УДК 004.4'423

Бурцев А.А. [1,2]

[1] Федеральный Научный Центр НИИ Системных Исследований (ФГУ ФНЦ НИИСИ РАН), г. Москва, Россия
[2] Филиал Московского государственного технического университета им. Н.Э. Баумана, г. Калуга, Россия

ОСОБЕННОСТИ ПРОГРАММИРОВАНИЯ В ДССП ДЛЯ ТРОИЧНОЙ МАШИНЫ

Аннотация

В статье подчёркиваются характерные свойства Диалоговой системы структурированного программирования (ДССП), которыми она существенно отличается от традиционных языков (Паскаль, Си), обычно используемых для освоения базового курса программирования. А также рассматриваются новые возможности построения программ, которые могут быть эффективно реализованы на троичном компьютере и которые в настоящее время обеспечивает система программирования ДССП для ТВМ — троичной виртуальной машины.

Ключевые слова

Троичная симметричная система счисления, троичная арифметика, троичная логика, троичный компьютер, троичная виртуальная машина (ТВМ), диалоговая система структурированного программирования (ДССП), ДССП-ТВМ.

Burtsev A.A. [1,2]

[1] Federal Scientific Center Scientific Research Institute for System Analysis of the Russian Academy of Sciences (FSC SRISA RAS), Moscow, Russia
[2] Kaluga branch of Bauman Moscow State Technical University, Kaluga, Russia

FEATURES OF PROGRAMMING IN DSSP FOR THE TERNARY MACHINE

Abstract

In article characteristic properties of the Dialogue System for Structured Programming (DSSP) in which it significantly differs from the traditional languages (Pascal, C) which are usually used for development of a basic course of programming are emphasized. And also the new possibilities of program creation which can be effectively realized on the ternary computer and which are provided now by programming system DSSP for TVM — the ternary virtual machine are considered.

Keywords

Ternary symmetric numeration system, ternary arithmetics, ternary logic, ternary computer, ternary virtual machine (TVM), dialogue system for the structured programming (DSSP), DSSP-TVM.

Введение

В НИЛ троичной информатики ВМК МГУ в 2010-2013 гг. создан программный комплекс ДССП-ТВМ, включающий имитатор троичной машины (ТВМ) и систему разработки программ для неё на языке ДССП-Т – троичном варианте языка ДССП [1,2]. В настоящее время ДССП-ТВМ может использоваться не только в качестве среды опережающей подготовки и отладки программного оснащения для создаваемой «в кремнии» (т.е. аппаратно реализованной) троичной машины. Она может применяться ещё и в качестве учебной среды программирования при подготовке квалифицированных специалистов в области троичной информатики [3].

Для успешного освоения «искусства программирования» в ДССП необходимо учитывать ряд её характерных особенностей. Словарная организация, арифметический стек для обработки данных, процедурная версия структурированных команд управления и управляющий стек для адресов возвратов из процедур, а также диалоговый режим — эти специфические черты существенно отличают ДССП от традиционных языков программирования (Паскаль, Си), обычно используемых для обучения программированию. В ДССП для троичной машины (ТВМ) к ним прибавляются ещё и другие особенности программирования, с которыми приходится сталкиваться при переходе от двоичной машины к троичной.

Прежде всего, необходимо учитывать, что некоторые привычные операции (сдвига, деления, поразрядного сложения, умножения и т.д.) в троичной машине действуют несколько иначе. А значит, потребуется не только учесть специфику представления чисел в троичном симметричном коде, но и приспособиться обрабатывать триты и трайты вместо битов и байтов. Переход к обработке данных на троичной машине позволяет реально ощутить некоторые хорошо известные преимущества применения троичного симметричного кодирования [3,с.7]: естественное представление положительных и отрицательных чисел, единство операций сдвига, трёхзначность операций сравнения и функции «знак числа», простота округления числа путём отбрасывания младших разрядов его представления в троичном коде.

