транспонируется в код на рис. 5.1.3.

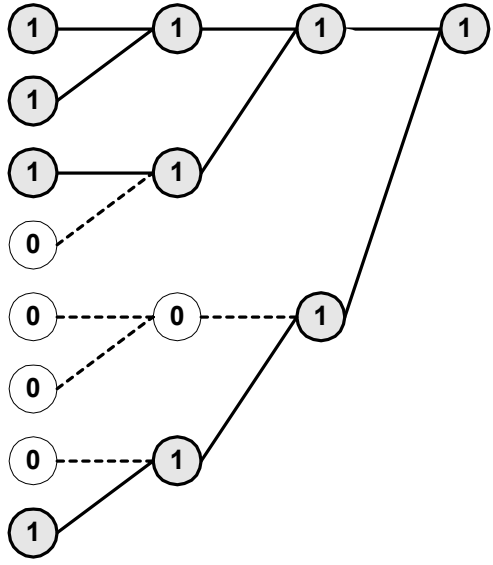

Рис. 5.1.3. Пример. Транспонированный код.

5.1.2.4. Сложение геометрического и базисного кодов при ρ=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:

$$\tau_{i,k} = \delta_{k+1} \oplus \pi_{i,k}, \qquad (5.1.1)$$

$$\eta_{i,k} = \alpha_{i,k} \wedge \delta_{k} \wedge \pi_{i-1,k-1} \quad \text{при } i \text{ - четном,} \qquad (5.1.2)$$

$$\mu_{i,k} = (\delta_{k} \vee \pi_{i,k-1}) \wedge \beta_{i,k} \quad \text{при } i \text{ - нечетном.} \qquad (5.1.3)$$

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 $\boldsymbol{K}$=<2> или $\boldsymbol{K}$=10, а геометрический код $\boldsymbol{G}$ изображает множество линейных кодов { 1100, 0010, 1010, 0110 } или, что одно и то же, множество чисел { 12, 2, 10, 6 }. Найдем геометрический код $\boldsymbol{R} = \boldsymbol{G} + \boldsymbol{K}$ – см. рис. 5.1.4. Процесс распространения переносов прекращается. Полученный код $\boldsymbol{R} = G_4$ 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 ρ=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 удается организовать цепи циклического переноса.

5.1.2.6. Алгебраическое сложение геометрического и базисного кодов при ρ=-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

$$\tau = \delta \quad \oplus \quad \pi, \quad \eta \ = \alpha \wedge (\delta \vee \overline{\pi}), \quad \mu \ = \beta \wedge \delta \wedge \overline{\pi}.$$

Таблица 5.1.1b. Обратное сложение GC с базисным кодом при ρ=-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 δ=0 описывают инвертирование геометрического кода. Здесь также можно воспользоваться формулами (5.1.1) и

$$\eta_{i,k} = \beta_{i,k} \wedge \delta_{k} \wedge \overline{\pi}_{i-1,k-1} \quad \text{при } i \text{ - четном,} \qquad (5.1.4)$$

$$\mu_{i,k} = (\delta_{k} \vee \overline{\pi}_{i,k-1}) \wedge \alpha_{i,k} \quad \text{при } i \text{ - нечетном.} \qquad (5.1.5)$$

Пример 5.1.2 обратного сложения при ρ =-2. Пусть базисный код $\boldsymbol{K} = < 2 >$ или $\boldsymbol{K} = 110$, а геометрический код $\boldsymbol{G}$ изображает множество линейных кодов {0000, 0100, 0010, 0110} или, что одно и то же, множество чисел { 0, 4, -2, 2 }. Найдем геометрический код $\boldsymbol{R} = - \boldsymbol{G} - \boldsymbol{K} -$ см. рис. 5.1.5.

Процесс распространения переносов прекращается. Полученный код $\boldsymbol{R} = G_4$ 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 ρ=-2 сводится к многократному транспонированию.

	$m=$	3 2 1 0	номер разряда
	$K=$	0 0 1 0	базисный код
1)	$G_1=$	1 1 1 1	$\pi_{10}=0$
		0	
		1 1	$\tau_{10}=\ \delta_1=1$
		0	
		1 1 1	
		0	
		1 1	
		0	
2)	$G_2=$	1 1 1 1	$\pi_{11}=\beta_{11}\wedge\delta_1\wedge\overline{\pi}_{10}=1$
		0	
		1 1	$\pi_{21}=\ \alpha_{21}\wedge(\delta_1\vee\overline{\pi}_{10})=1$
		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)	$G_3=$	1 1 1 1	$\pi_{12}=\ \pi_{32}=0$
		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)	$G_4=$	1 1 1 1	$\pi_{i,3}=0$
		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

$$G_{i,k} = G_{21} = \ldots \beta_{53} \beta_{32} 0$$
$$\alpha_{63}$$
$$\beta_{73} \alpha_{42}$$
$$\alpha_{83}$$

Для ликвидации переполнения разрядной сетки, которое может возникнуть при умножении, следует воспользоваться операцией округления, описанной ниже.

Данный способ умножения не применим при ρ =2, если среди чисел, представленных линейными кодами, имеются отрицательные числа.

Пример 5.1.3 умножения при ρ=-2. Найдем произведение базисного кода $\boldsymbol{K}$ = < -2 > или $\boldsymbol{K}$ = 10 и геометрического кода – см. рис. 5.1.6.

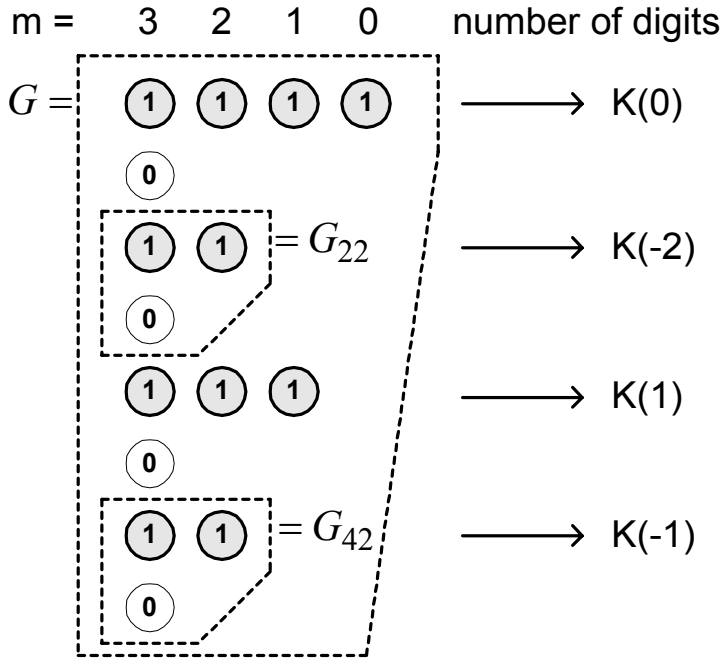

Рис. 5.1.6. К примеру 5.1.3.

В третьем ярусе кода *G* нет разрядов α = 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.

$G''_{11} =$
0 0 1 1
0
1 1
1
0 0 0
0
0 0
0

$G'' =$	→ K(…)
1 1 1 1	→ K(0)
1	→ K(4)
1 1	→ K(-2)
1	→ K(2)
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 $\left\{G_{i,k},\ i-\mathrm{var}\right\}$ 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 ρ=-2. Найдем произведение базисного кода $\boldsymbol{K}$=<-1> или $\boldsymbol{K}$=11 и геометрического кода – см. рис. 5.1.9a.

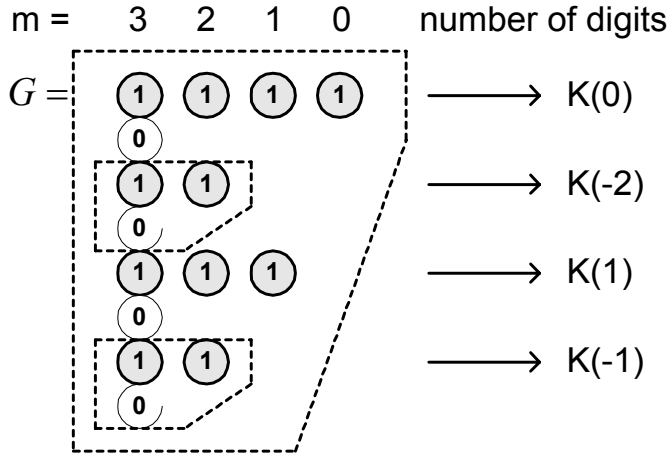

Рис. 5.1.9a. К примеру 5.1.3а.

В третьем ярусе G нет разрядов α=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. 5.1.9b.

