Особенности программирования троичной машины: новые возможности и новые задачи

А.А. Бурцев[1]

[1] ФГУ ФНЦ НИИСИ РАН, Москва, Россия, burtsev@niisi.msk.ru

Аннотация. В статье рассматриваются новые выразительные средства построения программ, которые может предоставить программисту троичная машина и которые обеспечивает система программирования ДССП для троичной виртуальной машины (ТВМ). А также демонстрируется способ построения формулы вычисления троичной логической функции по заданной таблице значений с последующей её реализацией в ДССП-ТВМ.

Ключевые слова: троичный компьютер, троичная арифметика, троичная логика, троичная функция, ЦМ “Сетунь” и “Сетунь-70”, троичная виртуальная машина (ТВМ), ДССП, ДССП-ТВМ.

1. Введение

В настоящее время общепризнано, что традиционная двоичная микроэлектроника уже почти достигла потолка своих потенциальных возможностей. Поэтому сейчас назрела острая потребность расширить область исследований, направленных на поиск новых путей развития вычислительной техники. В качестве одного из перспективных вариантов дальнейшего развития отечественной компьютерной индустрии может рассматриваться и возможность построения на современной элементной базе троичных компьютеров, т.е. компьютеров, функционирование которых основывается на применении троичной симметричной системы счисления, а также присущей ей троичной логики и троичной арифметики.

Когда-то на заре развития вычислительной техники, называя троичную симметричную систему счисления (с цифрами -1,0,+1) “самой изящной” [1], Д.Кнут отмечал её безусловные преимущества (в частности, перед традиционной двоичной, на которой основана вся современная вычислительная техника) и предрекал ей успешное применение в будущем. А это будущее стало реальностью (правда, лишь на некоторое время) уже во второй половине XX века с разработкой в НИЛ ЭВМ МГУ (под руководством Брусенцова Н.П.) троичных цифровых машин (ЦМ) “Сетунь”(1958) и “Сетунь-70”(1970) [2], позволивших на практике ощутить преимущества троичной логики и арифметики.

Спустя почти полвека сотрудники НИЛ троичной информатики (ТИ) ВМК МГУ предприняли попытки возродить на современной основе троичный компьютер, воплотив в нём характерные черты ЦМ “Сетунь-70”: двухстековую архитектуру с управляющими командами процедурного структурированного программирования. Программисты создали имитатор [3] троичной машины и кросс-систему ДССП-ТВМ [4] разработки программ для неё. А разработчики аппаратуры придумали [5], как можно создать на современной элементной базе такие микросхемы, которые смогли бы функционировать аналогично троичным элементам ЦМ “Сетунь” и “Сетунь-70”, и тем самым обозначили принципиальную возможность реализации в настоящее время подобной троичной машины «в кремнии».

Созданный в МГУ программный комплекс ДССП-ТВМ [6] можно использовать в настоящее время в качестве среды программирования [7] для разработки и прогона учебных программ для троичной машины. Что позволяет готовить практических специалистов в области троичной информатики [8].

Для переноса на троичную машину программ, созданных ранее для двоичных машин, приходится учитывать ряд особенностей, характерных для троичных операций обработки данных. Следует учитывать, что на троичной машине с симметричной системой счисления результаты некоторых операции над числами могут оказаться отличными от тех, которые получались на машине с обычной двоичной позиционной системой счисления (см. п. 2.1)

Но такую специфику троичных операций можно легко освоить. А что же дальше? Что такого нового может предоставить троичная машина, чего не умеет делать двоичная? Подобные ставшие уже традиционными вопросы задают обычно специалисты, сомневающиеся в особых преимуществах троичных компьютеров.

Опыт программирования, полученный авто-

ром при разработке троичных программ для ТВМ (в ДССП-ТВМ) и для ЦМ “Сетунь-70” (на ассемблере), позволяет с уверенностью утверждать, что ассортимент выразительных средств построения программного обеспечения, предоставляемый троичной машиной, не только полностью обеспечивает привычный арсенал средств, обеспечиваемый программисту на двоичной машине, но и значительно перекрывает его, обогащая программиста новыми приёмами разработки программ, которые двоичными машинами и языками программирования для них либо вообще никогда не предусматривались, либо широко не применялись на практике в силу неэффективности их реализации. Вот об этих новых возможностях программирования, предоставляемых троичной машиной, и пойдёт, главным образом, речь в предлагаемой статье.

В качестве таковых далее рассматриваются: команда организации троичного ветвления, новая команда цикла с троичным условием, способы построения троичного логического выражения с применением поразрядных операций троичной логики (и арифметики), а также приёмы эффективной реализации операций для работы с троичными множествами. Приводятся варианты воплощения этих команд и операций в ЦМ «Сетунь-70», в троичной виртуальной машине (ТВМ) и в системе программирования ДССП для ТВМ. Демонстрируются примеры программ, разработанных в кросс-системе ДССП-ТВМ (или в среде интерпретатора ДССП/ТВМ), применяющих такие специфические «троичные» команды и операции. Сравнивая их с аналогичными программами, разработанными для решения тех же задач с применением арсенала традиционных выразительных средств, обеспечиваемых обычно языками программирования для двоичных машин, можно убедиться, что реализация таких программ на троичной машине позволяет не только упростить сам алгоритм решения задачи, но и добиться в итоге улучшения его производительности.

Применение троичных машин неизбежно приводит и к расширению круга решаемых на компьютере задач, а также порождает и новые проблемы, без решения которых невозможно рассчитывать на успешное внедрение троичной вычислительной техники. Образно говоря, новые возможности как бы позволяют выйти из плоскости в пространство и увидеть на горизонте новые задачи. Одной из них является задача построения формулы вычисления троичной функции по заданной таблице её значений на основе имеющихся базовых операций троичной логики и арифметики. В булевой алгебре широко известен универсальный метод построения формулы логической функции по её таблице истинности, приводящий к получению совершенной ДНФ или совершенной КНФ, пригодных для реализации требуемой функции на основе трёх базисных операций: конъюнкции (&), дизъюнкции (|) и отрицания (~).

Аналогичный универсальный метод построения формулы вычисления функции троичной логики, возможно, тоже существует, но, к сожалению, он не так широко известен. Автор на практике убедился, что иногда предпочтительнее самому изобрести такой метод заново, чем искать его описание на просторах Интернета или печатных страниц научных книг и журналов. Опробовав такой «заново придуманный» метод на практике при реализации некоторых двухместных функций троичной логики в ДССП-ТВМ, автор предлагает ознакомиться с этим методом тем специалистам, кто проявляет живой интерес к практическому освоению приёмов разработки программ для троичных машин.

2. Специфика вычислительной обработки в троичной машине

Допустим, нам требуется перенести на троичную машину какую-то вычислительную программу, которая ранее уже была разработана для двоичной машины (скажем, на языке Си). Какие препятствия или так называемые «подводные камни» нам могут встретиться на этом пути? Рассмотрим их поподробнее.

2.1. Целочисленная арифметика

В троичной симметричной системе счисления единообразно представляются положительные и отрицательные числа. И результат выполнения целочисленной операции в такой троичной машине всегда знаковый и потому остаётся корректным, если он укладывается в 27-разрядную сетку, т.е. не превосходит по модулю значение $(3^{27}-1)/2$. (Заметим, что 3^{27} = 7625597484987 >> 2^{32} = 4294967296).