При таком переходе можно не только убедиться, что всё то, что удавалось ранее сделать на двоичной

машине, можно легко сделать и на троичной машине (правда, в чём-то немного по-другому). Но и осознать, что троичная машина предоставляет для построения компьютерной программы гораздо более широкие возможности. В их числе следует отметить возможность эффективной реализации троичного ветвления, цикла с троичным условием, операций троичной арифметики и троичной логики, а также операций над троичными множествами. Далее в статье подробно рассматривается, как эти новые возможности троичной машины можно использовать при создании программ в ДССП-ТВМ.

Словарная организация и диалоговый режим

ДССП построена на основе словарной организации аналогично языку (системе программирования) ФОРТ (FORTH) [4]. Всякая операция (процедура), поддерживаемая системой, обозначается отдельным словом (слово – это любая последовательность печатных символов, отличных от пробела). Совокупность слов, понимаемых системой, и составляет ее язык (словарь). Любое действие, совершаемое системой, вызывается словом. Предусмотренные в системе слова исполняют самые разнообразные функции. Они выполняют вычисления или обработку данных в стеке, доступ к памяти или к переменным, служат для организации ветвлений и циклов, обеспечивают построение тел для новых слов, вносимых в словарь, управляют состоянием словаря и даже самим процессом выполнения задания в системе.

Система расширяется путем определения новых слов, которые сразу же после добавления в словарь могут быть использованы в системе. Для определений новых слов используются особые слова-компиляторы. Самое простое из них служит для определения нового слова, как процедуры:

```
: P P1 P2 ... Pn ; { комментарий: определение процедуры с именем P }
```

Причём в отличие от языка ФОРТ ДССП допускает, что в этой последовательности слов P1 P2 **...** Pn могут встречаться в том числе и такие слова, которые еще не были определены и предполагается, что будут определены впоследствии. Именно благодаря такой своей особенности, ДССП позволяет составлять программу методом нисходящего анализа: сначала определить главное слово, затем слова следующего уровня, из которых оно определяется, и так постепенно спуститься до детального определения самых простых слов.

Вследствие такой словарной организации программирование в ДССП заключается просто в наполнении её словаря новыми словами. И путём наращивания словаря каждый пользователь может создать свой вариант расширения ДССП, отвечающим нуждам его прикладной области.

ДССП, как правило, обеспечивает диалоговый характер взаимодействия с пользователем на всех стадиях разработки программы: начиная от её редактирования и вплоть до её окончательной отладки. Такой диалог обеспечивается интерпретатором ДССП, функционирующим в режиме исполнения потока слов или предложений, образуемых последовательностями слов-команд, задаваемых пользователем.

Арифметический стек и постфиксная запись выражений

ДССП характеризуется двухстековой архитектурой: управляющий стек служит для сохранения адресов возвратов из процедур, а арифметический стек используется для выполнения операций над данными. Использование стека при обработке данных приводит к компактной постфиксной записи арифметических и логических выражений, которая получила название ПОЛИЗ (польско-инверсная запись).

Например, для вычисления в вершине стека значения по формуле: (a+b)×(c−d) в ДССП придётся записать такую последовательность слов:

```
a b + c d – * { ≈ (a+b)*(c-d) }
```

А для вычисления логического выражения, представляющего условие двоичной проверки (да/нет) попадания точки **x** в интервал (**a**,**b**): **a<x<b**, на языке ДССП надо записать такую последовательность слов:

```
x a > x b < AND { ≈ (x>a)and(x<b) }
```

Управляющие конструкции для процедурного структурированного программирования

В ДССП каждая из команд ветвлений вместо перехода по телу кода (как это обычно делается в традиционных языках программирования, а также и в системе ФОРТ) по сути осуществляет вызов процедуры слова, выбранного по условию, а каждая команда цикла организует многократный вызов процедуры слова, пока такой вызов допускается условием цикла. В результате поддержка структурированного программирования в ДССП осуществляется непосредственно на уровне процедурного сшитого кода, используемого для внутреннего представления ДССП-программ, аналогично тому, как такая поддержка была обеспечена в своё время в ЦМ “Сетунь-70” [5] на уровне машинных команд.

