А.П. Стахов

«Золотая» арифметика как основа информационных технологий 21-го века и важный прикладной результат «современной теории чисел Фибоначчи» (к обоснованию «Математики Гармонии»)

1. Введение

В 70-е и 80-е годы 20-го столетия в Советском Союзе были проведены теоретические и инженерные разработки, которые показали высокую эффективность использования *кода Фибоначчи* (КФ) и *кода золотой пропорции* (КЗП) для создания принципиально нового типа компьютеров, названных *компьютерами Фибоначчи* или *«золотыми» компьютерами*, и новых средств измерительной техники («золотых» аналого-цифровых и цифроаналоговых преобразователей (АЦП и ЦАП)). Основной эффект от использования КФ и КЗП в вычислительной и измерительной техники состоял в существенном повышении контролеспособности компьютерных средств, а также точности и метрологической стабильности АЦП и ЦАП. Эти исследования были начаты в Таганрогском радиотехническом институте (1971-1977), а затем продолжены в Винницком политехническом институте (1977-1990). Теоретические результаты этих исследований опубликованы в книгах [1, 2], а инженерные разработке описаны в брошюре [3]. По данному направлению в СССР было проведено широкое патентование изобретений на тему «Компьютеры Фибоначчи» во всех ведущих странах-продуцентах компьютерной техники (США, Япония, Англия, ФРГ, Франция, Канада и др.). 65 зарубежных патентов являются официальными юридическими документами, которые подтверждают приоритет советской науки в этом важном направлении. В 1989 г. это направление было заслушано и одобрено на специальном заседании Президиума Академии наук Украины. К сожалению, «горбачевская перестройка» и резкое сокращение финансирования оборонных программ привело в конце 80-х годов к прекращению инженерных разработок в этом направлении. Но сами идеи создания «золотых» компьютеров как альтернативы «неймановских» компьютеров стали еще более актуальными для создания перспективных информационных технологий будущего, о чем я неоднократно писал в статьях [4-7].

Цель настоящей статьи – в популярной форме изложить основы «золотой» арифметики и еще раз привлечь внимание к проблеме создания «золотых» компьютеров как основы информационных технологий 21-го века.

2. Код Фибоначчи, код золотой пропорции и особенности «золотой» арифметики

В основу «золотых» компьютеров положены два новых позиционных представления:

Код Фибоначчи (КФ)

$$N = a_n F_n + a_{n-1} F_{n-1} + \ldots + a_i F_i + \ldots + a_1 F_1 , \qquad (1)$$

N – натуральное число, $a_i \in \{0,1\}$ - двоичная цифра i-го разряда КФ (1), i=0,±1,±2,±3,…, F_i - число Фибоначчи, вес i-го разряда КФ (1).

Заметим, что веса разрядов в КФ (1) связаны рекуррентным соотношением Фибоначчи:

$$F_i = F_{i-1} + F_{i-2}; F_1 = F_2 = 1; i = 1,2,3,...,n \tag{2}$$

Код золотой пропорции (КЗП)

$$A = \sum_i a_i \Phi^i, \tag{3}$$

где A – действительное число, $a_i \in \{0,1\}$ – двоичная цифра i-го разряда КЗП (3), i=0,±1,±2,±3,…, Φ^i - вес i-го разряда КЗП (3), $\Phi = \left(1+\sqrt{5}\right)/2$ (золотая пропорция) - основание системы счисления (3).

Заметим, что веса «золотых» разрядов Φ^i $(i = 0,\pm1,\pm2,\pm3,...)$ связаны *аддитивным соотношением*:

$$\Phi^i = \Phi^{i-1} + \Phi^{i-2}, \tag{4}$$

подобным рекуррентному соотношению Фибоначчи (2), и *мультипликативным соотношением*:

$$\Phi^i = \Phi \times \Phi^{i-1}. \tag{5}$$

Сделаем несколько важных замечаний, касающихся «золотой» арифметики:

1. КФ (2) используется для кодирования натуральных чисел, в частности, кодов программ и выполнения над кодами программ логических операций.
2. Благодаря наличию мультипликативного соотношения (5), КЗП (3) подобен классическому двоичному коду. Поэтому КЗП (3) используется для *кодирования действительных чисел А,* их представления в *форме с плавающей запятой* и выполнения арифметических операций *умножения* и *деления.* То есть, «золотая» арифметика в этой части подобна классической двоичной арифметике.
3. С другой стороны, благодаря наличию аддитивного соотношения (4), КЗП (3) подобен КФ (1). Поэтому в части выполнения арифметических операций *сложения* и *вычитания* «золотая» арифметика подобна «арифметике Фибоначчи».