G' =	→ K(…)
1 1 1 1	→ K(0)
0	
0 1	
1	→ K(2)
1 1 1	→ K(1)
0	
0 1	
1	→ K(3)

G'' =	→ K(…)
1 1 1 1	→ K(0)
0	
0 1	
1	→ K(2)
1 1 1	→ K(1)
0	
1 1	→ K(-1)
0	

Рис. 5.1.9b. 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. 5.1.9b. 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 умножение 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 *Jm* 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:

$$\tau_{i,k} = \delta_{k+1} \oplus \overline{\pi}_{(i+1)/2,k-1} \quad \text{при } i \text{ - четном,} \qquad (5.1.6)$$

$$\tau_{i,k} = \delta_{k+1} \oplus \overline{\pi}_{i/2,k-1} \quad \text{при } i \text{ - нечетном,} \qquad (5.1.7)$$

$$\eta_{i,k} = \beta_{i,k} \wedge \delta_{k} \wedge \overline{\pi}_{j,k-2} \quad \text{при } i \text{ - четном,} \qquad (5.1.8)$$

$$\mu_{i,k} = (\delta_{k} \vee \overline{\pi}_{j,k-2}) \wedge \alpha_{i,k} \quad \text{при } i \text{ - нечетном,} \qquad (5.1.9)$$

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 $\boldsymbol{K}$ = < $j\sqrt{2}$ > = 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	$\pi_{11} = \beta_{11} \wedge \delta_1 = 1$
		0	
		0 0	$\pi_{21} = \alpha_{21} \wedge \delta_1 = 1$
		0	
		1 1 1	$\tau_{11} = \delta_2 = 0$
		0	
		0 0	$\tau_{21} = \delta_2 = 0$
		0	
3)	$G_3 =$	1 1 1 1	$\pi_{12} = \pi_{22} = \pi_{32} = \pi_{42} = 0$
		0	
		0 0	
		0	
		1 1 1	$\tau_{12} = \tau_{22} = \tau_{32} = \tau_{42} = 1$
		0	
		0 0	
		0	
4)	$G_4 =$	0 1 1 1	$\pi_{i,3} = 0$
		1	
		0 0	$\tau_{i,3} = 0$
		0	
		0 1 1	
		1	
		0 0	
		0	

Рис. 5.1.10. К примеру 5.1.4.

Процесс распространения переносов прекращается. Полученный код $\boldsymbol{R} = G_4$ изображает коды 1010 и 1000 чисел (-

$j\sqrt{2}$) и (-2 $j\sqrt{2}$) 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 ***K*** = <1+ $j\sqrt{2}$ > или ***K*** = 11 и геометрический код ***G*** - см. рис. 5.1.11.

m = 3 2 1 0 -1	m = 3 2 1 0 -1
$\boldsymbol{G}$ = 1 1 1 1 1	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 $\boldsymbol{G}$ 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 $\boldsymbol{K}$, в результате чего получаем коды

$G'_{73} = G'_{153}$ 0 0 1
0
0 1
1

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 $\boldsymbol{K}$ и в результате получаем коды G''_{10} и G''_{20}. Налагаем, далее, эти коды на код G''_0 и получаем окончательно код $\boldsymbol{R}$ – см. рис. 5.1.13. Далее будет дана геометрическая интерпретация этого примера.

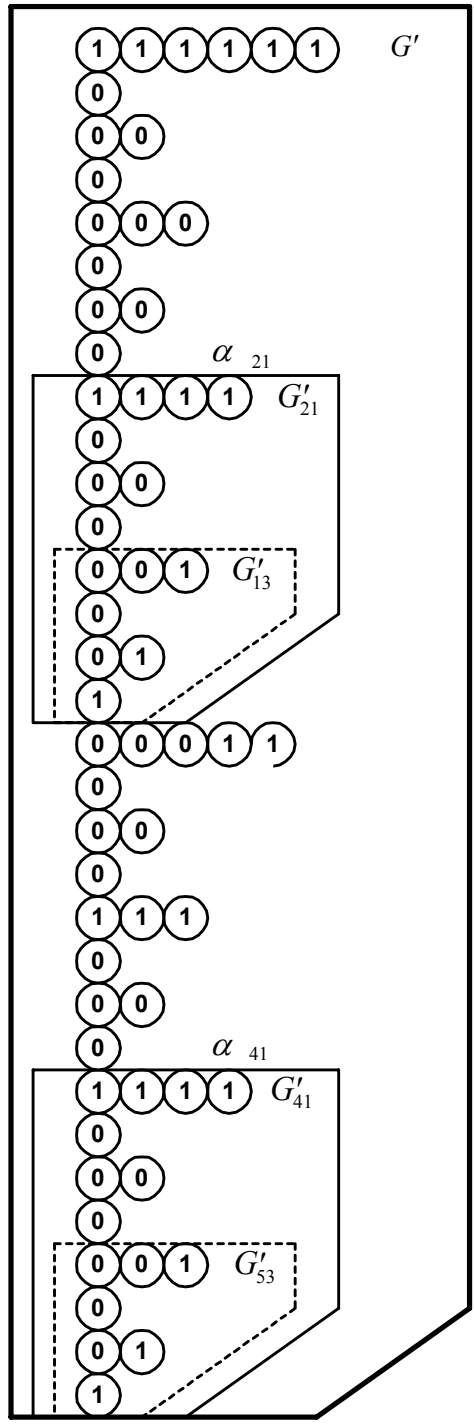

Рис. 5.1.12. К примеру 5.1.5.

$G_0''=$	$G_{10}''=$	$G_{20}''=$	$\boldsymbol{R}=$
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.

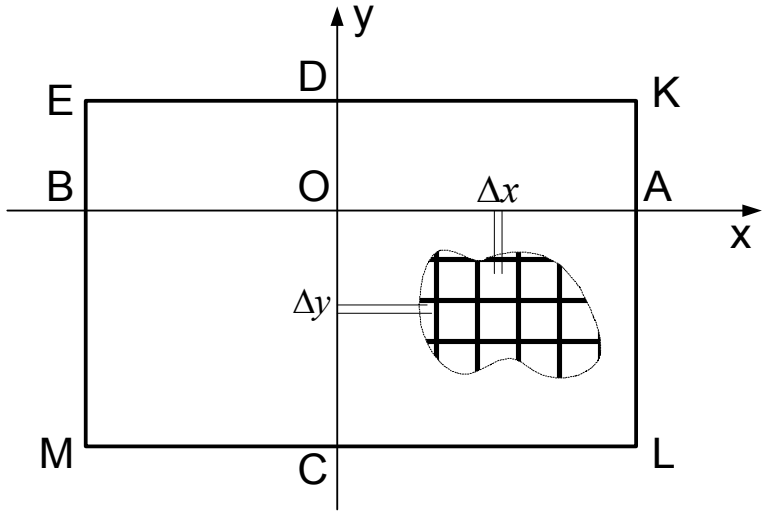

Рис. 5.1.14. Кодирование плоской фигуры.

Кодируемый геометрическим кодом участок плоскости имеет вид прямоугольника EKLM, стороны которого проходят через точки A, B, C, D перпендикулярно осям - см. рис. 5.1.14. В области EKLM 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.

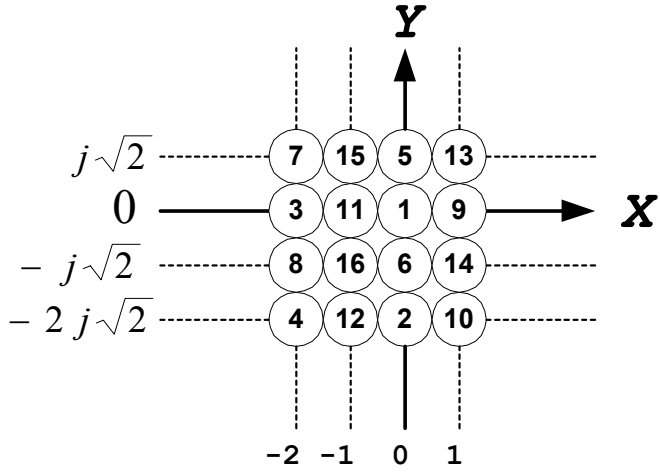

Рис. 5.1.15. Кодирование плоскости при y=3, m=-1, r=4 для примера 5.1.6.

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**

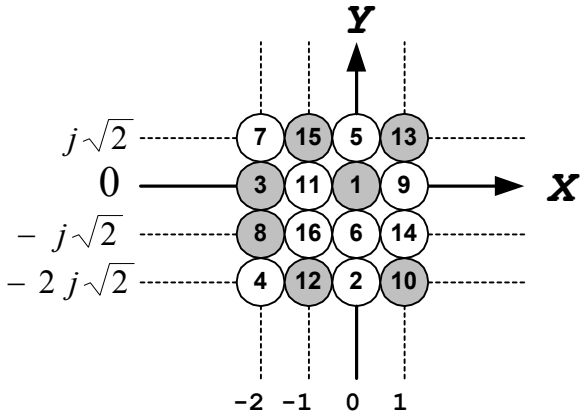

Рис 5.1.15a. Пример: плоская фигура.

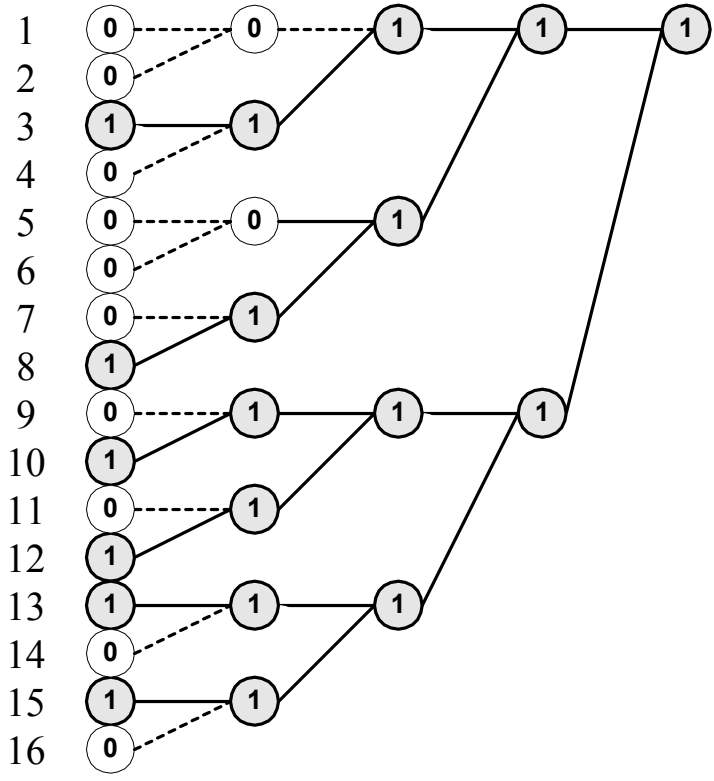

Рис 5.1.15b. Пример: дерево GC плоской фигуры.