Для первоначального знакомства с управляющими командами ДССП и демонстрации структуры создаваемой в ДССП программы приведём «хрестоматийный» пример подпрограммы вычисления наибольшего общего делителя двух натуральных чисел НОД(X,Y):

подпрограмма-функция NOD(X,Y) на языке Паскаль	процедура на языке ДССП Z=NOD(X,Y)

function NOD(x,y: integer):integer;
begin
while x<>y **do**
if x>y **then** x:=x-y **else** y:=y-x;
NOD:= x {или y} ;
end
	: **NOD**{X,Y} x<>y? **DW** x-y|y-x D {Z};
	: x<>y? {x,y} C2 C2 <> ;
	: x-y|y-x {x,y} x>y? **BR+** x-y y-x ;
	: x>y? {x,y} C2 C2 > ;
	: x-y {x,y} E2 C2 - E2 {x-y,y} ;
	: y-x {x,y} C2 – {x,y-x} ;

Специфика арифметических вычислений в троичной машине

Арифметические операции сложения, вычитания и умножения целых чисел при исполнении на троичной машине дадут тот же результат, что и на двоичной. Только результат этот всегда знаковый, т.к. в симметричной системе счисления отсутствует различие между знаковым и беззнаковым значением. Результат может быть некорректным лишь в случае переполнения 27-тритного машинного слова, т.е. если он по модулю превзойдёт значение $(3^{27}-1)/2$ (заметим, что значение 3^{27}=**7625**597484987 намного превышает 2^{32}=**4**294967296).

А вот операция деления целых с получением частного и остатка в троичной симметричной системе счисления действует немного иначе: она производит минимальный по модулю остаток, который может быть не только положительным, но и отрицательным. Так, при обычном целочисленном делении 20 на 3 получится частное 6 и остаток 2, а целочисленное деление в симметричной системе счисления даст другой результат: частное 7 и остаток −1 (см.[3,с.38-39]).

В двоичной машине операции умножения и деления на 2 (или 2^n) можно реализовать одной командой сдвига влево или вправо соответственно на 1 разряд (или на n разрядов). В троичной машине командами сдвига осуществляется умножение или деление числа на 3 (или 3^n). Причём можно обойтись единой командой сдвига, т.к. не требуется различать разные виды сдвигов (арифметический или логический), как в двоичной машине.

Специфика доступа к трайтам и троичным словам памяти

Производя обработку данных, расположенных в памяти троичной машины (ТВМ) по заданным адресам (указателям), требуется учитывать, что память адресуется с точностью до трайта, адрес начала троичного слова указывает на его младший трайт (из 3-х), а для перехода к следующему троичному слову к значению указателя надо прибавлять 3 (а не 4, как это делается на двоичной машине, где 32-разрядное слово состоит их 4-х байтов). В ТВМ (и языке ДССП-Т) предусмотрены команды (слова в ДССП-Т) для взятия из памяти в вершину стека 9-тритного значения трайта (**@T**), 27-тритного троичного слова из 3-х трайтов (**@W**), а также 2-х трайтного 18-тритного значения (**@TT**). А также предусмотрены аналогичные команды для засылки в память трайта (**!T**), троичного слова (**!W**) и 2-х трайтного значения (**!TT**) из стека (точнее из подвершины стека по адресу, указанному в вершине).

Рассмотрим в качестве примера обработки троичных слов памяти процедуру циклического сдвига на 1 позицию влево (к началу) вектора (одномерного массива) 27-тритных значений, параметры которого (адрес начала и количество элементов) передаются в стеке (в подвершине и вершине):

процедура на языке Паскаль	процедура на языке ДССП

const kMax= 1000;
type Vctr=array[0..kMax]of Word;
procedure L1ShiftVctr
(**var** A:Vctr; k:integer);
var n,j: integer; T:Word;
begin
T:=A[0]; n:=k-1;
for j:= 0 to n-1 0 **do**
A[j]:=A[j+1];
A[n]:=T;
end
	{ { A - адрес вектора, k= кол-во элементов, n=k-1
	{ { вектор A[0:n]=[A0,A1,..,An] => [A1,..,An,A0]
	: L1ShiftVctr {A,k} C2 **@W** {A0=A[0]} C3 C3 1-
	{A,k,A0,'A[0],n} **DO- LShift_A[j]** {i=n-1,..,1,0}
	{A,k,A0,'A[n]} **!W** { A[n]:= A0 } { A,k } DD { } **;**
	: LShift_A[j] {~,'A[j],i } {i=n-1,..,1,0; j=0,1,..,n-1}
	{ ~,'A[j],i } C2 **3+** C **@W**
	{ ~,'A[j],i,'A[j+1], A[j+1] } E2 E4
	{ ~,'A[j+1],i, A[j+1],'A[j] } **!W** { A[j]:=A[j+1] }
	{ ~,'A[j+1],i } **;**