3. Базовые микрооперации «золотой» арифметики

3.1. ***Свертка и развертка «золотых» разрядов.*** Основной особенностью КФ (1) и КЗП (3) является *многозначность* представления одного и того же числа. Различные представления одного и того же числа в КФ (1) и КЗП (3) могут быть получены одно из другого путем специфических преобразований над двоичными разрядами, называемыми *сверткой* и *разверткой* двоичных разрядов. Эти кодовые преобразования выполняются в рамках одной и той же двоичной комбинации, представляющей суммы (1) или (3). Эти кодовые преобразования основаны на рекуррентных соотношениях (2) и (4), связывающих веса разрядов в КФ (1) и КЗП (3). Операция *свертки* выполняется над группой из трех соседних разрядов

$a_i a_{i-1} a_{i-2} = 011$. Свертка состоит в замене тройки двоичных разрядов своими отрицаниями, то есть,

$$[011 \to 100] \tag{6}$$

Операция *развертки* выполняется над группой двоичных разрядов $a_i a_{i-1} a_{i-2} = 100$ и состоит в замене тройки двоичных разрядов своими отрицаниями, то есть,

$$[100 \to 011]. \tag{7}$$

Основное свойство этих операций состоит в том, что их выполнение в КФ (1) и КЗП (3) не приводит к изменению числа, представляемого этой кодовой комбинации в виде сумм (1) или (3). Ниже приведены примеры применения операций *свертки* и *развертки* для получения различных «фибоначчиевых» представлений чисел 7 и 5:

$$\text{Свертка}: \ 7 = \begin{cases} 0\ 1\ 1\ 1\ 1 \\ 1\ 0\ 0\ 1\ 1 \\ 1\ 0\ 1\ 0\ 0 \end{cases} \qquad \text{Развертка}: \ 5 = \begin{cases} 1\ 0\ 0\ 0\ 0 \\ 0\ 1\ 1\ 0\ 0 \\ 0\ 1\ 0\ 1\ 1 \end{cases} \tag{8}$$

Если в двоичном представлении КФ (1) или КЗП (3) выполнить все возможные *свертки*, то мы придем к характерному двоичному представлению, в котором две единицы рядом не встречаются (см. нижнее «фибоначчиевое» представление числа 7 в примере (8)). Такое двоичное представление называется *минимальной формой числа*. Заметим, что младший разряд «фибоначииевого» представления типа (8) в минимальной форме всегда равен 0.

Если в двоичном представлении КФ (1) или КЗП (3) выполнить все возможные *развертки*, то мы придем еще к одному характерному двоичному представлению, в котором два нуля рядом справа от старшего значащего разряда не встречаются (см. нижнее «фибоначчиевое» представление числа 5 в примере (8)). Такое двоичное представление называется *максимальной формой числа*. Заметим, что младший разряд в максимальной форме КФ (1) всегда равен 1 (см. «фибоначчиевое» представление числа 5 в примере (8)).

3.2. *Сравнение чисел в КФ и КЗП.* Простота сравнения чисел и вытекающая отсюда «наглядность» цифрового представления является одним из важнейших преимуществ позиционных систем счисления, включая двоичную систему. Доказано [1, 2], что КФ (1) и КЗП (3) обладают таким же свойством. Единственное отличие состоит в том, что перед сравнением «фибоначчиевые» или «золотые» представления приводятся к минимальной форме, после чего минимальные формы сравниваются поразрядно, начиная со старшего разряда, до момента появления первой пары несовпадающих разрядов.

Например, для сравнения двух чисел A=00111101101 и B=00111110110, представленных в КФ (1), необходимо выполнить следующее:

1. Приведение сравниваемых кодов к минимальной форме:

$$A = 0\boxed{\mathbf{011}}11\boxed{\mathbf{011}}\boxed{\mathbf{01}} = 010\boxed{\mathbf{011}}1010 = 0101001010;$$
$$B = 0\boxed{\mathbf{011}}111\boxed{\mathbf{011}}0 = 010\boxed{\mathbf{011}}11000 = 01010\boxed{\mathbf{011}}000 = 01010100000.$$

Здесь все кодовые комбинации, подлежащие свертке на каждом этапе приведения к минимальной форме, выделены.