Но не всякая операция целочисленной арифметики будет исполняться на троичной машине с симметричной системой счисления так же, как на двоичной. Есть исключения.

В первую очередь, следует отметить, что в такой троичной машине операция целочисленного деления вычисляет в качестве результата частное и остаток так, чтобы последний оказался как можно меньше по абсолютной величине. При этом (в отличие от обычной операции деления) значение остатка может оказаться не только положительным, но и отрицательным.

В некоторых случаях результат такого деления на троичной машине может совпадать с результатом, полученным на двоичной. Напри-

мер, 16%3 = [5,1] (т.к. 5*3+1=16). А в других случаях он может отличаться. Так, на двоичной машине 17%3 = [5,2] (5*3+2=17), а на троичной машине 17%3 = [6,-1] (6*3-1=17).

В базовом словаре ДССП-ТВМ слово, обозначаемое знаком деления **/** , выполняет целочисленное деление, получая результат [частное, остаток] как раз по описанным выше правилам, характерным для троичной симметричной системы счисления. А для выполнения привычного целочисленного деления в ДССП-библиотеке предусмотрено определение слова **/DivMod** :

```
: /DivMod {X,Y}
  C2 ABS C2 ABS {X,Y,|X|,|Y|} / {X,Y,D,M}
{корректируем результат, если остаток < 0 : }
 {X,Y,D,M} C IF- D-1_M+|Y|
{ корректируем знаки результатов: }
 {X,Y,D,M} C4 IF- NEG {меняем знак для M}
 {X,Y,D,M} E4 SGN C3 SGN <>
 IF+ NEG {и знак для D, если sign(X)<>sign(Y)}
 {M,Y,D} E2D E2 {D,M} ;
: D-1_M+|Y| {X,Y,D,M}
   C3 ABS + E2 1- E2 {D-1,M+|Y|} ;
```

Другое исключение касается операций сдвигов (<<, >>), которые могут применяться при вычислении арифметического выражения (например, в языке Си) для умножения и деления значения на степень двойки (2^n). При исполнении таких операций на троичной машине результат окажется умноженным (или поделённым) уже на степень тройки (3^n).

Подобные особенности обработки целых чисел не позволяют, вообще говоря, легко перенести на троичную машину многие программы, уже разработанные для двоичных машин.

2.2. Операции доступа к трайтам и троичным словам памяти

Есть ещё одна причина, которая может затруднить автоматический перенос существующего ПО на троичную машину. Это различие в строении и соответственно в адресации памяти у двоичных и троичных машин.

Память в двоичной машине обычно адресуется с точностью до байта. Так что при обработке машинных слов, образованных из нескольких байтов, например, элементов массивов, хранимых в 16-, 32- или 64-разрядных машинных словах, для передвижения к следующему слову требуется прибавлять к адресу-указателю соответственно значение 2, 4 или 8.

Память троичного компьютера будет адресоваться с точностью до трайта. А диапазон адресов будет охватывать пространство от отрицательного значения –P, до положительного значения +P, где $P=(3^{27}-1)/2$, если для адреса отводится 27 разрядов. Причём нулевое значение адреса будет указывать не на начало всего пространства памяти (как это обычно бывает в двоичной машине), а на его середину. Что тоже приходится учитывать при размещении программ и данных в памяти троичной машины.

В троичной машине при обработке элементов массивов, занимающих два трайта или полное 27-тритное машинное слово (3 трайта), для передвижения к следующему элементу к адресу-указателю придётся прибавлять уже совсем другие значения (2 или 3 соответственно). А значит, все программы, работающие с массивами и указателями, потребуется тщательно переделывать для переноса на троичную машину.

В ТВМ, разработанной в НИЛ троичной информатики на ВМК МГУ, можно читать и записывать в память по указанному адресу троичные слова, трайты и двухтрайтовые значения. Для этого предусмотрены команды (слова в языке ДССП-Т) **@W**, **@T, @TT** для взятия значения из памяти в вершину стека, и команды **!W**, **!T, !TT** для записи в память троичного значения из стека.

В качестве примера демонстрации обработки троичных слов в памяти приведём простую процедуру сортировки по возрастанию вектора 27-тритных значений (на языке ДССП-Т):

```
{{ A - адрес вектора, n= кол-во элементов
: SortVctr{A,n} 1- DO- Sorti D { };
 : Sorti{A,i} 1+ Maxi {A,i,k}
    C2 C2 <> IF+ Exchi D 1- {A,i};
: Maxi{опр-ет индекс k макс.эл-та среди A0..Ai }
   {A,i} C C {A,i,k,i} DO- Maxj {A,i,k} ;
 : Maxj{A,i,k,j} C4 C {~,A,A} C3 @[x] E2
   {~,Aj,A} C4 @[x] {~,Aj,Ak}
    > IF+ k:=j {A,i,k,j} ; {j=i-1,..,0}
  : k:=j {k,j} E2 D C {j,j} ;
  : @[x] {A,x} *3 + @W {A[x]} ;
: Exchi {A,i,k} {переставляет A[i]⇔A[k] }
   C3 C3 *3 + {~,'Ai} C4 C3 *3 +
   {~,'Ai,'Ak} C @W C3 @W E4
   {~,Ai,'Ak,Ak,'Ai} !W {A[i]:=Ak}
                     !W {A[k]:=Ai} {A,i,k} ;
```

Процедура **SortVctr** сортирует вектор методом простого выбора. На каждом i-ом шаге цикла (i=n-1,..,1) она находит среди элементов вектора A0..Ai максимальный, используя процедуру **Maxi** для определения его индекса k, и переставляет этот k-ый элемент с i-ым элементом, используя процедуру **Exchi**.

2.3. Поразрядная (потритная) обработка троичного слова

Ещё одну проблему при переносе ПО на троичную машину могут предоставить программы, в которых как отдельные величины обрабатываются значения, размещённые внутри одного машинного слова.

Для доступа к таким величинам в программах обычно применяются операции логических сдвигов (<<, >>), поразрядной конъюнкции (&),

дизъюнкции (|), и отрицания (~), которые позволяют выделять в машинном слове отдельные биты или битовые поля, образуемые соседними разрядами одного машинного слова.

При разработке программ для троичной машины нужны аналогичные операции, чтобы обеспечить доступ к отдельным тритам и троичным значениям полей, образованных соседними тритами машинного слова. Для поразрядной обработки данных внутри троичного машинного слова ДССП-ТВМ предоставляет набор команд для поразрядного умножения, сложения, инверсии (**TMUL, TADD, NEG**), а также команды сдвигов влево и вправо на одну или несколько позиций (**SHL, SHR, SHT**).