Поразрядная (потритная) обработка троичного слова

Если возникает необходимость работать со значениями, которые располагаются внутри одного машинного слова в его отдельных разрядах (флаги) или занимают поля из нескольких смежных его разрядов, то обычно для этой цели переходят на прямое использование ассемблера или используют специальные возможности, которые предоставляют некоторые высокоуровневые языки программирования для обработки отдельных битов или битовых полей машинного слова, в котором представлены обрабатываемые величины. При программировании на троичной машине вместо обработки битов и битовых полей приходится иметь дело с тритами и образуемыми ими полями из рядом расположенных тритов.

Для потритной обработки данных внутри трайта или троичного машинного слова троичная машина (ТВМ) предоставляет базовый набор команд (слов-операций в ДССП) для поразрядного сложения (**TADD**), умножения (**TMUL**), инверсии или смены знака (**NEG**), сдвига тритов слова влево (**SHL**) или вправо (**SHR**) на одну или даже на несколько позиций (**SHT**), а также выполнения операций поразрядного вычисления минимума (**TMIN**) и максимума (**TMAX**), которые могут применяться в качестве троичных логических операций, являющихся троичными аналогами операций конъюнкции (**and**) и дизъюнкции (**or**) булевой алгебры.

Продемонстрируем на конкретных примерах приёмы использования этих троичных операций для выполнения характерных действий с отдельными полями троичного машинного слова. В первом примере продемонстрируем, как извлечь из троичного слова значение, содержащееся в нескольких его смежных разрядах. Допустим, в заданном 27-разрядном троичном машинном слове W (или 32-разрядном двоичном) в разрядах с 7-го по 11-ый содержится некоторое значение Z, которое требуется считать оттуда как 5-разрядное

целое число (договоримся разряды машинного слова нумеровать от 0 до 26, начиная с младшего разряда). Приведём вариант программной реализации такой задачи для троичной машины (на языке ДССП-Т) и сопоставим его с вариантом реализации аналогичной задачи для двоичной машины (на языке Си):

выражение на языке Си	предложение на языке ДССП-Т
`/* Z= */ (W>>7)&(0x1F)`	`{W} -7 SHT #144 TMUL {Z}`

Заметим, что здесь для обозначения константного значения, состоящего из пяти единиц в младших разрядах и нулей в остальных разрядах (т.е. двоичного значения 00011111b и троичного .0000+++++.), используются литерные константы `0x1F` и `#144`, представляющие требуемые значения в 16-ной и 9-ной симметричной системах счисления соответственно.

Другой пример демонстрирует решение «обратной» задачи, в которой требуется 5-разрядное значение Z записать в разряды с 7 по 11-ый 27-разрядного троичного (или 32-разрядного двоичного) машинного слова W:

выражение на языке Си	предложение на языке ДССП-Т
`/* new W= */` `W&(0xFFFFF07F)|((Z&0x1F)<<7))`	`{W,Z} #144 TMUL +7 SHT E2` `#14444444001444 TMUL TADD {new W}`

Заметим, что здесь 16-ная константа `0xFFFFF07F` и 9-ная константа `#14444444001444` задают значение (32-битное двоичное или 27-тритное троичное), состоящее из пяти нулей в разрядах с 7-го по 11-й и единиц в остальных разрядах.

Троичное ветвление

Троичное ветвление предполагает выбор дальнейшего пути хода исполнения программы не из двух, а из трёх возможных вариантов, предусмотренных для её продолжения. В ДССП для ТВМ троичное ветвление организуется (см. блок-схему варианта А на рис. 1) с помощью команды (**BRS**) выбора на исполнение одной из трёх предусмотренных процедур в зависимости от знака значения, сформированного в вершине стека. Существенно, что в троичной машине оценка знака вершины стека и выбор соответствующей процедуры на исполнение может выполняться за одну команду. Такая команда была предусмотрена ещё в архитектуре экспериментальной ЦМ «Сетунь-70» [5], опытный образец которой реально функционировал в НИЛ ЭВМ на факультете ВМК МГУ с 1970 по 1987 гг.