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+jh) (покомпонентное умножение*). Эту операцию описывает формула $z+jv=x(c+jd)+jy(g+jh)$ - 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 геометрическрго 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 $\Psi = 55^0$ $(tg\Psi = \sqrt{2})$. Все коды чисел a_i изображаются единым геометрическим кодом ***G*** исходной фигуры.

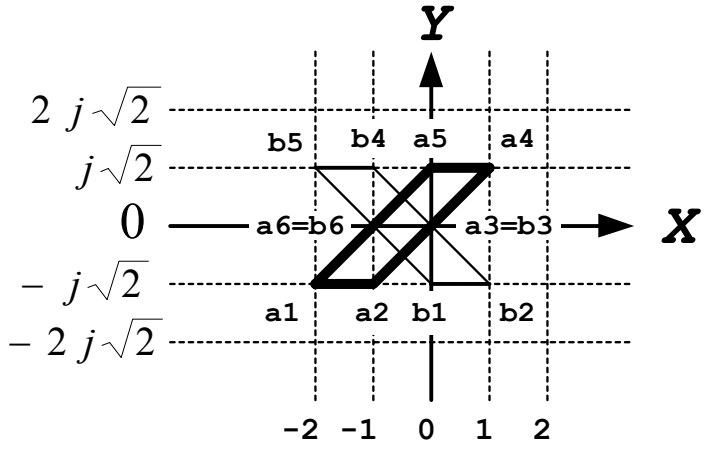

Рис. 5.1.16. Центроаффинное преобразование фигуры для примера 5.1.7

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 эквивалентна умножению мнимой части геометрического кода ***G*** на базисный код *K*=0011 числа $(1 + j\sqrt{2})$. Такое умножение выполнено в примере 5.1.5. В результате образуется геометрический код ***R***. Декодируя код ***R***, находим, что он объединяет линейные коды точек b_i дефрмированной 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 $\rho = j\sqrt[3]{2}$. 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 поменятся 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 прстранстве), *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 α=1, β=1, γ=(0,1):

$$\alpha_{p,n}\cdots\ \beta_{2j-1,q}\cdots\ \alpha_{2i,k}\cdots\ \beta_{1,m}$$

$$\gamma_{p/2,n}\cdots\ \gamma_{j,q}\cdots\ \gamma_{i,k},$$

Этот путь изображает линейный код

$$\bar{\gamma}_{p/2,n}\cdots\ \gamma_{j,q}\cdots\ \bar{\gamma}_{i,k},$$

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 ... 0 ... 1 ... 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 AGC каждый путь имеет номер, значение и атрибут.

На рис. 5.2.1 представлен AGC. Количество разрядов AGC определяется по следующей формуле:

$$V = 1 + 3\sum_{k=0}^{n-m} 2^k = 3\cdot 2^{n-m} - 2\cdot$$

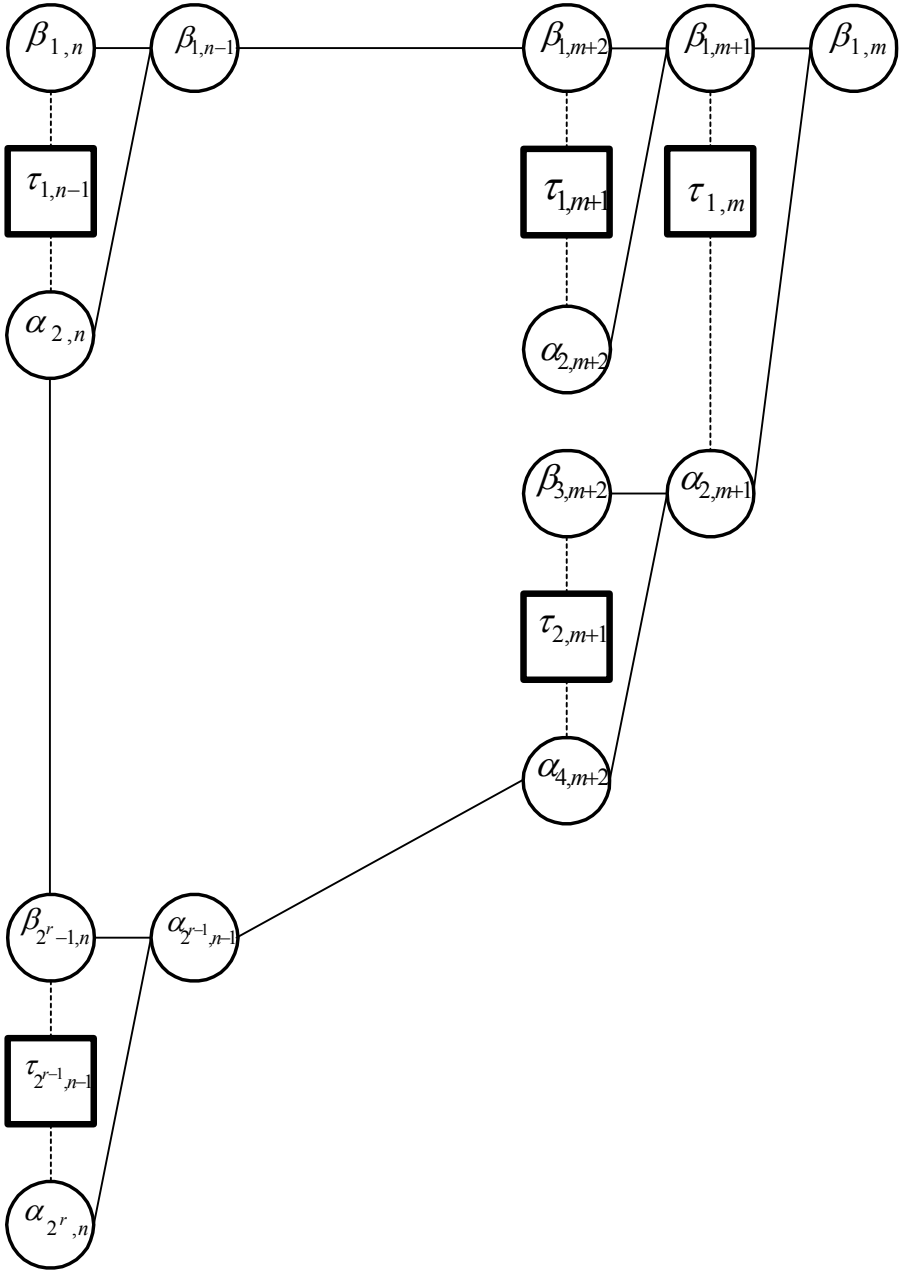

Рис. 5.2.1. Атрибутный геометрический код.

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:

$$\mu = \pi \wedge \overline{\delta}, \qquad \eta = \pi \wedge \delta.$$

При μ=1 разряд β принимает значение "1" вне зависимости от его прежнего значения. Аналогично, при η=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:

$$\mu = \pi \wedge \overline{(\delta \oplus \gamma)}, \qquad \eta = \pi \wedge (\delta \oplus \gamma).$$

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 μ=1 разряд β принимает значение "1" вне зависимости от его прежнего значения. Аналогично, при η=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 ($\beta\equiv1$ и $\alpha\equiv1$) 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,

$$\mu = \pi \wedge \bar{\delta}, \qquad \eta = \pi \wedge \delta,. \qquad \omega = (\mu \wedge \gamma) \vee (\eta \wedge \bar{\gamma}).$$

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 ρ=2

описывается табл. 5.2.3. Сигналы переносов μ, η и сигнал транспонирования τ вырабатываются, как функции от π, δ, γ. После этого сигнал τ=1 изменяет значение γ на противоположное, а сигналы μ и η распространяются дальше (если β=1 и α=1 соответственно).

Таблица 5.2.3. Сложение AGC с базисным кодом при ρ=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 ***ρ=2***. Пусть, как и в примере 5.1.1, базисный код ***K***=<2> или ***K***=10, а атрибутный геометрический код ***G*** изображает множество линейных кодов {1100, 0010, 1010, 0110 } или, что одно и то же, множество чисел {12, 2, 10, 6 }.

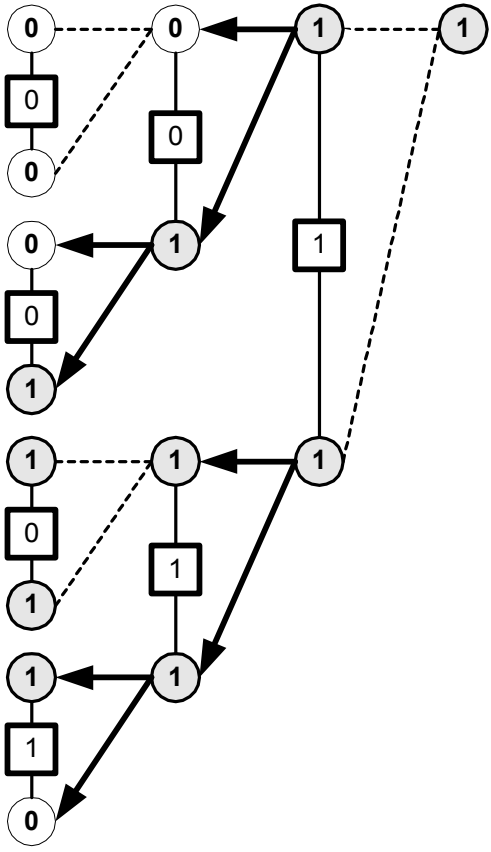

Рис. 5.2.2. К примеру 5.2.1.