Продемонстрируем характерные приёмы использования этих троичных операций для обработки отдельных разрядов и полей троичного машинного слова. И приведём для сравнения примеры аналогичных действий на языке Си с битами и битовыми полями:

взять значение V из разрядов [9:12] слова S	
ДССП-Т	`{S} -9 SHT #44 TMUL {V}`
Си	`V=(S>>9)&(0xF)`

записать значение V в разряды [9:12] слова S	
ДССП-Т	`{S,V} #44 TMUL +9 SHT E2 #14444443014444 TMUL TADD{S'}`
Си	`S=S&(0xFFFFE1FF)|((V&0xF)<<9))`

взять значение L из 5-го разряда слова S	
ДССП-Т	`{S}#300 TMUL -5 SHT{L}`
Си	`L= ((S&0x2F)>>5)`

записать значение L в 5-й разряд слова S	
ДССП-Т	`{S,L} #1 TMUL +5 SHT E2 #1444444444144 TMUL TADD{S'}`
Си	`S=S&(0xFFFFFFDF)|((L&0x1)<<5))`

Сложнее реализовать операции чтения и записи троичного флага, если разряд этого флага в троичном слове не фиксируется заранее, а определяется в ходе исполнения программы. Тогда необходимые значения масок нельзя приготовить заранее в виде констант: их приходится формировать динамически в зависимости от заданного номера (n) разряда троичного флага.

Покажем, например, как можно определить в ДССП-ТВМ такие операции чтения и записи над произвольным разрядом вершины стека:

```
{ запись значения флага L в n-й трит вершины }
: T!I {V[27:0],L,n} E2 #1 TMUL
  C2 SHT E3 {L<<n,n,V[27:0]} E2
  M+j0 TMUL TADD {V'[27:0] } ;
{ M+j0 =маска, где M[j]= 0, M[i]=+1 для i<>j }
```

```
{ чтение значения n-го трита вершины стека }
: T@I {V[27:0],n}
    NEG SHT #1 TMUL {V[n]} ;
```

3. Троичное условие и троичное логическое выражение

В нашей повседневной жизни достаточно часто встречается необходимость троичного ответа на вопрос. Например, на вопрос «как закончился матч?» возможны три варианта ответа: выигрыш – ничья − проигрыш. На хрестоматийный вопрос богатырю из русских сказок: «куда двигаться?» возможен ответ: влево – прямо – направо. Ещё примеры ответов с тремя значениями: дружба – нейтралитет − вражда; больше – равно − меньше; выше – вровень − ниже; тяжелее – равновесно − легче; горячее – одинаковой температуры – холоднее; лучше – нормально – хуже; вчера – сегодня – завтра.

По сути на любой вопрос, ответить на который нельзя ни утвердительно («да»), ни отрицательно («нет»), не получив нужной информации, в качестве третьего варианта надо предусматривать ответ «не знаю». Например, в настоящее время наука вряд ли может чётко ответить (да или нет) на вопрос «есть ли жизнь на Марсе?». И тут ответ «неизвестно» будет наиболее подходящим.

Вычислять выражения троичной логики, предусматривающие не два, а три варианта ответа, на троичной машине можно непосредственно (напрямую). При этом значения +1, 0, −1 можно трактовать по-разному. Например, так: «да» (+1), «нет» (−1), «не знаю» (0); или немного иначе: «да» (+1), «нет» (0), «не знаю» (−1) . А путём составления троичных логических выражений (условий) можно более эффективно решать некоторые задачи.

В базовом наборе команд ДССП-ТВМ имеются операции, выдающие в качестве результата троичное логическое значение. Это операция оценки знака числа **SGN** и операция сравнения двух чисел **CMP**. А также предусмотрены операции, которые позволяют производить поразрядные действия с отдельными тритами машинного слова. Это потритная инверсия **NEG**, потритный минимум **TMIN** и потритный максимум **TMAX**. С помощью этих операций можно составлять (на языке ДССП-Т) сколь угодно сложные троичные логические выражения.

В качестве первого примера определим в ДССП для ТВМ слово-выражение TSEG для троичной проверки попадания точки **z** в сегмент прямой **[x,y]**, которое в качестве результата L выдаёт одно из трёх значений: 1) L=+1, если точка **z** расположена внутри отрезка (x<z<y); 2) L=0, если точка **z** лежит на одной из граница отрезка (**z**=**x** или **z**=**y**); 3) L=−**1**, если точка **z** находится вне отрезка (z<x или z>y) :

```
: TSEG{x,y,z} C E4
  {z,y,z,x} CMP {z?x} E3{z?x,y,z}
   CMP {z?x,y?z} TMIN {L} ;
```

и сравним его с фрагментом (на языке Си), который позволяет решить эту задачу на двоичной машине:

```
if((x<z)&&(z<y)) L=+1; else
if((z=x)||(z=y)) L= 0; else L=-1;
```

Заметим, что для двоичной машины (на языке Си) при выполнении такой троичной проверки приходится использовать команды ветвления, а на языке ДССП-Т (для троичной машины) можно без них обойтись, используя лишь логические операции.

Аналогично можно составить логические выражения для троичной проверки попадания точки с координатами **(x,y)** в заданную фигуру на плоскости. Результат такой проверки= **+1 (**если точка внутри фигуры); **0 (**если точка на границе фигуры); **−1 (**если точка вне фигуры).

В качестве примеров составим определения слов (на языке ДССП-Т) для выполнения троичных проверок (условий) попадания точки с заданными координатами в следующие две фигуры: A) прямоугольник размерами g×h, левый нижний угол которого расположен в начале координат; и B) круг радиуса r с центром в начале координат:

троичное условие попадания точки **(x,y)** в фигуру A: прямоугольник высотой **h** и шириной **g**, левый нижний угол которого расположен в начале координат

```
: inFigA? {x,y,g,h} E4 0 E3 E2
  {h,y,0,g,x} TSEG {h,y,x?in[0,g]}
   E3 0 E3 {x?in[0,g],0,h,y} TSEG
  {x?in[0,g],y?in[0,h]} TMIN { z=+1|0|-1 } ;
```

троичное условие попадания точки **(x,y)** в фигуру B: круг радиуса r с центром в начале координат

```
: inFigB? {x,y,r} C * E3 C * E2 C * +
  {r^2,x^2+y^2} CMP {z= +1| 0|-1 } ;
```

Применяя дополнительные операции потритной инверсии, а также потритного максимума и минимума как логические связки, можно выразить и более сложные троичные выражения-условия попадания точки в объединение (AUB) или пересечение двух фигур (A∩B). А также составить соответствующие проверки для фигур, образованных вырезанием фигуры B из фигуры A (A\B), или наоборот, фигуры A из фигуры B (B\A):

троичное условие попадания точки в фигуру AUB

```
: inFigA+B? {x,y,r,g,h}
   E3 C4 6 CT E3 {x,y,h,g,x,y,r} inFigB?
   5 ET {(x,y)inFigA?,y,h,g,x} E4 E3
   {(x,y)inFigA?,x,y,g,h} inFigA?
   {(x,y)inFigA?,(x,y)inFigB?} TMAX
    { z= (x,y)inFig(A+B)? } ;
```

троичное условие попадания точки в фигуру A∩B

```
: inFigA*B? {x,y,r,g,h}
   E3 C4 6 CT E3 {x,y,h,g,x,y,r} inFigB?
   5 ET {(x,y)inFigA?,y,h,g,x} E4 E3
   {(x,y)inFigA?,x,y,g,h} inFigA?
   {(x,y)inFigA?,(x,y)inFigB?} TMIN
    { z= (x,y)inFig(A*B)? } ;
```

троичное условие попадания точки в фигуру A\B

```
: inFigA\B? {x,y,r,g,h}
   E3 C4 6 CT E3 {x,y,h,g,x,y,r} inFigB?
   NEG 5 ET {~(x,y)inFigA?,y,h,g,x} E4 E3
   {~(x,y)inFigA?,x,y,g,h} inFigA?
   {~(x,y)inFigA?,(x,y)inFigB?} TMIN
    { z= (x,y)inFig(A\B)? } ;
```

троичная проверка попадания точки в фигуру B\A

```
: inFigB\A? {x,y,r,g,h}
   E3 C4 6 CT E3 {x,y,h,g,x,y,r} inFigB?
   5 ET {(x,y)inFigA?,y,h,g,x} E4 E3
   {(x,y)inFigA?,x,y,g,h} inFigA?
   {(x,y)inFigA?,(x,y)inFigB?} NEG TMIN
    { z= (x,y)inFig(B\A)? } ;
```

Другие примеры составления троичных логических выражений для вычисления функций троичной логики рассматриваются в п.6.