2. Поразрядное сравнение минимальных форм чисел A и B, начиная со старшего разряда слева направо до появления первой пары несовпадающих разрядов:

$$A = 01010[\mathbf{0}]10010$$
$$B = 01010[\mathbf{1}]00000.$$

Мы видим, что первая пара несовпадающих разрядов содержит двоичную цифру 0 в минимальной форме первого числа A и двоичную цифру 1 в минимальной форме второго числа B. Это означает, что $B>A$.

3.3. *Базовые микрооперации.* Как известно, в классической двоичной арифметике основной арифметической операцией является *сложение*. Вычитание сводится к сложению путем использования понятий *инверсного* и *дополнительного кодов*. Умножение и деление выполняются с использованием операций сложения, вычитания и сравнения. В основе двоичного сложения лежит тривиальное тождество, связывающее двоичные числа:

$$2^m + 2^m = 2^{m+1}, \tag{9}$$

откуда вытекает следующая широко известная таблица сложения двоичных чисел:

$$\boxed{\begin{array}{lllll} 0 & + & 0 & = & 0 \\ 0 & + & 1 & = & 1 \\ 1 & + & 1 & = & 1 \\ 1 & + & 1 & = & 10 \end{array}} \tag{10}$$

Однако, особенность «золотой» арифметики состоит в принципиально новых правилах выполнения арифметических операций *сложения* и *вычитания* в КФ (1) и КЗП (3). Они основываются на четырех *базовых микрооперациях* [3]:

$$\text{Свертка:} \quad \boxed{\mathbf{011 \to 100}} \quad \text{Развертка:} \quad \boxed{\mathbf{100 \to 011}}$$
$$\text{Перемещение:} \quad \boxed{\begin{array}{lcr} 1 & & 0 \\ \downarrow & = & \\ 0 & & 1 \end{array}} \quad \text{Поглощение:} \quad \boxed{\begin{array}{lcr} 1 & & 0 \\ \updownarrow & = & \\ 1 & & 0 \end{array}} \tag{11}$$

Операции *свертка* и *развертка* рассмотрены нами выше: они основаны на основных рекуррентных соотношениях (2) и (4), связывающих веса разрядов в КФ (1) и КЗП (3). Эти операции являются *одноместными*, то есть, выполняются в рамках одной кодовой комбинации, расположенной в одном *регистре*. Напомним, что выполнение этих операций не приводит к изменению значения числа, находящегося в регистре.

Микрооперации *перемещение* и *поглощение* являются *двухместными*, то есть выполняются в рамках кодовых комбинаций, расположенных в двух регистрах. Операция *перемещение* выполняется над одноименными (k-ми) разрядами двух кодовых комбинаций КФ (1) и КЗП (3) при условии, что значение этого разряда в верхнем регистре равно 1, а в нижнем регистре равно 0. Операция состоит в «перемещении» $(\downarrow)\,1$ из верхнего регистра в нижний. Операция *поглощение* также является *двухместной* операцией. Она состоит во взаимном «поглощении» $(\updownarrow)$ единиц, находящихся в k-х разрядах верхнего и нижнего регистров.

3.4. ***Выполнение логических операций с использованием базовых микроопераций.*** Как известно, над кодами программ выполняются следующие операции: *счет* и *вычитание единиц,* а также различные *логические операции* типа *логического сложения, логического умножения, инверсии, сложения по модулю 2* и др. В «золотом» компьютере коды программ представляются в КФ (1). Продемонстрируем вначале возможность выполнения *логических операций* над двоичными комбинациями КФ (1) с использованием введенных выше *базовых микроопераций.* Выполним, например, все возможные *перемещения* из верхнего регистра A в нижний регистр B:

$$
\begin{array}{lcccccccc}
A & = & \mathbf{1} & 0 & 0 & \mathbf{1} & 0 & 0 & 1 \\
 & & \downarrow & & & \downarrow & & & \\
B & = & \mathbf{0} & 1 & 0 & \mathbf{0} & 1 & 0 & 1 \\
\hline
A' & = & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
B' & = & 1 & 1 & 0 & 1 & 1 & 0 & 1
\end{array}
$$