Результирующий атрибутный геометрический код ***R=G+K.*** На рис. 5.2.2 иэображен код ***R***. Толстыми стрелками указаны связи, по которым распространялся перенос π=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 ρ=-2 описывается табл. 5.2.3а. Сигналы переносов μ, η и сигнал транспонирования τ вырабатываются, как функции от π, δ, γ. После этого сигнал τ=1 изменяет значение γ на противоположное, а сигналы μ и η распространяются дальше (если β=1 и α=1 соответственно).

Таблица 5.2.3а. Обратное сложение AGC с базисным кодом при ρ=-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 ρ=-2.
Пусть, как и в примере 5.1.2, базисный код ***K***=< 2> или ***K*** = 110, а атрибутный геометрический код ***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 ***R=-G-K.*** На рис. 5.2.3 иэображен код ***R***. Толстыми стрелками указаны связи, по которым распространялся перенос π=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 5.1.2. 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.

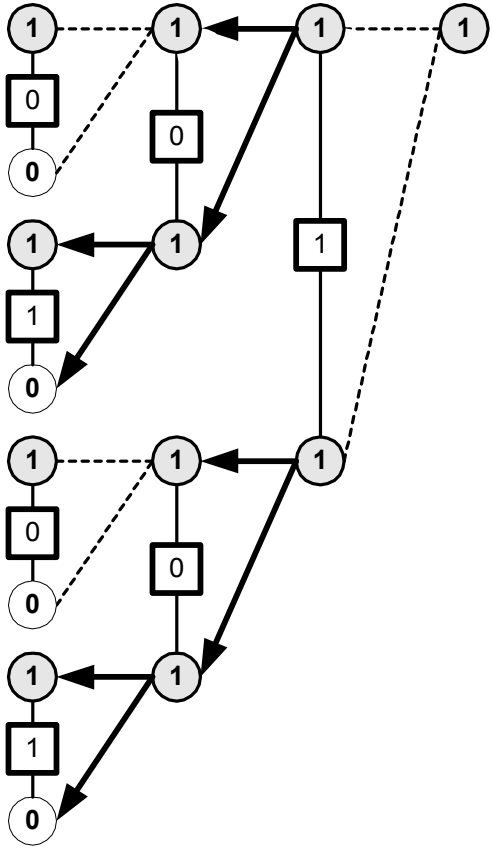

Рис. 5.2.3. К примеру 5.2.2.

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.

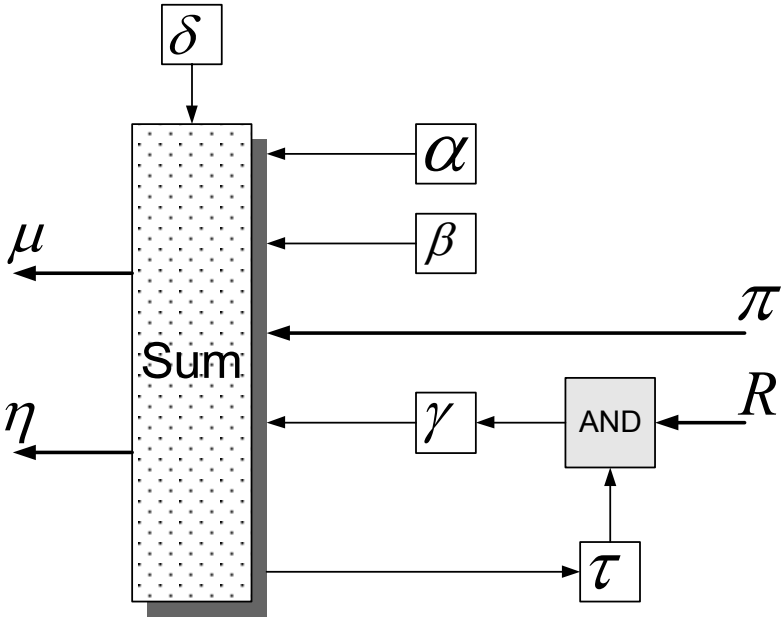

Рис. 5.2.3а. Одноразрядная схема обратного сложения

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 ρ=-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 ρ=-2.

π	γ	μ	η	τ
0 0	0 1	0 0	0 0	0 0
1 1	0 1	0 0	0 0	1 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.

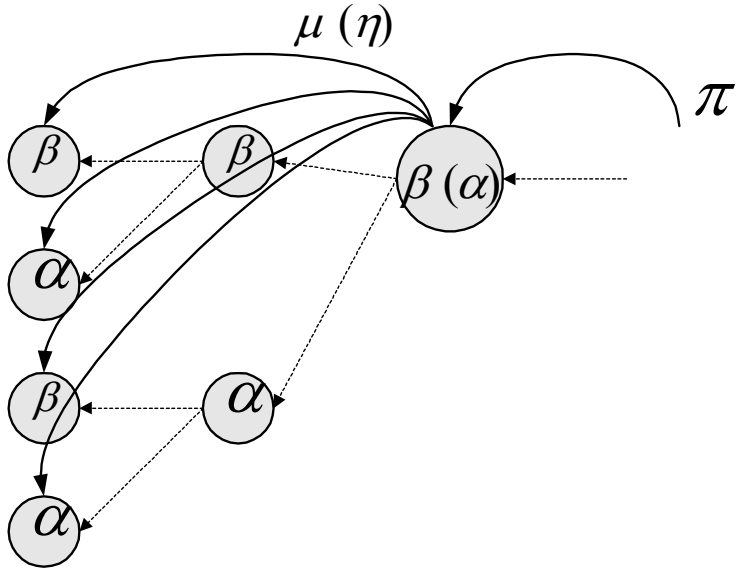

Рис. 5.2.4. Схема распространения переносов в комплексном GC

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. Центроаффинное преобразование AGCC.

Эта операция эквивалентна центроаффинному преобразованию фигуры и выполняется по формуле

$$S = \mathrm{Re}\, G \cdot Z' + \mathrm{Im}\, G \cdot Z'',$$

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 τ=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 τ=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.

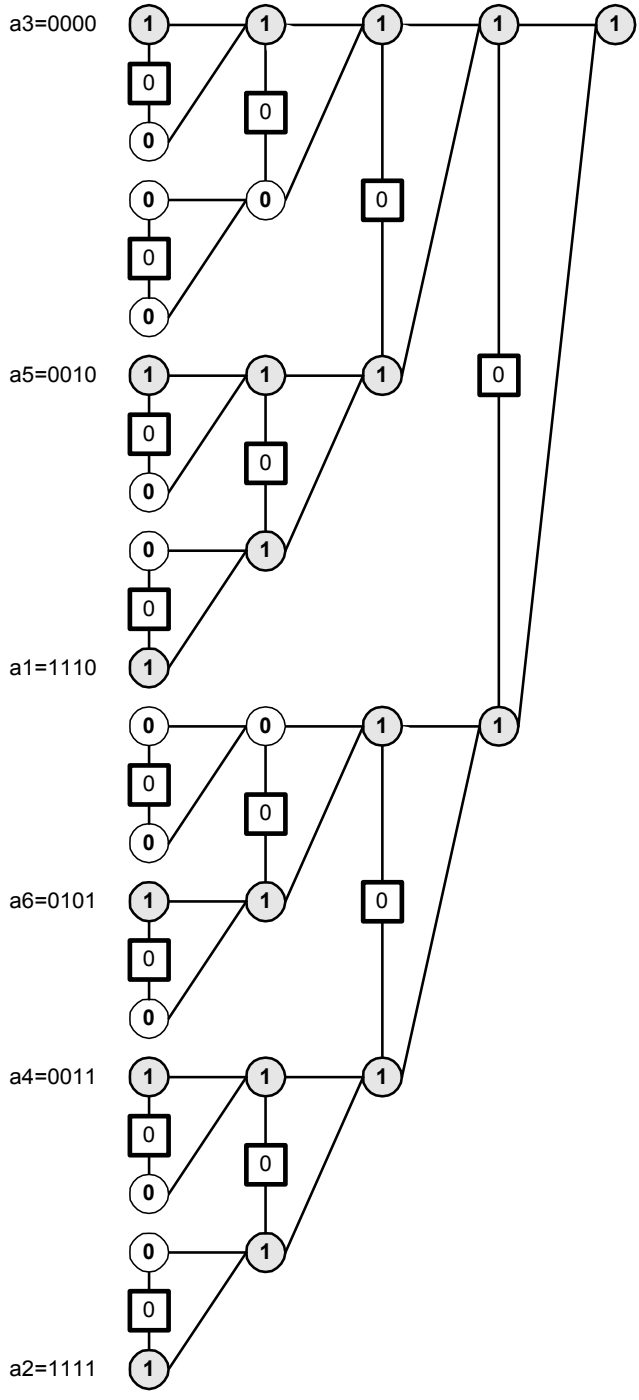

Рис. 5.2.5. К примеру 5.2.3: AGC исходной фигуры.