4. Троичное множество

Для представления обычных множеств на двоичных машинах обычно резервируется массив машинных слов, в котором для каждого потенциально возможного элемента отводится один бит. Присутствие элемента в множестве фиксируется значением 1 в этом бите, а отсутствие − значением 0.

На троичной машине можно эффективно обрабатывать такие множества, относительно вхождения элемента в которые предусматривается не два (да, нет), а три варианта ответа: (да, нет, неизвестно). Будем называть такие множества троичными. Для работы с ними каждому элементу, претендующему на вхождение в множество, будем отводить в памяти один трит.

Троичное множество можно образно представить некой фигурой на плоскости с обозначенной границей. Элементу, входящему в него, соответствует некая точка на плоскости внутри этой фигуры (а его трит = +1). Элементу, который не входит в него, соответствует некая точка на плоскости вне этой фигуры (а его трит = −1). Если про элемент неизвестно, входит ли он в это множество, то ему соответствует точка на границе этой фигуры (а его трит 0).

Для множества, количество элементов в котором будет не более 27, в ДССП-ТВМ достаточно объявить переменную (типа TWORD), занимающую в памяти одно 27-тритное ма-

шинное слово. Каждому j-му элементу множества будет сопоставлен j-ый трит такого слова (j=0,1,..,26):

```
{ объявление множества A } TWORD VAR SA
{ объявление множества B } TWORD VAR SB
```

При заполнении множества требуется записывать в j-ый трит ответ на вопрос о вхождении j-ого элемента в это множество. Это можно сделать с помощью рассмотренной ранее операции записи (**T!I**) значения отдельного j-ого трита в троичное слово, помещённое в вершину стека.

Вопрос, входит ли j-ый элемент в множество (+1), не входит (−1) или про это ничего неизвестно (0), можно выяснить, применяя операцию чтения **T@I** значения отдельного j-ого трита троичного слова, предварительно записав значение множества в вершину стека.

Продемонстрируем, как можно первоначально сформировать значения множеств. Пусть в одномерных массивах (векторах) X[0:26] и Y[0:26] заданы значения координат некоторого набора точек плоскости:

```
{ вектора, задающие координат точек: }
 26 VCTR X  26 VCTR Y
```

И пусть на плоскости представлены две фигуры. Фигура A − это прямоугольник высотой h=12 и шириной g=16, левый нижний угол которого расположен в начале координат, а фигура B – это круг радиуса r=10 с центром в начале координат.

Сформируем множества SA и SB, проверяя, какие из заданных точек попадают в фигуры A и B. Чтобы сформировать множество SA, проверим каждую точку (Xi,Yi) (i=0,1,..,26) на попадание в фигуру A и отметим этот факт в переменной SA, представляющей множество A:

```
{ для каждой j–ой точки j=0,..,26 выполним: }
 SA j X j Y {Xj ,Yj} 16 12 inFigA?
     j T!I ! SA
```

Аналогично проверим каждую точку на попадание в фигуру B и отметим этот факт в переменной SB, представляющей множество B:

```
{ для каждой j–ой точки j=0,..,26 выполним: }
 SB j X j Y {Xj ,Yj} 10 inFigB?
     j T!I ! SB
```

С троичными множествами можно выполнять такой же набор характерных операций, который обеспечивался для работы с обычными множествами на двоичной машине. Можно проверять принадлежность элемента множеству (операцией T@I), включать или исключать элементы из множества (T!I), а также получать из двух множеств новое множество путём их объединения (TMAX), пересечения (TMIN) или вычитания одного множества из другого (NEG TMIN).

Продемонстрируем выполнение таких операций над множествами на языке ДССП-Т:

```
{ проверка, есть ли j-й элемент в множестве A}
  SA j T@I
{ проверка, есть ли j-й элемент в множестве B}
  SB j T@I
{ включить j-ый элемент в множество A}
   SA +1 j T!I ! SA
{ включить j-ый элемент в множество B}
   SB +1 j T!I ! SB
{ исключить j-ый элемент из множества A}
   SA -1 j T!I ! SA
{ исключить j-ый элемент из множества B}
   SB -1 j T!I ! SB
{ объявление нового множества C } VAR SC
{  пересечение множеств: C=A∩B }
   SA SB TMIN ! SC
{ объединение множеств: C=AUB }
   SA SB TMAX ! SC
{ вычитание множества B из A: C=A\B }
   SA SB NEG TMIN ! SC
{ вычитание множества A из B: C=B\A }
   SA NEG SB TMIN ! SC
```

5. Троичные управляющие конструкции

Благодаря возможности формулировать логическое условие с тремя возможными вариантами ответа на троичной машине можно применять новые виды ветвлений и циклов: троичное ветвление и цикл с троичным условием.

5.1. Троичное ветвление

Троичное ветвление означает, что для выбора дальнейшего хода исполнения программы предусматривается не два, а три возможных пути её продолжения. В ДССП для троичной машины (и в самой ТВМ) для организации такого ветвления предусмотрена команда **BRS**. Она позволяет выбрать на исполнение одну из трёх операций (процедур) в зависимости от знака значения, сформированного в вершине стека (см. блок-схему варианта А на рис. 1).

Следует упомянуть, что подобную команду **<BR P1,P2,P3>** было предложено реализовать ещё в ЦМ «Сетунь-70» [9]. Впоследствии она сразу же появилась в самой первой версии ДССП, реализованной для микрокомпьютера «Электроника-НЦ03Д».

В языках программирования высокого уровня (для двоичных машин) оператор троичного ветвления предлагается крайне редко (хотя он был когда-то предусмотрен в языке Фортран), видимо, из-за его неэффективности. Ведь для его реализации на двоичной машине придётся использовать (в сгенерированном коде) две проверки и соответственно две команды двоичного ветвления (см. блок-схему варианта Б на рис. 1). А при необходимости организации троичного ветвления в программе просто предла-

гается воспользоваться оператором двоичного ветвления 2 раза.