Мы получили две новые кодовые комбинации A' и B' как результат *перемещений.* Важно подчеркнуть, что двоичная комбинация A' является *логической конъюнкцией* $(\wedge)$ исходных двоичных комбинаций A и B, то есть,

$$A' = A \wedge B,$$

а двоичная комбинация B' является *логической дизъюнкцией* $(\vee)$ исходных двоичных комбинаций A и B, то есть,

$$B' = A \vee B.$$

Логическая операция *сложения по модулю 2* выполняется путем одновременного выполнения всех возможных операций *перемещения* и *поглощения.* Например,

$$
\begin{array}{lcccccccccccl}
A & = & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & & \\
 & & \updownarrow & & \downarrow & & & \downarrow & \downarrow & & \updownarrow & & \\
B & = & \mathbf{1} & 1 & \mathbf{0} & 0 & 1 & \mathbf{0} & \mathbf{0} & 0 & \mathbf{1} & & \\
\hline
A' & = & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & = & const\,0 \\
B' & = & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & = & A \oplus B
\end{array}
$$

Мы видим, что две новые кодовые комбинации $A' = \mathrm{const}\,0$ и $B' = A \oplus B$ являются результатом этого преобразования.

Логическая операция *инверсия кода A* сводится к выполнению операции *поглощения* над исходной кодовой комбинацией A и «единичной» кодовой комбинацией $B = const1$:

$$
\begin{array}{lcccccccccc}
A = & 1 & 0 & \mathbf{1} & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} \\
 & \updownarrow & & \updownarrow & & & \updownarrow & \updownarrow & & \updownarrow \\
B = & \mathbf{1} & 1 & \mathbf{1} & 1 & 1 & \mathbf{1} & \mathbf{1} & 1 & \mathbf{1}
\end{array}
$$

$$
\begin{array}{l}
A' \; = 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \ = const\ 0 \\
B' \; = 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 0 \ = \quad \overline{A}
\end{array}
$$

3.5. ***Счет и вычитание единиц в КФ.*** Как известно, счётчики используются для построения таймеров или для выборки инструкций из ПЗУ в микропроцессорах. Они могут использоваться как делители частоты в управляемых генераторах частоты (синтезаторах). Во многих случаях применения счетчиков, к ним предъявляется требование высокого быстродействия. Как известно, быстродействие счетчика характеризуется временем установления в нем нового состояния. «Критической» ситуацией для счетчиков, построенных на основе классической двоичной системы счисления, является установление нового состояния счетчика в следующей ситуации:

$$0111111111+1=1000000000.$$

В этом случае осуществляется перенос единицы из младшего в старший разряд счетчика. Если счетчик строится на так называемых «счетных триггерах», то такой счетчик обладает очень низким быстродействием. Для ускорения переноса используются схемы группового переноса, что приводит к усложнению схемы счетчика.

Покажем теперь, что *фибоначчиевые счетчики*, основанные на микрооперациях *свертки* (суммирующий счетчик) и *развертки* (вычитающий счетчик) обладают существенными преимуществами по сравнению с двоичными счетчиками. *Счет* единиц в КФ (1) (*суммирующий счетчик*) осуществляется следующим образом. Перед добавлением единицы в младший разряд исходное «фибоначчиевое» представление, соответствующее числу N, приводится в такую форму, чтобы значение младшего разряда было равным 0. Затем в младший разряд добавляется 1, что приводит к тому, что числовое содержание счетчика становится равным $N+1$. После этого с помощью *свертки* «фибоначчиевое» представление числа $N+1$ приводится в такую форму, чтобы значение младшего разряда было равным 0. Продемонстрируем «фибоначчиевый» счет на следующем примере:

$$
\begin{aligned}
&000000+1=0000\boxed{\mathbf{01}}=000010=1\\
&000010+1=000\boxed{\mathbf{011}}=000100=2\\
&000100+1=0001\boxed{\mathbf{01}}=000110=3\\
&00\boxed{\mathbf{011}}0+1=0010\boxed{\mathbf{01}}=001010=4\\
&001010+1=001\boxed{\mathbf{011}}=001100=5\\
&0\boxed{\mathbf{011}}00+1=0100\boxed{\mathbf{01}}=010010=6\\
&010010+1=010\boxed{\mathbf{011}}=010100=7\\
&010100+1=0101\boxed{\mathbf{01}}=010110=8\\
&01\boxed{\mathbf{011}}0+1=\boxed{\mathbf{011}}0\boxed{\mathbf{01}}=100010=9\\
&100010+1=100\boxed{\mathbf{011}}=100100=10\\
&100100+1=1001\boxed{\mathbf{01}}=100110=11\\
&10\boxed{\mathbf{011}}0+1=1010\boxed{\mathbf{01}}=101010=12\\
&101010+1=101\boxed{\mathbf{011}}=1\boxed{\mathbf{011}}00=\boxed{\mathbf{110}}000=000000
\end{aligned}
\tag{12}
$$