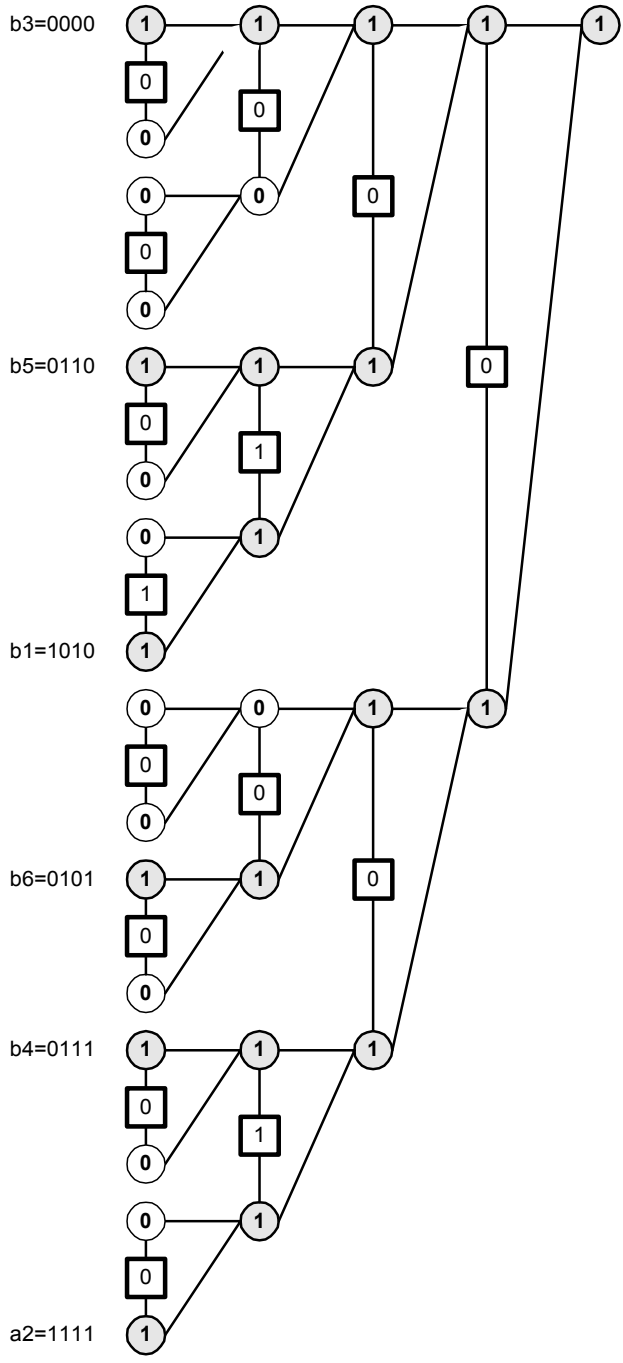

Рис. 5.2.6. К примеру 5.2.3: AGC деформированной фигуры

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 – четырмя 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

$$S = \mathrm{part1}(G) \cdot Z_1 + \mathrm{part2}(G) \cdot Z_2 + \mathrm{part3}(G) \cdot Z_3 + \mathrm{part4}(G) \cdot Z_4$$

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},

i, j, k, m – орты векторного пространства,

h >= 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

$$Z = Z_1, \quad \text{if} \quad r = 3\,h+1,$$
$$Z = Z_2, \quad \text{if} \quad r = 3\,h+2,$$
$$Z = Z_3, \quad \text{if} \quad r = 3\,h+3;$$

для четырехмерных векторов

$$Z = Z_1, \quad \text{if} \quad r = 4\,h+1,$$
$$Z = Z_2, \quad \text{if} \quad r = 4\,h+2,$$
$$Z = Z_3, \quad \text{if} \quad r = 4\,h+3,$$
$$Z = Z_4, \quad \text{if} \quad r = 4\,h+4.$$

При способе 1 кодирования векторов код замены Z выбирается следующим образом:

для трехмерных векторов

$$Z = Z_1, \quad \text{if} \quad r = 3\,h+1,$$
$$Z = j \cdot Z_2, \quad \text{if} \quad r = 3\,h+2,$$
$$Z = k \cdot Z_3, \quad \text{if} \quad r = 3\,h+3;$$

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 атрибутные геометрические коды

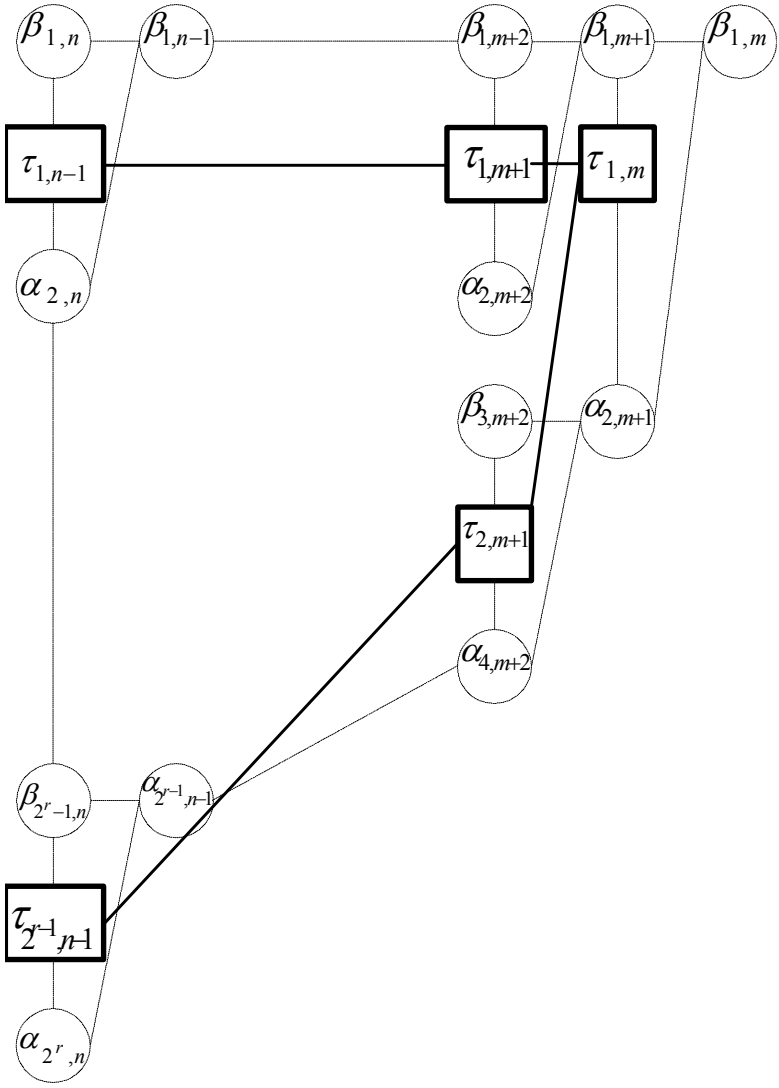

Рис. 5.2.7. Сокращенный AGC

Выше рассматривались AGC, в которых пути могли быть открытыми или закрытыми. При этом значения разрядов α=1 (или 0) и β=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 α=1 и все β=1. В таком случае нет необходимости реально включать эти разряды в геометрический код. Во всех манипуляциях с AGC можно преполагать, что все α=1 и все β=1. AGC без разрядов будем называть сокращенным AGC - **CAGC**.

На рис. 5.2.7 представлен сокращенный AGC. Разряды, реально отсутствующие, обозначены пунктиром. Количество разрядов сокращенного AGC определяется по следующей формуле:

$$V = \sum_{k=0}^{n-m} 2^k = 2^r - 1.$$

Каждой терминальной вершине $\tau_{k,n-1}$ дерева сокращенного AGC соответствует два атрибута – верхний, соответствующий мнимой вершине $\beta_{k,n}$, и нижний, соответствующий мнимой вершине $\alpha_{k+1,n}$.

Очевидно, в сокращенном AGC операционные схемы существенно сокращаются, а его объем сокращается в три раза. Точнее, если в обычном AGC количество разрядов $V = 3 \cdot 2^r - 2$, то в сокращенном GC количество разрядов $V = 2^r - 1$.

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 ревномерной сетки. Координаты узла представляются *pn*–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 опередеенная 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 $2^{pn}pn$ 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 *смещанным 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 сегодняшней 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.