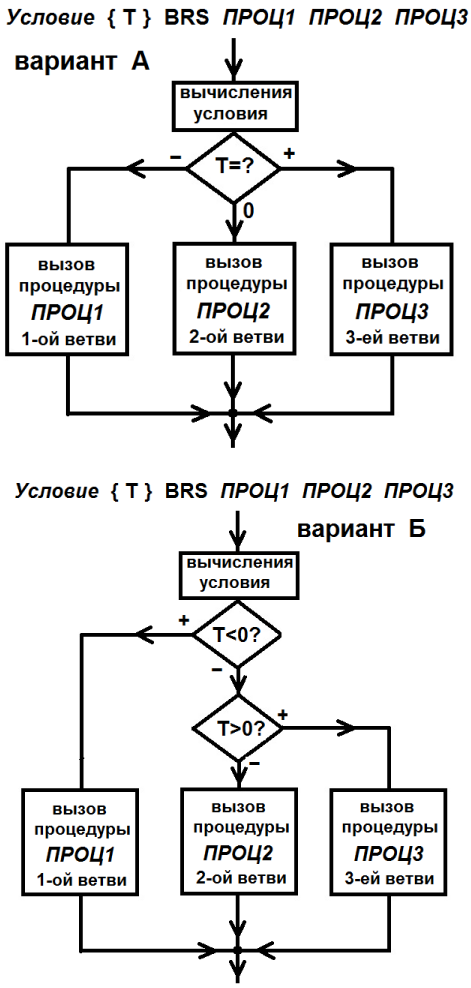

Рис. 1. Блок-схема реализации троичного ветвления на троичной (А) и на двоичной машине (Б).

Пожалуй, простейшим примером, демонстрирующим потребность троичного ветвления, является формула вычисления знака числа:

$$z = \begin{cases} -1, x < 0 \\ 0, x = 0 \\ +1, x > 0 \end{cases}$$

Сравним варианты её реализации для двоичной (на языке Си) и троичной машины (на языке ДССП-Т):

Си	`if (x<0) z=-1; else` `if (x>0) z=+1; else z=0;`
ДССП-Т	`: SGN{x} BRS -1 0 +1 {z};`

В качестве ещё одного примера продемонстрируем варианты реализаций (на языке ДССП-Т для троичной машины и на языке Си для двоичной машины) троичной операции сравнения двух чисел:

$$z = \begin{cases} -1, x < y \\ 0, x = y \\ +1, x > y \end{cases}$$

Си	`if (x<y) z=-1; else` `if (x>y) z=+1; else z=0;`
ДССП-Т	`: CMP{x,y} –` `{x-y} BRS -1 0 +1 {z};`

Следует сказать, что слова **SGN** и **CMP** уже предусмотрены в базовом словаре ДССП. И в ассемблерном ядре ДССП их тела представлены вызовами соответствующих команд (**SIGN** и **CMP**) троичной машины. А здесь их определения (на языке ДССП-Т) приведены лишь для демонстрации применения команды **BRS**.

Сами же эти операции **SGN** и **CMP** могут, в свою очередь, применяться для формирования более сложного троичного логического выражения (условия), по результату которого организуется троичное ветвление командой **BRS**. Например, они могут пригодиться при определении слова, предназначенного для вычисления функции z=F(x,y):

$$z = \begin{cases} x + y, x < y \\ x, x = y \\ x - y, x > y \end{cases}$$

Си	`if (x<y) z=x+y; else` `if (x>y) z=x-y; else z=x;`

ДССП-Т	`: F{x,y} C2 C2 CMP`
	`{x,y,x?y} BRS + D –`
	`{z=x+y|x|x-y} ;`

Заметим, что при определении в ДССП слова **F**, как и при определении слов **CMP** и **SGN**, использовалась лишь одна команда троичного ветвления. А для двоичной машины (на языке Си) для реализации тех же действий приходится использовать две команды ветвления.

Выигрыш, обеспечиваемый применением команд троичного ветвления, становится ещё более ощутимым с возрастанием количества анализируемых вариантов продолжения программы. Рассмотрим пример функции, при вычислении которой потребуется проанализировать 9 вариантов:

$$F(x,y) = \begin{cases} \begin{cases} x^2 + y^2 + 1 & \text{при } y > 0, x > 0 \\ x^4 & \text{при } y = 0, x > 0 \\ x^2 + y^3 & \text{при } y < 0, x > 0 \end{cases} \\ \begin{cases} y^2 - 1 & \text{при } y > 0, x = 0 \\ 1 & \text{при } y = 0, x = 0 \\ y^3 + 1 & \text{при } y < 0, x = 0 \end{cases} \\ \begin{cases} x^4 + y^4 & \text{при } y > 0, x < 0 \\ x & \text{при } y = 0, x < 0 \\ x^3 + y^3 & \text{при } y < 0, x < 0 \end{cases} \end{cases}$$

Приведём пример её определения в ДССП для ТВМ с использованием команд троичного ветвления:

```
: F{x,y} C2 BRS x- x0 x+ {z} ;
 : x-{x,y} C BRS x-y- x-y0 x-y+ {z};
  : x-y-{x,y} ^3 E2 ^3 + {z=x^3+y^3};
  : x-y0{x,y} D {z=x} ;
```

```
  : x-y+{x,y} ^4 E2 ^4 + {z=x^4+y^4};
 : x0{x,y} C BRS x0y- x0y0 x0y+ {z};
  : x0y-{x,y} ^3 1+ E2D {z=y^3+1};
  : x0y0{x,y} D T1 {z=1} ;
  : x0y+{x,y} ^2 1- E2D {z=y^2-1};
 : x+{x,y} C BRS x+y- x+y0 x+y+ {z};
  : x+y-{x,y} ^3 E2 ^2 + {z=x^2+y^3};
  : x+y0{x,y} D ^4 {z=x^4} ;
  : x+y+{x,y} ^2 1+ E2 ^2 +
              {z=x^2+y^2+1};
  : ^2 {a} C * {a^2} ;
  : ^3 {a} C C * * {a^3} ;
  : ^4 {a} ^2 ^2 {a^4} ;
```

И сравним его с вариантом реализации этой функции на языке Си для двоичной машины:

```
int FC(int x, int y) {
int z;
if (x>0) {
  if (y>0)    z= x*x + y*y + 1; else
  if (y==0)   z= x*x*x*x;       else
 /*if(y<0)*/ z= x*x + y*y*y;
}else if (x==0) {
  if (y>0)    z= y*y - 1;       else
  if (y==0)   z= 1              else
 /*if(y<0)*/ z= y*y*y + 1;
}else{ /*if(x<0)*/
  if (y>0)  z=x*x*x*x+y*y*y*y-1;else
  if (y==0) z= x                 else
 /*if(y<0)*/ z= x*x*x + y*y*y ;
}
return(z);
}//F
```

В слове **F** в ДССП-ТВМ будет всегда исполнено 2 команды троичного ветвления. А Си-функции **FC** может потребоваться исполнить от 2-х до 4-х команд двоичного ветвления.

Наконец, заметим, что при любом сколь угодно большом количестве вариантов P продолжения программы анализ, по какому варианту её следует продолжать, будет производиться быстрее с помощью команд троичного ветвления, т.к. таких команд потребуется выполнить значительно меньше. Ведь количество требуемых троичных команд ветвления определяется величиной $\log_3 P$ (округлённой до ближайшего большего целого), а количество двоичных команд ветвления исчисляется величиной $\log_2 P$, а для любого целого P>1 всегда справедливо соотношение: $\log_3 P < \log_2 P$.