Здесь жирным шрифтом в скобках выделены те ситуации, когда в фибоначчиевом счетчике осуществляются *свертки*. Рассмотрим ситуацию перехода от «фибоначчиевого» представления числа $8=010110$ к «фибоначчиевому» представлению числа 9. В этом случае одновременно с записью 1 в младший (первый) разряд осуществляется *свертка* единиц 2-го и 3-го разрядов в 4-й разряд ($011\to100$), что приводит к получению «фибоначчиевого» представления числа $9=011001$. На следующем этапе в этом «фибоначчиевом» представлении осуществляется две свертки одновременно, что приводит к появлению нового «фибоначчиевого» представления $\mathbf{011}0\mathbf{001}\to\mathbf{100}00\mathbf{10}\mathbf{=9}$. Таким образом, особенность «фибоначчиевого» счета состоит в том, что в любой ситуации переход от числа N к числу N+1 осуществляется за время последовательного выполнения не более двух сверток. Заметим, что нижний ряд таблицы (12) соответствует *переполнению* фибоначчиевого счетчика.

Вычитание единиц в КФ (*вычитающий счетчик*) осуществляется путем вычитания единицы из младшего разряда «фибоначчиевого» представления числа N, в котором значение младшего разряда равно 1, с последующей *разверткой* в младших разрядах. Рассмотрим пример функционирования вычитающего «фибоначчиевого» счетчика:

$$
\begin{aligned}
&1111-1=11\boxed{\mathbf{10}}=1101=6\\
&1101-1=1\boxed{\mathbf{100}}=1011=5\\
&1011-1=10\boxed{\mathbf{10}}=1001=4\\
&\boxed{\mathbf{100}}1-1=0110=0101=3\\
&0101-1=0\boxed{\mathbf{100}}=0011=2\\
&0011-1=00\boxed{\mathbf{10}}=0001=1\\
&0001-1=0000=0
\end{aligned}
\tag{13}
$$

Здесь жирным шрифтом в скобках выделены те ситуации, когда в фибоначчиевом счетчике осуществляются *развертки*. Таким образом, особенность «фибоначчиевого» вычитающего счетчика состоит в том, что в любой ситуации переход от числа N к числу N-1 осуществляется за время последовательного выполнения не более двух разверток.

Рассмотренные выше алгоритмы функционирования суммирующего и вычитающего «фибоначчиевых» счетчиков показывают, что в таких счетчиках заложены предпосылки для конструирования супер-быстрых счетчиков (без использования сложных схем группового переноса). То есть, уже этот простейший пример показывает преимущества кодов Фибоначчи перед классическим двоичным кодом.

3.6. *Сложение в КФ и КЗП.* В качестве примера рассмотрим сложение следующих чисел, представленных в КФ:

$$A_0=010100100 \text{ и } B_0=001010100.$$

При сложении мы используем микрооперации *перемещения, свертки и развертки*. Сумма формируется в нижнем регистре B.

Первый шаг сложения состоит в выполнении всех возможных *перемещений* двоичных 1 из регистра A в регистр B:

$$\begin{array}{lcccccccccc}
A_0 & = & 0 & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 1 & 0 & 0 \\
 & & & \downarrow & & \downarrow & & & & & \\
B_0 & = & 0 & \mathbf{0} & 1 & \mathbf{0} & 1 & 0 & 1 & 0 & 0 \\
\hline
A_1 & = & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
B_1 & = & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0
\end{array}$$

Второй шаг состоит в выполнении всех возможных *разверток* в двоичной комбинации A_1 и всех возможных *сверток* в двоичной комбинации B_1, то есть,

$$A_1=000000\boxed{\mathbf{100}}\to A_2=000000011$$

$$B_1=\boxed{\mathbf{011}}110100\to B_2=100110100$$

Третий шаг состоит в выполнении всех возможных *перемещений* двоичных 1 из регистра A в регистр B:

$$\begin{array}{lcccccccccc}
A_2 & = & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} \\
 & & & & & & & & & \downarrow & \downarrow \\
B_2 & = & 1 & 0 & 0 & 1 & 1 & 0 & 1 & \mathbf{0} & \mathbf{0} \\
\hline
A_3 & = & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
B_3 & = & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1
\end{array}$$

Сложение закончено, поскольку все двоичные 1 перемещены в из регистра A в регистр B. После приведения фибоначчиевой двоичной комбинации B_3 к минимальной форме, мы получим сумму $B_3=A+B$, представленную в минимальной форме:

$$B_3=10\boxed{\mathbf{011}}\boxed{\mathbf{011}}1=1010010\boxed{\mathbf{01}}=101001010.$$

Точно так же эти правила могут быть использованы для сложения чисел в КЗП (3).