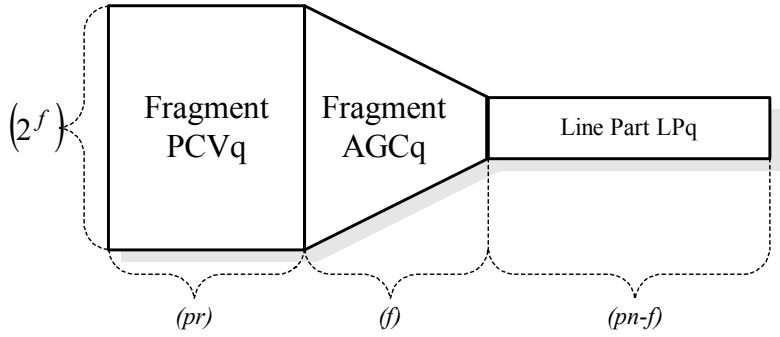

Рис. 6.0.1. Структура MCF

Такую структуру MCF назовем *горизонтальной фрагментацией*. При этом

- линейная часть LP_q кода MCF_q содержит *(pn-f)* разрядов,
- прямоугольная часть RCV_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

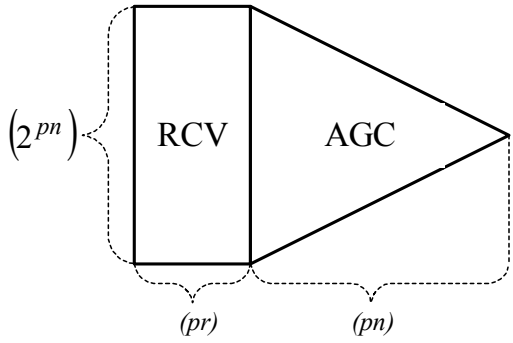

Рис. 6.1.1. Полная специализированная оперативная память

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».*

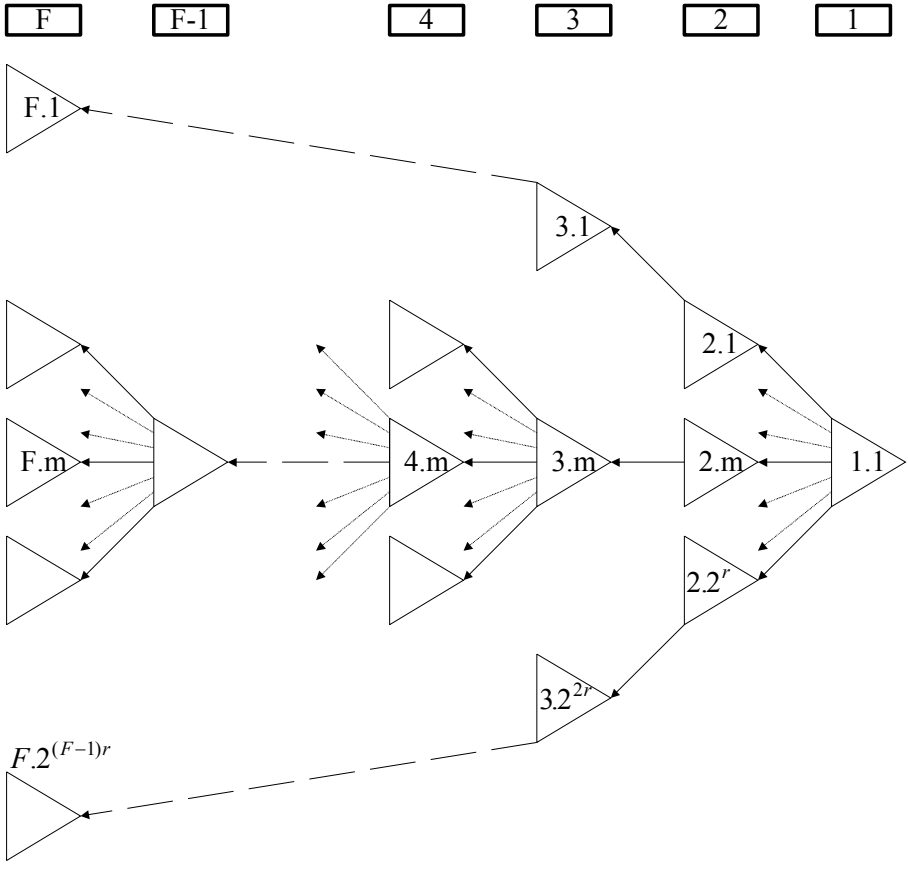

Рис. 6.2.1. Ярусы геометрического кода и сегменты линейного кода.

В пямяти фрагменты располагаются последовательно ярус за ярусом:

$$1.1,$$

$$2.1,\ 2.2,\ \ldots,\ 2.2^{r},$$

$$3.1,\ 3.2,\ \ldots,\ 3.2^{2r},$$

$$\ldots$$

$$k.1,\ k.2,\ \ldots,\ k.m,\ \ldots,\ k.2^{(k-1)r},$$

$$\ldots$$

$$F.1,\ F.2,\ \ldots,\ F.2^{(F-1)r}$$

При этом номер j фрагмента в этой последовательности связан с номером k яруса и номером m фрагмента в ярусе следующей зависимостью:

$$j = m + \sum_{a=1}^{k-1} 2^{(a-1)r} = m + \frac{1 - 2^{(k-1)r}}{1 - 2^{r}}. \qquad (6.2.1)$$

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:

$$f = m + v - 1. \qquad (6.2.2)$$

При этом номер фрагмента, куда поступает перенос,

$$j = f + \frac{1 - 2^{kr}}{1 - 2^{r}}. \qquad (6.2.3)$$

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,

$\left(2^{2r} - 1\right)$ - 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

$$a_1 = 1 + 2^r + 2^{2r} + \ldots + 2^{(F-1)r} = \frac{1 - 2^{Fr}}{1 - 2^r}$$

элементарных операций с фрагментом или

$$a_1 \approx 2^{(F-1)r} = 2^{n-r} \qquad (6.2.4)$$

Отношение разрядности оперативной памяти всех фрагментов SSRAM) к разрядности одного фрагмента SSRAM

$$R \approx 2^{n-r} \qquad (6.2.5)$$

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

$$\theta \approx 2^n + 3 \cdot 2^r = \left(3 + 2^F\right) \cdot 2^r \qquad (6.2.6)$$

6.3. Максимальное арифметическое устройство геометрических фигур

Максимальное арифметическое устройство геометрических фигур **MGAU** аналогично полной специализированной оперативной памяти PSSRAM оперирует с кодом MCF представленым на рис. 6.1.1. Это устройство содержит развитую схему распространения переносов и благодаря этому может выполнять следующие операции:

M1. Запись данного кода вектора – см. раздел 5.2.2.1.

M2. Резерв.

M3. Чтение кода вектора по адресу, в котором он расположен – см. раздел 5.2.2.3.

M4. Резерв.

M5. Резерв.

M6. Определение номера старшего значащего разряда в коде MCF, что необходимо для округления.

M7. Сложение MCF с данным кодом вектора – см. раздел 5.2.2.7.

M8. Умножение MCF на матрицу преобразования – см. раздел 5.2.2.9.

M9. Определение длины кода вектора по адресу.

M10. 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.

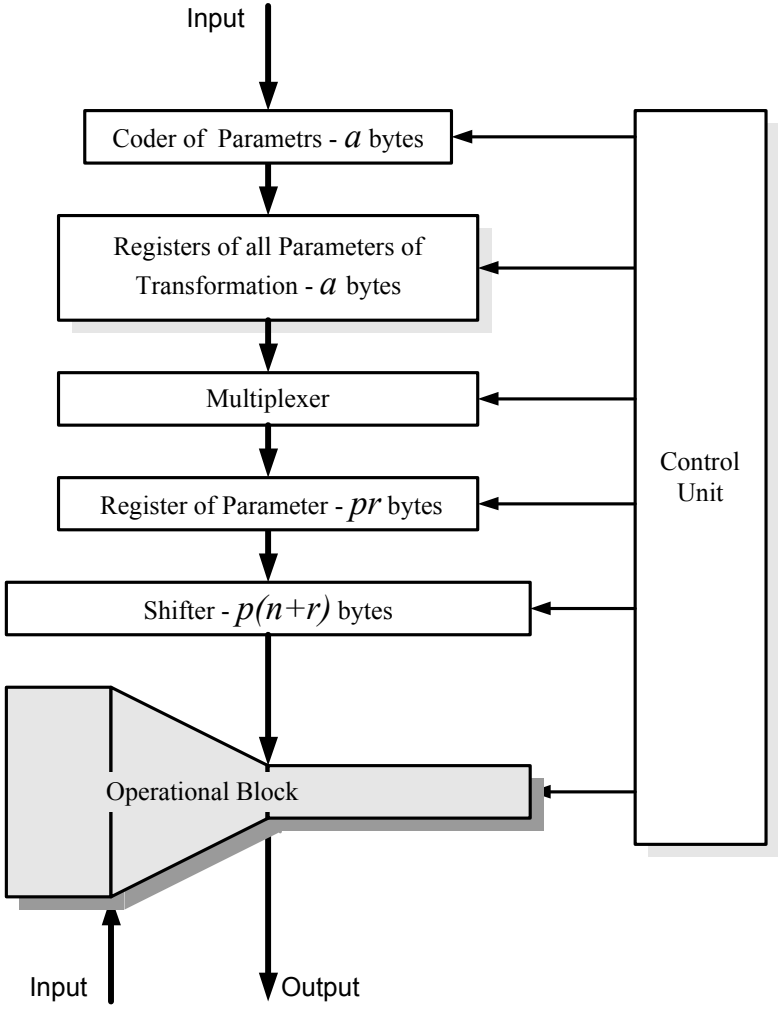

Рис. 6.4.1. Фрагментарное арифметическое устройство

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 првилам арифметики геометрических кодов.

Соотношение между разрядностью операционных блоков FVAU и FGAU

$$R_6 / R_7 \approx (n+r)/r \quad (6.4.3)$$

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.

A6. Выдача вектора по адресу - без округления или с округлением.

A7. Определение длины кода вектора по адресу.

A8. 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.