5.2. Цикл с троичным условием

Представляемый здесь цикл с троичным условием является принципиально новым видом цикла. Такой вид цикла совмещает в себе проверку, требуется ли прекращать его повторение, с выбором одного их двух действий, предусмотренных для исполнения в качестве тела цикла на очередном его шаге.

Такой вид цикл не встречался ранее в системе команд какой-либо вычислительной машины (ни двоичной, ни троичной). Не было такой команды цикла и на троичной ЦМ «Сетунь-70», в системе команд которой впервые (в 1975 г.) появились управляющие команды, эквивалентные основным конструкциям структурированного программирования.

Вместе с командой обычного вызова процедуры **<SR P>** в системе команд ЦМ «Сетунь-70» появились команда **<BR P1,P2,P3>** для условного вызова одной из процедур и команда **<DW P>** повторяемого вызова процедуры, действующая как цикл с предусловием (**while do**), которые и обеспечили возможность структурированного программирования на уровне машинного кода. Но команда цикла **DW** в ЦМ «Сетунь-70» анализировала вершину стека как двоичное условие (равно нулю или нет) и вызывала в качестве тела цикла всегда одну и ту же процедуру. А предлагаемая команда нового вида цикла анализирует вершину стека уже как троичное условие.

В широко распространённых языках программирования цикл с троичным условием тоже не встречался ранее. Можно сказать, что практическую полезность такого вида цикла неявно отметил когда-то Эдсгер Дейкстра. В своей знаменитой книге «Дисциплина программирования» [10] он предложил конструкцию цикла с множественными условиями и телами, применяя которую продемонстрировал [10, с.70], в частности, как можно упростить известный алгоритма Евклида для вычисления наибольшего общего делителя (НОД) двух натуральных чисел:

НОД(X,Y) на основе цикла с двоичным условием и ветвлением

```
x,y:=X,Y;
do x≠y→if x>y→x:=x-y
         ■ y>x→y:=y-x
       fi
od
печатать(x)
```

НОД(X,Y) на основе цикла с двумя телами

```
x,y:=X,Y;
do x>y→x:=x-y
 ■ y>x→y:=y-x
od
печатать(x)
```

Цикл с троичным условием впервые был предложен в ДССП для троичной машины (ТВМ). И был сначала реализован в ней, как новое слово **DW-+**, определённое сначала на языке ДССП-Т, а также в коде ассемблера [11]. А затем уже (после его апробации в ДССП-ТВМ) было предложено обеспечить функционирование такого цикла соответствующей командой (**DWMP**) самой троичной виртуальной машины.

Объясним подробно правила функционирования (семантику) цикла с троичным условием и познакомимся с соответствующей ему командой **DW−+** языка ДССП-Т (см. блок-схему варианта А на рис. 2). Эта команда проверяет, надо ли завершать цикл, и при этом тут же выбирает одну из двух процедур в качестве тела цикла на случай его продолжения. Такая совмещённая троичная проверка осуществляется в зависимости от знака значения, сформированного в вершине стека. Заметим, что после проверки значения вершины стека само это значение из стека удаляется перед вызовом одной из предусмотренных процедур (или при завершении цикла).

Для обеспечения реализации такой конструкции цикла на двоичной машине потребовалось бы использовать (в сгенерированном коде) две проверки (одну на прекращение цикла, а другую для выбора одного из тел) и соответственно две команды двоичного ветвления (см. блок-схему варианта Б на рис. 2). Поэтому никакого выигрыша в производительности такой вид цикла на двоичной машине не сможет обеспечить. (Правда, такой вид цикла мог бы упростить хотя бы запись алгоритма, освободив его от лишнего ветвления).

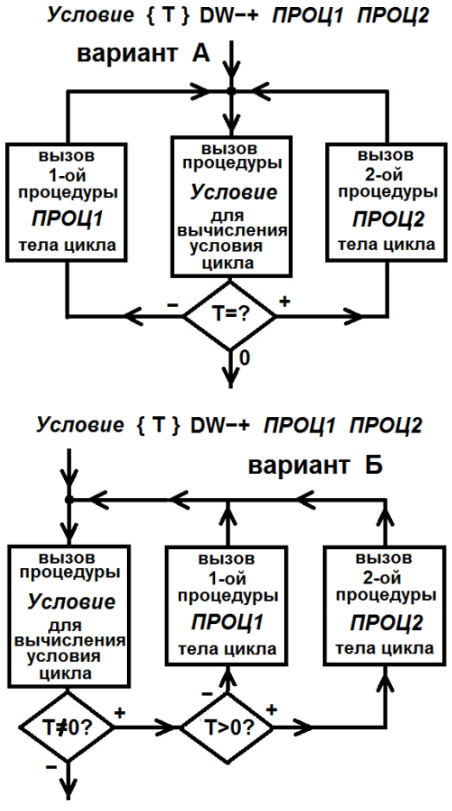

Рис. 2. Блок-схема реализации цикла с троичным условием на троичной (А) и на двоичной машине (Б)

А в троичной машине оценка знака вершины стека вместе с принятием решения о прекращении цикла или его продолжении с выбором соответствующей процедуры на исполнение может выполняться за одну команду. Поэтому на троичной машине такой вид цикла может повысить производительность исполняемой программы.

Продемонстрируем, как применение команды цикла **DW−+** с троичным условием в программе на языке ДССП-Т позволяет не только упростить её алгоритм, но и ускорить её исполнение. В качестве примера рассмотрим задачу вычисления наибольшего общего делителя двух натуральных чисел НОД(X,Y) алгоритмом Евклида. Сначала приведём вариант её решения, в котором применяется команда цикла **DW** с двоичным условием и команда **BR−** условного выбора одной из двух процедур:

```
: NOD1 {x,y} x-y? DW y-x|x-y D ;
: y-x|x-y{x,y} x-y? BR- y-x x-y ;
 : x-y?{x,y} C2 C2 - ;
 : y-x{x,y} C2 - {x,y-x};
 : x-y{x,y} E2 C2 - E2 {x-y,y};
```

А вот вариант решения той же задачи с применением команды троичного цикла:

```
: NOD2 x?y DW-+ y-x x-y  D ;
 : x?y{x,y} C2 C2 CMP ;
 : y-x{x,y} C2 - {x,y-x} ;
 : x-y{x,y} E2 C2 - E2 {x-y,y};
```

Теперь сравним процедуры NOD1 и NOD2. Прежде всего, отметим, что запись алгоритма процедуры NOD2 выглядит заметно проще: в нём стало на одну строку меньше, так как отпала необходимость определения слова `y-x|x-y`, задающего общее тело цикла. Далее сравним их по производительности.