3.7. *Вычитание в КФ и КЗП.* Идея вычитания числа *B* из числа *A*, представленных в КФ (1) или КЗП (3) также основывается на *базовых микрооперациях* (11). Суть вычитания состоит во взаимном *поглощении* всех двоичных единиц в числах *A* и *B* до тех пор, пока одно из них не станет равным 0. Для этого используются микрооперации *поглощения* и *развертки.* Результат вычитания всегда формируется в регистре, который содержит большее число. Если результат вычитания формируется в верхнем регистре *A*, это означает, что результат вычитания имеет знак «+»; в противном случае результат вычитания отрицательный.

Рассмотрим следующий пример. Пусть требуется произвести вычитание числа B_0=101010010 из числа A_0=101001000 в КФ (1).

Первый шаг состоит в *поглощении* всех возможных 1 в числах *A* и *B*:

$$
\begin{array}{lcccccccccc}
A_0 & = & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 1 & 0 & 0 & 0 \\
 & & \updownarrow & & \updownarrow & & & & & & \\
B_0 & = & \mathbf{1} & 0 & \mathbf{1} & 0 & 1 & 0 & 0 & 1 & 0 \\
\hline
A_1 & = & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
B_1 & = & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0
\end{array}
$$

Второй шаг состоит в выполнении микрооперации *развертка* в кодовых комбинациях A_1 и B_1:

$$A_1 = 00000\boxed{\mathbf{100}}0 \to A_2 = 000000110$$

$$B_1 = 0000\boxed{\mathbf{100}}\boxed{\mathbf{10}} = B_2 = 000001101$$

Третий шаг состоит в выполнении микрооперации *поглощение* над кодовыми комбинациями A_2 и B_2:

$$
\begin{array}{lcccccccccc}
A_2 & = & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} & 1 & 0 \\
 & & & & & & & & \updownarrow & & \\
B_2 & = & 0 & 0 & 0 & 0 & 0 & 1 & \mathbf{1} & 0 & 1 \\
\hline
A_3 & = & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
B_3 & = & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1
\end{array}
$$

Четвертый шаг состоит в *развертке* кодовых комбинаций A_3 и B_3:

$$A_3 = 0000000\boxed{\mathbf{10}} \to A_4 = 000000001$$

$$B_3 = 00000\boxed{\mathbf{100}}1 \to A_4 = 000000111$$

Пятый шаг состоит в выполнении *поглощений* над кодовыми комбинациями A_4 и B_4:

$$
\begin{array}{lcccccccccc}
A_4 & = & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1} \\
 & & & & & & & & & & \updownarrow \\
B_4 & = & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & \mathbf{1} \\
\hline
A_5 & = & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
B_5 & = & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0
\end{array}
$$

Вычитание закончено. После приведения кодовой комбинации B_5 к минимальной форме получаем:

B_5=000001000.

Результат вычитания находится в регистре *B*. Это означает, что результат вычитания отрицательный, то есть разность чисел *A*-*B* равна:

$$R = A - B = -000001000 .$$

Заметим, что такие же правила используются для вычитания чисел, представленных в КЗП (3).

Отметим существенное преимущество правил вычитания, основанных на *базовых микрооперациях.* Сложение и вычитание осуществляется в *прямом коде,* то есть в «золотой» арифметике мы не используем понятия *инверсного* и *дополнительного* кодов, принятые в классической двоичной арифметике, что упрощает арифметические структуры.

3.8. ***Представление чисел с плавающей запятой, умножение и деление в КЗП.*** Эти операции основываются на мультипликативном свойстве (5) КЗП (3).

Таблица «золотого» умножения, основанная на соотношении (5), имеет следующий вид:

0×0	=	0
0×1	=	0
1×0	=	0
1×1	=	1

Таким образом, таблица «золотого» умножения совпадает с таблицей умножения в классической двоичной системе счисления.

Еще одно сходство с классической двоичной арифметикой – возможность представления «золотых» чисел в форме с плавающей запятой.

В качестве примера рассмотрим «золотое» умножение правильных дробей:

A=0.010010 и B=0.001010.