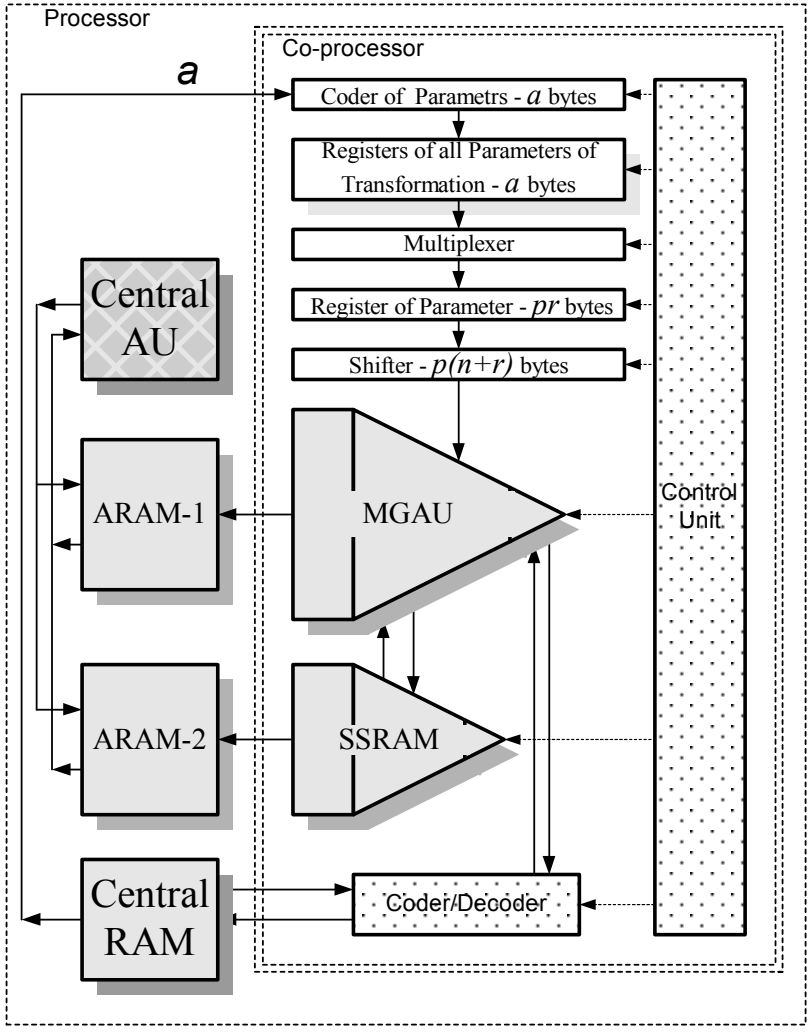

Рис. 6.5.1. Процессор с максимальным арифметическим устройством

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.

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 *M7*).

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.

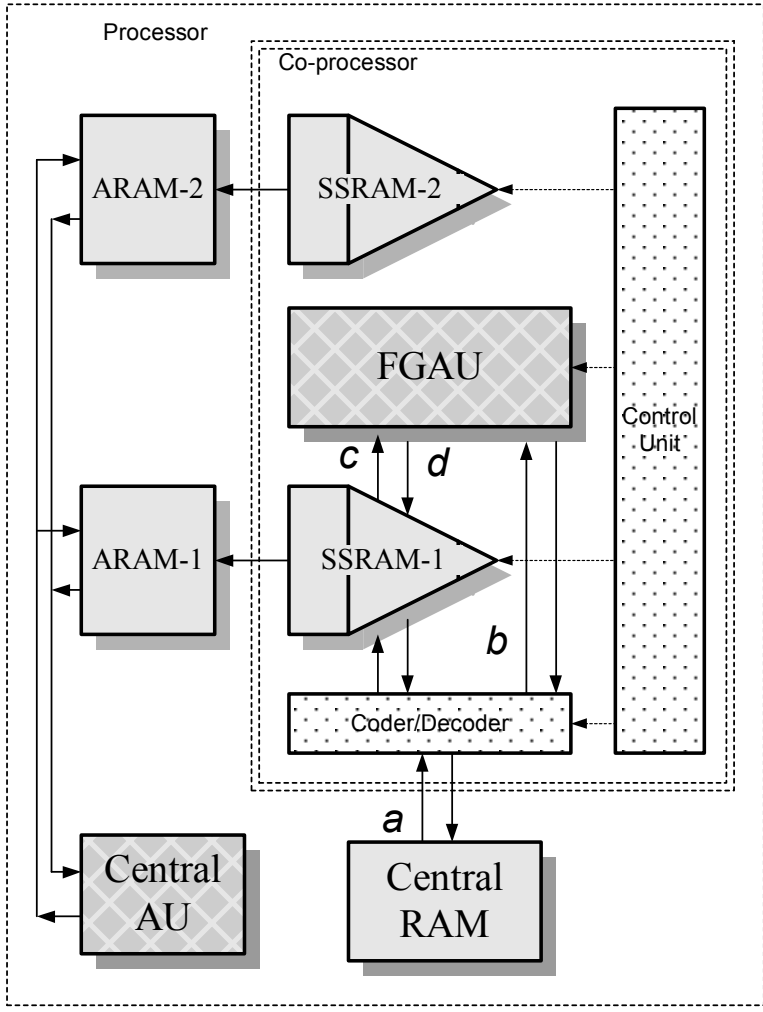

Рис 6.6.1. Процессор с фрагментарным FGAU

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 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 присутствуе 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 перобразование 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 перобразование 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 μ=1 или η=1 устанавливает соответствующий триггер в «1». Сигнал μ=0 или η=0 не изменяет состояния соответствующего триггера.

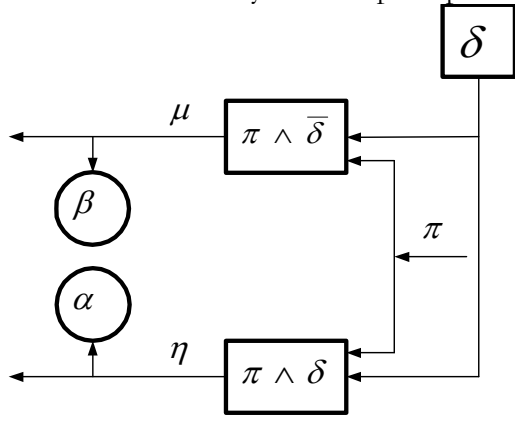

Рис. 6.8.1. Блок записи номера с данным кодом.

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 μ=1 или η=1 устанавливает соответствующий триггер в «1». Сигнал μ=0 или η=0 не изменяет состояния соответствующего триггера.

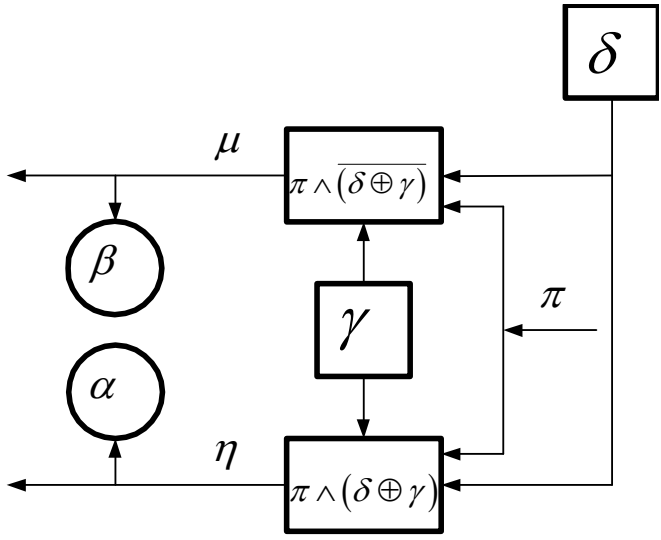

Рис. 6.8.2. Блок записи значения с данным кодом.

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 регистро кода значения. Сигнал ω вырабатывается в том разряде α или β, через который прошел сигнал переноса.

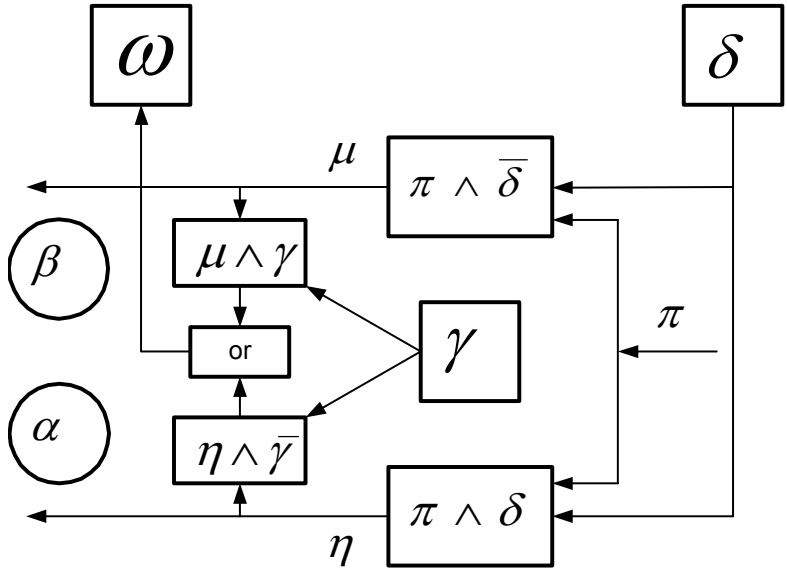

Рис. 6.8.3. Блок чтения значения с данным кодом.

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 α=0.

В частном случае при δ≡0 этот же сумматор выполняет функцию инвертора.