Таблица 1. Сравнение производительностей алгоритмов NOD1(X,Y) и NOD2(X,Y)

аргументы		результат Z=	Количество команд		КВ
X	**Y**	**NOD(X,Y)**	**NOD1**	**NOD2**	
1019	123	1	590	358	**1.65**
1020	3560	20	544	312	**1.74**
2015	135	5	574	350	**1.64**
39366	99	9	8054	4838	**1.66**
5000	250	250	398	246	**1.62**
7777	49	1	3254	1958	**1.66**
КВ– коэффициент выигрыша, который обеспечивает алгоритм **NOD2** для троичной машины по отношению к алгоритму **NOD1** для двоичной машины					

Процедуры вычисления НОД обоих этих вариантов (NOD1 и NOD2) многократно запускались на программном имитаторе троичной машины (ТВМ) с различными аргументами. Для оценки их производительности измерялись по-

казания счётчика исполненных команд ТВМ до и после исполнения каждой процедуры. Результаты проведённых испытаний представлены в таблице 1.

Как видно из этой таблицы, процедура **NOD2**, применяющая цикл с троичным условием, работает значительно быстрее (более, чем в полтора раза), так как исполняется за меньшее число команд ТВМ.

Таким образом, реализация подобных процедур на троичной машине с применением троичного цикла может принести существенный выигрыш в производительности.

6. Способы реализации функций троичной логики

Да, троичный компьютер открывает разработчикам мир новых возможностей. Но при создании программного и аппаратного оснащения, задающего алгоритмы поведения троичной машины, разработчикам требуется также решать и новые задачи. Причём, некоторые из известных задач, решённые когда-то для двоичных машин, для троичной машины приходится решать почти что заново. Одной из таких является задача построения формулы вычисления функции троичной логики по заданной для неё таблице значений.

В двоичной логике (булевой алгебре) широко известны как минимум два конструктивных метода получения формулы вычисления функции по заданной для неё таблице истинности: СДНФ (совершенная дизъюнктивная нормальная форма) и СКНФ (совершенная конъюнктивная нормальная форма). Для получения формулы функции можно также применять универсальный метод так называемых «полиномов Жегалкина», который пригоден для любой k-значной логики. Но он неудобен тем, что для каждой новой функции вынуждает снова решать систему линейных уравнений, чтобы определить коэффициенты того конкретного полинома, который нужно получить для вычисления заданной функции.

Столкнувшись с необходимостью получения формул для некоторых функций троичной логики, автор сначала попытался отыскать (на просторах Интернета и в доступной ему печатной литературе) некоторые аналоги методов СДНФ и СКНФ применительно к троичной симметричной логике (+1,0,-1). А своевременно не найдя их, сам «изобрёл» подобные им методы получения требуемой формулы, объяснению которых и посвящён данный раздел.

Поясним суть одного из возможных предлагаемых методов, ограничив задачу получением формулы вычисления лишь для двуместной троичной функции. (Заметим, что в троичной логике количество различных функций от двух переменных исчисляется величиной 3^9=**19683**, в то время как в двоичной логике таких двуместных функций всего 2^4=**16**).

Итак, пусть троичная функция F(x,y) от двух переменных заданна таблицей своих значений TF на всевозможных наборах пар (x_i,y_j):

TF={ $f_{ij}=F(x_i,y_j)$, i=-1,0,+1; j=-1,0,+1 }.

Формулу вычисления этой функции представим в виде троичной суммы, каждый из слагаемых которой представляется характеристической функцией Pk(x,y), принимающей значение 1 только на одном выделенном наборе (x_{Ik},y_{Jk}) и 0 на остальных, или функцией Nm(x,y), принимающей значение -1 только на одном выделенном наборе (x_{Im},y_{Jm}) и 0 на остальных:

$$\Sigma^{t}_{k} Pk(x,y) +^{t} \Sigma^{t}_{k} Nm(x,y)$$

В изображении этой формулы используются следующие обозначения: $+^t$ – это операция троичного сложения, а Σ^t_k – это операция троичного сложения всех операндов, образуемых перебором всевозможных значений k.

В правую часть этой суммы включаются функции вида Pk(x,y) для тех наборов, на которых заданная функция F принимает значения +1, а в левую часть включаются функции вида Nm(x,y) для тех наборов, где заданная функция F принимает значения -1.

А формулы самих характеристических функций Pk(x,y) и Nm(x,y) представим в виде троичного произведения двух одноместных характеристических функций:

$$Pk(x,y)= H_a(x) \times^t H_b(y) \text{ (где } a=I_k,\ b=J_k),$$

$$Nm(x,y)= \sim^t H_a(x) \times^t H_b(y) \text{ (где } a=I_m,\ b=J_m),$$

используя операции троичного умножения ($\times^t$) и троичной инверсии ($\sim^t$).

Каждая из трёх одноместных характеристических функций (H_{+1}, H_0, H_{-1}) призвана принимать значение +1 на выделенном входном значении и 0 на остальных. Для их вычисления можно предложить следующие формулы:

$$H_{+1}(x)= \sim^t x \times^t (x +^t 1),\ H_0(x)= (x +^t 1) \times^t (1 -^t x),$$

$$H_{-1}(x)= x \times^t (1 -^t x),$$

учитывая, что операцию троичного вычитания $-^t$ можно реализовать с использованием операций троичного сложения и инверсии (смены знака): $a -^t b= a +^t (\sim^t b)$.

В результате получим формулу, вычисляющую значение троичной функции в виде многочлена на основе троичных операций сложения $+^t$, умножения $\times^t$, и инверсии $\sim^t$ (смены знака). Полученную таким способом формулу далее можно упростить, используя правила эквивалентных преобразований для троичной симметричной логики.

Продемонстрируем описанный метод на примере получения формулы вычисления двухместной троичной функции, заданной следующей таблицей значений (см. таблицу 2). На просторах Интернета она упоминается под названием «функция следования Брусенцова» (см., например, http://phg.su/basis2/X19.HTM).

Таблица 2. Таблица значений двухместной троичной функции следования Брусенцова.

аргументы	**X**	**−1**	**−1**	**−1**	**0**	**0**	**0**	**+1**	**+1**	**+1**
	Y	**−1**	**0**	**+1**	**−1**	**0**	**+1**	**−1**	**0**	**+1**
результат	**Z**	**+1**	**0**	**−1**	**0**	**0**	**0**	**0**	**0**	**+1**

Соберём требуемую формулу поэтапно. В левой части троичной суммы поставим в качестве слагаемых двуместные характеристические функции, соответствующие наборам аргументов (−1, −1) и (+1, +1), на которых функция принимает значение +1. В правой части суммы поставим характеристическую функцию, соответствующую набору (−1,+1), на котором функция принимает значение −1. А для наборов, на которых функция принимает значение 0, в итоговую сумму никаких слагаемых не ставим вообще. И представим каждое их 3-х слагаемых в виде произведения двух одноместных характеристических функций.

В результате для нашей функции получим следующую первоначальную формулу:

$$\mathbf{tSLBr1(x,y) = (H_{-1}(x) \times^t H_{-1}(y) +^t +^t H_{+1}(x) \times^t H_{+1}(y)) +^t (\sim^t H_{-1}(x) \times^t H_{+1}(y))}$$

По ней уже можно вычислять требуемую функцию:

```
: tSLBr1 {x,y} {1-й вариант}
  C2 tH- C2 tH- TMUL {x,y,x(x-1)*y(y-1)}
  C3 tH+ C3 tH+ TMUL + E3
  {x(x-1)*y(y-1) + x(x-1)*y(y-1), y,x }
     tH- E2 tH+ TMUL tSUB ;
: tH-{x} { x*(1-x) = if x=-1 then +1 else 0 }
  C Mask--- TADD TMUL NEG ;
: tH+{x} { -x*(x+1) = if x=1 then +1 else 0 }
  Mask+++ C2 TADD TMUL NEG;
: tSUB {x,y} NEG TADD {x-y} ;
```

используя определения слов **tH-** и **tH+** для вспомогательных функций H_{-1} и H_{+1}. Но этот вариант её вычисления не будет эффективным.