Перед умножением представим сомножители в форме с плавающей запятой:

$$A = 010010\times\Phi^{-6}; B = 001010\times\Phi^{-6}.$$

Это означает, что мантиссы и порядки сомножителей *A* и *B* соответственно равны:

$$m(A) = 010010;\ e(A) = -6 \text{ and } m(B) = 001010; e(B) = -6.$$

Умножение мантисс:

$$\begin{array}{ccccccccccc}
 & & & & & 0 & 1 & 0 & 0 & 1 & 0 \\
 & & & & & 0 & 0 & 1 & 0 & 1 & 0 \\
\hline
 & & & & & 0 & 0 & 0 & 0 & 0 & 0 \\
 & & & & 0 & 1 & 0 & 0 & 1 & 0 & \\
 & & & 0 & 0 & 0 & 0 & 0 & 0 & & \\
 & & 0 & 1 & 0 & 0 & 1 & 0 & & & \\
 & 0 & 0 & 0 & 0 & 0 & 0 & & & & \\
0 & 0 & 0 & 0 & 0 & 0 & & & & & \\
\hline
0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0
\end{array}$$

Заметим, что сложение в данном примере выполняется по правилам «золотой» арифметики. После сложения порядков: $e(A)+e(B)=-12$ и приведения результата умножения мантисс к минимальной форме получим результат умножения в форме с плавающей запятой:

$$A\times B=00100000100\times\Phi^{-12}.$$

Деление в «золотой» арифметике выполняется подобно тому, как это делается в двоичной арифметике, то есть сводится к сравнению и вычитанию; при этом вычитание осуществляется по правилам «золотой» арифметики.

4. ***Самоконтролирующийся «золотой» процессор.*** Такой проект финансировался Министерством общего машиностроения СССР (советским ракетным министерством). Именно по заказу Минобщемаша в Специальном конструкторско-технологическом бюро «Модуль» Винницкого технического университета в 1986 г. началась разработка самоконтролирующегося «золотого» процессора, предназначенного для использования в бортовых системах управления ракетами [3]. Задача состояла в том, чтобы с использованием рассмотренных выше *базовых микроопераций* разработать «золотой» процессор, позволяющий эффективно обнаруживать и исправлять случайные ошибки, возникающие в процессоре в результате сбоев триггеров в момент их переключения, чтобы исключить выполнения ложной программы.

Аппаратная реализация *самоконтролирующегося «золотого» процессора* основана на принципе *причина – следствие* (*cause-effect*), изложенного в [3]. Сущность принципа демонстрируется с помощью Рис.1. Процессор на Рис. 1 состоит из информационного и контрольного регистров, связанных логическими схемами «*причина*» (“cause”) и «*следствие*» (“effect”). Исходная информация, которая подвергается обработке, преобразуется в некоторый *результат* с использованием некоторой *базовой микрооперации*. Информация о *результате* фиксируется в контрольном регистре. После этого с помощью логической схемы «*следствие*» (“effect”) проверяется соответствие *результата* некоторой *причине,* то есть *результат* должен соответствовать своей *причине.*

Например, при выполнении микрооперации *свертка* над кодовой комбинации 011 (“cause”) мы получаем новую кодовую комбинацию 100 (“effect”), которая является необходимым условием для выполнения микрооперации *развертки*. Это означает это, правильное выполнение *свертки* приводит к условию для выполнения *развертки*. Аналогично правильное выполнение *развертки* приводит к условию *свертки*. Из этого рассуждения вытекает, что микрооперации *свертка* и *развертка* являются взаимно контролируемыми. Этот же принцип справедлив для остальных *базовых микроопераций*.

При «регистровой интерпретации» установление соответствия между «причиной» и «результатом» реализовалось с помощью «контрольного триггера». «Причина» устанавливает соответствующий «контрольный триггер» в состояние 1, а правильное выполнение микрооперации («результат» соответствует «причине») переключает «контрольный триггер» в состояние 0. Если «результат» не соответствует «причине» (микрооперация выполнена неправильно), тогда «контрольный триггер» остается в состоянии 1 и этот факт является свидетельством ошибки.