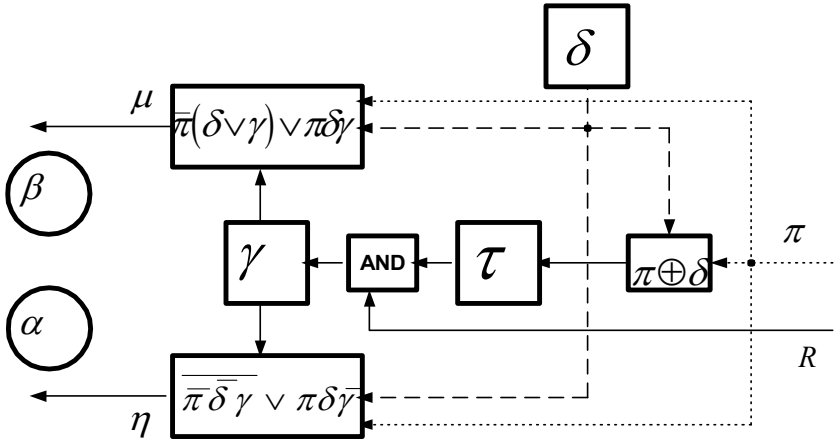

Рис. 6.8.4. Обратный сумматор.

6.8.5. Блок поиска первого открытого пути, его номера и его значения.

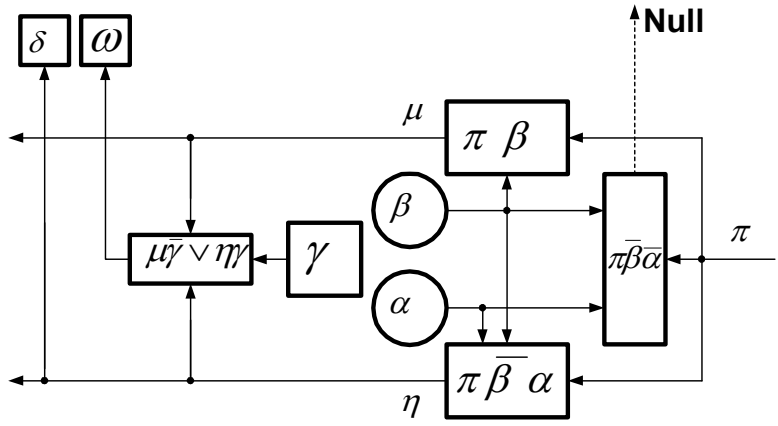

Рис. 6.8.5. Блок поиска первого открытого пути, его номера и его значения.

На рис 6.8.5 изображен фрагмент схемы для поиска первого открытого пути и чтения его номера и 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.

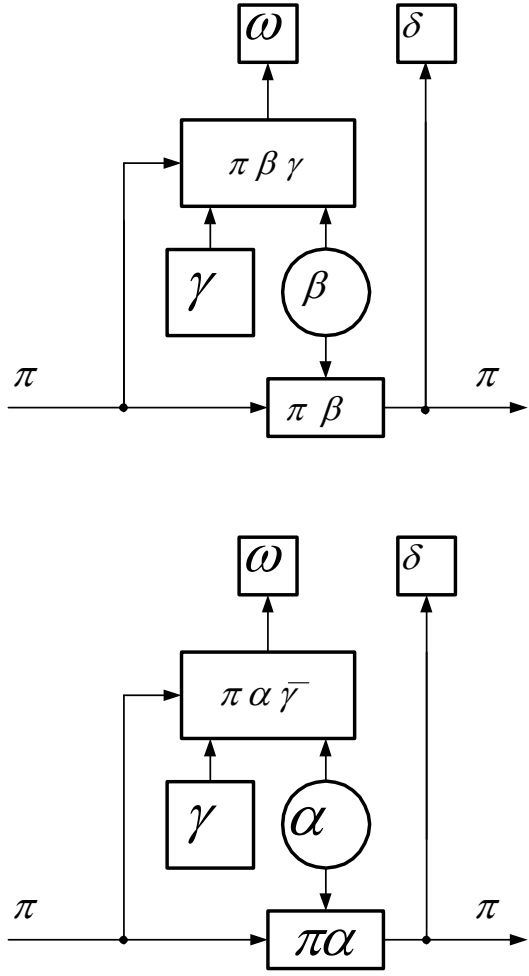

Рис. 6.8.6. Блок чтения номера и значения пути с данной терминальной вершиной.

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.

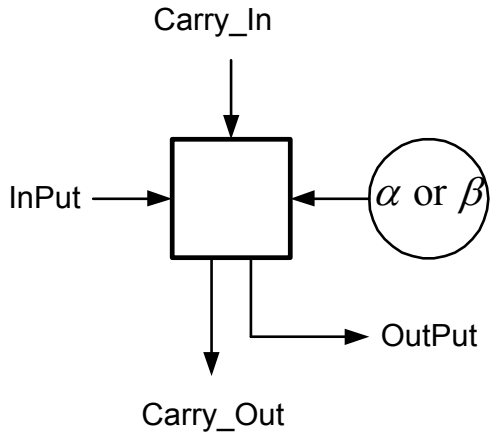

Рис. 6.8.7. Блок поиска следующей терминальной вершины.

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 аффином 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	$F\left(\frac{M}{F}pr + \sqrt[F]{M}\right)$	-
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 тзвестно статистика операций доступа к памяти.

В табл. 7.3. указан относительный объем всех рассмотренных выше устройств.

Таблица 7.3. Относительный объем сравниваемых устройств.

№		*AR*	*SR*
1	TRAM		$\frac{M^2 p(n+r)}{2}$
2	PSSRAM		$F^2\left(\frac{M}{F} pr + \sqrt[F]{M}\right)$
3	FSSRAM		Mpr
4	SAU	$14\,Mp^2(n+r)$	
5	MSAU		
6	FSAU		
7	VAU	$7Mp(n+r)$	
8	MVAU		
9	FVAU		
10	FGAU	Mpr	
11	MGAU	Mpr	Mpr

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	$W_1 = W_2 =$ $F^2\left(\frac{M}{F} pr + \sqrt[F]{M}\right)$
5	Скалярный процессор	$14\,Mp^2(n+r)$
8	Векторный процессор	$7Mp(n+r)$
10	Геометрический процессор	Mpr

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 $W2$ устройства TRAM при $n=r$ в M раз превышает относительный объем $W2$ устройства PSSRAM. При F>1 относительные объемы $W2$ устройств TRAM и FSSRAM относятся как $\frac{M(n+r)}{2Fr}$. Например, при $n=r$ эти объемы относятся как M/F.

Относительный объем $W1$ устройства TRAM при $n=r$ в 2 раза превышает относительный объем $W1$ устройства PSSRAM. При F>1 относительные объемы $W1$ устройств TRAM и FSSRAM относятся как $\frac{(n+r)}{Fr}$. Например, при $n=r$ эти объемы относятся как $2/F$, т.е относительный объем $W1$ устройства TRAM меньше в $F/2$ раз относительного объема $W1$ 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

$$W_T = Mp(n+r)\left(H + M/2\right)$$

и

$$W_F = F\left(\frac{M}{F}pr + \sqrt[F]{M}\right)(FH + F) \approx FMpr\,(H+1)\cdot$$

Отношение $\frac{W_F}{W_T} \approx \frac{FMpr\,(H+1)}{Mp(n+r)(H+M/2)}\cdot$

При $1 << H << M$, $n = r$ 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.

Литература

1. Sébastien Roy, Daniel Lefebvre and Henri H. Arsenault. Recognition invariant under unknown affine transformations of intensity, Optics Communications, Volume 238, 2004.
2. Shih-Hsuan Yang, Chun-Yen Liao, and Chin-Yun Hsieh. Watermarking MPEG-4 2D Mesh Animation in Multiresolution Analysis, Computer Science, Volume 2532, 2002.
3. Хмельник С.И., Кодирование плоских фигур. Автоматика и вычислительная техника, АН ЛССР, 1970, №6.
4. Хмельник С.И., Алгебра многомерных векторов и кодирование пространственных фигур. Автоматика и вычислительная техника, АН ЛССР, 1971, №1.
5. Хмельник С.И., Быстродействующие поисковые процедуры. Третий международный симпозиум по теории информации, часть II, Таллин, 1973.
6. Хмельник С.И., Многокритериальная задача о назначениях. Изв.АН СССР, Техническая кибернетика, №4, 1977.
7. Хмельник С.И., Поисковые процедуры с геометрическими кодами, ж. "Кибернетика", АН УССР, 1990, №6.
8. Хмельник С.И., Цифровое устройство для геометрических преобразований изображения. Авт. св. 333573, 1972, БИ-11
9. Хмельник С.И., Устройство для геометрических преобразований изображения. Авт. св. 1030816, БИ-27, 1983.
10. S. Khmelnik. Method and System for Processing Geometrical Figures. International patent application under PCT. PCT/CA02/00835, WO 02 099625 A2. Priority 07.06.01.
11. С. Хмельник. Компьютерная арифметика векторов, фигур и функций, изд. «Mathematics in Computers», Москва – Тель-Авив, 1995.
12. S. Khmelnik. A Method and System for Processing Complex Numbers. International patent application under PCT. PCT/CA01/00007, WO 01 50332 A2. Priority 05.01.00.
13. Хмельник С.И., Кодирование векторов, ж. "Кибернетика", АН УССР, 1969, №5.

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 (fragmetntary geometrical arithmetic unit)
FSAU – фрагментарное скалярное арифметическое устройство (fragmetntary scalar arithmetic unit)
FSSRAM – фрагментарная специализированная статическая оперативная память (fragmetntary specialized static random access memory)
FVAU – фрагментарное векторное арифметическое устройство (fragmetntary 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 - смещанный 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 FGAU
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

Список таблиц

Список рисунков