В первоначально полученную формулу подставим формулы для вычислений H_{-1} и H_{+1} и выполним эквивалентные преобразования:

$$x\times^t(1-^t x)\times^t y\times^t (1-^t y) +^t \sim^t x\times^t (x+^t 1)$$
$$\times^t\sim^t y\times^t (y+^t 1) -^t x\times^t (1-^t x)\times^t\sim^t y\times^t (y+^t 1) \approx$$
$$\approx x\times^t y\times^t ((x-^t 1)\times^t (y-^t 1) +^t (x+^t 1)\times^t$$
$$(y+^t 1) -^t (x-^t 1)\times^t (y+^t 1)) \approx$$
$$\approx x\times^t y\times^t ((x-^t 1)\times^t ((y-^t 1) -^t (y+^t 1))$$
$$+^t (x+^t 1)\times^t (y+^t 1)) \approx$$
$$\approx x\times^t y\times^t ((x-^t 1)\times^t ((-1 -^t -1))$$
$$+^t x\times^t y+^t x+^t y+^t 1) \approx$$
$$\approx x\times^t y\times^t (x-^t 1 +^t x\times^t y+^t x+^t y+^t 1) \approx$$
$$\approx x\times^t y\times^t (x\times^t y-^t x+^t y)$$
(т.к. $-1-^t 1=+1$ и $x+^t x=\sim^t x$)

Запишем итоговую полученную формулу:

$$\mathbf{tSLBr2(x,y)= x\times^t y\times^t (x\times^t y-^t x+^t y)}$$

И определим на языке ДССП-Т слово, которое будет вычислять троичную функцию следования Брусенцова по этой формуле:

```
: tSLBr2 {2-й вариант}
 {x,y} C2 C2 TMUL C E4
 {x*y,y,x*y,x} NEG TADD
 {x*y,y,x*y-x} TADD TMUL
 {x*y*(y+x*y-x)} ;
```

Раскрыв скобки и обозначив $\mathbf{x\times^t x=|x|}$, можем получить ещё один вариант формулы вычисления функции следования Брусенцова:

$$\mathbf{tSLBr3(x,y)= |x\times^t y| -^t |x|\times^t y +^t x\times^t |y|}$$

Но этот 3-й вариант будет уступать предыдущему (2-му) по количеству команд:

```
: tSLBr3 {3-й вариант}
  C2 C2 TMUL tABS {x,y,|x*y|}
  C3 tABS C3 TMUL tSUB E3
  {|x*y|-|x|*y,y,x} E2 tABS TMUL TADD
  {|xy|-|x|*y+x*|y| } ;
: tABS {x} C TMUL { |x|= x*x } ;
```

Так что самым оптимальным (по числу команд) будет именно 2-ой вариант вычисления функции, определённый словом **tSLBr2**.

Описанный способ получения формулы вычисления троичной функции по заданной таблице её значений на всевозможных наборах аргументов можно распространить и на функции от многих переменных. Например, для случая трёх переменных надо будет формулы функций Pk(x,y,z) и Nm(x,y,z) представлять в виде произведения не двух, а уже трёх одноместных характеристических функций:

$Pk(x,y,z)=H_a(x)\times^t H_b(y)\times^t H_c(z)$, где $a=I_k, b=J_k, c=L_k$

$Nm(x,y,z)=\sim^t H_a(x)\times^t H_b(y)\times^t H_c(z)$, $a=I_m, b=J_m, c=J_m$

7. Заключение

В статье охарактеризованы специфические особенности программирования, которые приходится учитывать при создании программного оснащения для троичного компьютера. На примерах ряда характерных программ, разработанных в ДССП для троичной виртуальной машины, продемонстрированы новые возможности, которые может предоставить программисту троичная цифровая машина. А также отмечено, что на определённом классе задач троичный компьютер по ряду параметров может превосходить двоичный. И это даёт основание полагать, что с созданием аппаратной реализации троичной цифровой машины человечество обретёт более совершенный инструмент обработки информации.

Features of programming of ternary computer: new capabilities and new tasks

A.A.Burtsev

Abstract. The article considers new expressive facilities of constructing programs, which can be provided to the programmer by the ternary computer and which are provided now by the programming system DSSP for the ternary virtual machine (TVM). It also demonstrates a method of constructing a formula for calculating a ternary logic function according to a given table of values with its subsequent implementation in DSSP-TVM.

Keywords: ternary computer, ternary arithmetics, ternary logic, ternary function, "Setun", "Setun-70", ternary virtual machine (TVM), DSSP, DSSP-TVM.

Литература

1. Д. Кнут. Искусство программирования на ЭВМ. М., Мир, 1978, Т.2, с. 216-219.

2. Н.П. Брусенцов, Х. Рамиль Альварес. Троичные ЭВМ “Сетунь” и “Сетунь 70”. «Первая Международная конференция "Развитие вычислительной техники в России и странах бывшего СССР: история и перспективы" SORUCOM-2006». Россия, г. Петрозаводск, 3-7 июля 2006 г. (труды конференции в 2-х ч.). Изд-во ПетрГУ, 2006. ч.1, 45-51.

3. С.А. Сидоров, Ю.С. Владимирова. Троичная виртуальная машина. «Программные системы и инструменты. Тематический сборник», №12 (2011), 46-55.

4. А.А. Бурцев, Х. Рамиль Альварес. Кросс-система разработки программ на языке ДССП для троичной виртуальной машины. «Программные системы и инструменты. Тематический сборник», №12 (2011), 183-193.

5. С.П. Маслов. Троичная схемотехника. «Программные системы и инструменты. Тематический сборник», №13 (2012), 152-158.

6. А. А. Бурцев, С. А. Сидоров. Троичная виртуальная машина и троичная ДССП. «Программные системы: теория и приложения», Т.6 (2015), №4, 29–97, http://psta.psiras.ru/read/psta2015_4_29-97.pdf.

7. А.А. Бурцев, С.А. Сидоров. Программный комплекс ДССП-ТВМ для структурированного программирования троичной [виртуальной] машины. «Современные информационные технологии и ИТ-образование», Т. 2 (2015), № 11, 371–379.

8. Ю.С. Владимирова. Введение в троичную информатику: учебное пособие. М., АРГАМАК-МЕДИА, 2015.

9. Н.П. Брусенцов, Х. Рамиль Альварес. Структурированное программирование на малой цифровой машине. «Вычислительная техника и вопросы кибернетики», Вып. 15. М., Изд-во МГУ, 1978, 3-8.

10. Э. Дейкстра. Дисциплина программирования. М., Мир, 1978.

11. А.А. Бурцев. Разработка собственных управляющих конструкций в среде ДССП для троичной машины. «Труды НИИСИ РАН», Т. 8 (2018), № 2, 52–64.