В языках программирования для двоичных машин явный оператор для организации троичного ветвления редко предусматривается (исключение, пожалуй, составляет лишь язык Фортран), поскольку для его реализации на двоичной машине всё равно потребуется использовать (в сгенерированном коде) две проверки и соответственно две команды двоичного ветвления (см. блок-схему варианта Б на рис. 1). Поэтому в случае необходимости организации в программе троичного ветвления программисту просто предлагается самому воспользоваться оператором двоичного ветвления 2 раза.

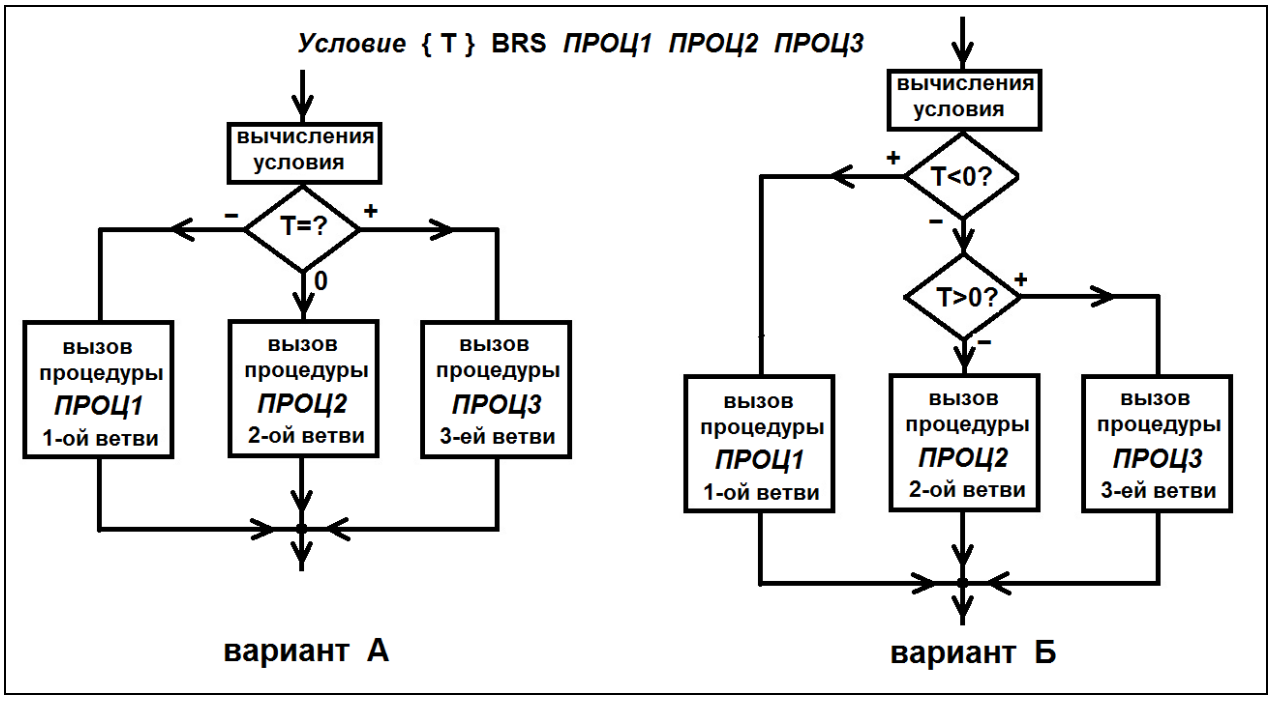

Рис. 1. Блок-схема выполнения троичного ветвления на троичной (А) и на двоичной машине (Б).