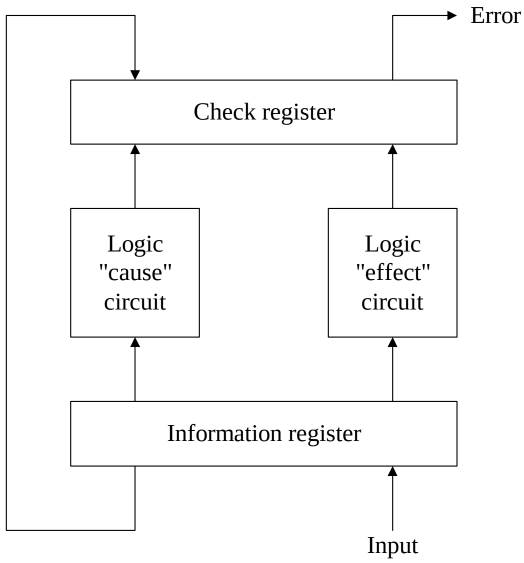

Рисунок 1. Блок-схема «золотого» процессора

Рассмотренный выше принцип "причина-следствие" был положен в основу самоконтролирующегося "золотого" процессора, который был реализован в виде БИС, спроектированных НПО «Научный Центр» (г. Зеленоград). Процессор выполнял следующие микрооперации: *запись, чтение, свертка, развертка, перемещение, поглощение, сложение, вычитание, приведение к минимальной форме, циклический сдвиг, конъюнкция, дизъюнкция, сложение по модулю 2*. Наличие контрольного выхода "Ошибка" являлось важным преимуществом "золотого" процессора. Если на контрольном выходе модуля формировался двоичный сигнал 1 (наличие "ошибки"), то все информационные выходы модуля блокировались, то есть ошибочная команда не выполнялась. Для исправления "ошибки" предшествующая микрооперация повторялась. Если при повторении на контрольном выходе возникал двоичный сигнал 0 (отсутствие "ошибки"), то это означало, что "ошибка" возникла в результате "случайного сбоя", и блокирование информационных выходов снималось. Если при повторении микрооперации двоичный сигнал 1 снова появляется на выходе "ошибка", это означало, что "ошибка" является следствием «отказа» в модуле и блокирование информационных выходов оставалось.

Интересное решение использовалось при кодировании команд программы, выполняемой процессором. Все коды программы представлялись в *минимальной форме* КФ (1), что само по себе позволяло контролировать коды программы. Однако при записи кода программы в ячейки памяти с четными адресами использовались *минимальные формы* представления информации в КФ (1), а в случае записи кода программы в ячейки памяти с четными адресами они преобразовывались в *максимальные формы*. Это обеспечивало довольно эффективный контроль кодов программы при их считывании из памяти по принципу *минимальной* или *максимальной формы* кодового представления.

5. Заключение

1. Таким образом, принципы построения «золотых» компьютеров, основанные на использовании КФ (1) и КЗП (3) а также на «золотой» арифметике [1-7], являются основой для создания компьютеров принципиально нового типа – «золотых» компьютеров, в которых **за счет использования новых правил выполнения арифметических операций может быть достигнута высокая надежность всех преобразований информации в компьютере.** Такие компьютеры могут найти широкое применение в перспективных информационных технологиях 21-го века, особенно для таких приложений, в которых надежность обработки информации является определяющим требованием к компьютеру.

2. Коды Фибоначчи [1] и коды золотой пропорции [2] и основанная на них новая компьютерная арифметика («золотая» арифметика) являются **важным прикладным результатом «современной теории чисел Фибоначчи» и нацеливает ее на создание новых компьютеров.**

Литература

1. Стахов А.П. Введение в алгоритмическую теорию измерения. Москва: Советское радио, 1977
2. Стахов А.П. Коды золотой пропорции. Москва: Радио и связь, 1984
3. Помехоустойчивые коды: Компьютер Фибоначчи, Москва, Знание, серия «Радиоэлектроника и связь», вып.6, 1989 г.
4. А.П. Стахов, Тьюринг, филлотаксис, математика гармонии и «золотая» информационная технология. Часть 1. Математика Гармонии // «Академия Тринитаризма», М., Эл № 77-6567, публ.14876, 16.09.2008
5. А.П. Стахов, Тьюринг, филлотаксис, математика гармонии и «золотая» информационная технология. Часть 2. «Золотая» Информационная Технология // «Академия Тринитаризма», М., Эл № 77-6567, публ.14878, 19.09.2008
6. А.П. Стахов, Автобиографическая повесть: компьютеры Фибоначчи, «Золотая» Информационная Технология, Математика Гармонии и «Золотая» Научная Революция) // «Академия Тринитаризма», М., Эл № 77-6567, публ.15083, 10.02.2009
7. А.П. Стахов, Десять прорывных технологий 21-го века и «золотая» информационная технология // «Академия Тринитаризма», М., Эл № 77-6567, публ.15251, 24.04.2009