Классическим примером применения троичного ветвления является алгоритм определения знака числа;

во многих языках программирования по такому алгоритму как раз и реализуется стандартная функция sign:

задача	алгоритм (на Паскале)	на языке ДССП-Т

$z=\begin{cases}-1, x<0\\ 0, x=0\\ +1, x>0\end{cases}$	`if x<0 then z:=-1 else`
	`if x=0 then z:=0 else`
	`{ if x>0 } z:=+1 ;`
		`: SGN{x} BRS -1 0 +1 {z};`

Похожий алгоритм можно применять и при троичном варианте сравнения двух чисел:

задача	алгоритм (на Паскале)	на языке ДССП-Т

$z=\begin{cases}-1, x<y\\ 0, x=y\\ +1, x>y\end{cases}$	`if x<y then z:=-1 else`
	`if x=y then z:=0 else`
	`{ if x>y } z:=+1 ;`
		`: CMP{x,y} - {x-y}`
		`BRS -1 0 +1 {z};`

Заметим, что слова-операции **SGN** и **CMP** в ДССП для ТВМ реализуются напрямую соответствующими командами (**SIGN** и **CMP**) троичной машины (ТВМ), а их возможные реализации приведёны здесь лишь в качестве примеров использования команды **BRS**.

Сами же эти слова могут использоваться для формирования троичного условия, которое будет оцениваться последующей командой троичного ветвления **BRS**. Например, для трёхвариантного вычисления функции z= F(x,y) :

задача	алгоритм (на Паскале)	на языке ДССП-Т

$z=\begin{cases}x+y, x<y\\ x, x=y\\ x-y, x>y\end{cases}$	`if x<y then z:=x+y else`
	`if x=y then z:=x else`
	`{ if x>y } z:=x-y ;`
		`: F{x,y} C2 C2 CMP`
		`{x,y,x?y} BRS + D -`
		`{ z = x+y | x | x-y } ;`

Троичное условие и троичное логическое выражение

Предусмотренные в ДССП (и в ТВМ) команды (**SGN** и **CMP**), выдающие троичное значение (+1| 0 |−1) в качестве результата оценки знака числа или сравнения двух чисел, служат основой для вычисления логических условий, принимающих три возможных значения: +1, −1, 0, которые в троичной логике можно трактовать как ответы на вопрос: «Да», «Нет», «Не Знаю». Получаемые ими троичные значения можно далее использовать для составления более сложных троичных условий, применяя к ним операции потритной инверсии (**NEG**), потритного минимума (**TMIN**) и потритного максимума (**TMAX**), которые могут служить троичными аналогами операций отрицания, конъюнкции и дизъюнкции булевой алгебры.

Пусть требуется, например, выполнить троичную проверку попадания точки **x** в отрезок [**a**,**b**], результатом которой должно стать одно их трёх значений: 1) +1, если точка **x** находится внутри отрезка (a<x<b); 2) −1, если точка **x** расположена вне отрезка (x<a или x>b); 3) 0, если точка **x** лежит на одной из границ отрезка (x=a или x=b). Продемонстрируем, как такую троичную проверку можно легко выполнить в ДССП для троичной машины и сравним с решением аналогичной задачи на двоичной машине (на Паскале):

задача	алгоритм (на Паскале)	на языке ДССП-Т

$z=\begin{cases}+1, & a<x<b\\ 0, x=a \text{ или } x=b\\ -1, x<a \text{ или } x>b\end{cases}$	`if(a<x)and(x<b)then z:=+1`
	`else`
	`if (x=a)or(x=b) then z:=0`
	`else z:=-1 ;`
		`: TSEG{a,b,x} C E4`
		`{x,b,x,a} CMP {x?a}`
		`E3 {x?a,b,x} CMP {b?x}`
		`{x?a,b?x} TMIN {z} ;`

Заметим, что на языке ДССП-Т для выполнения такой троичной проверки совсем не потребовалось использовать команды ветвления, т.к. задача легко решалась просто составлением троичного логического выражения. Подобные выражения можно составить и для троичной проверки попадания точки с координатами (**x**,**y**) в заданную фигуру на плоскости. Продемонстрируем, как выполнить такие проверки на языке ДССП-Т для двух фигур: 1) круга радиуса **R** с центром в начале координат (фигура A); и 2) прямоугольника шириной **W** и высотой **H**, левый нижний угол которого расположен в начале координат (фигура B):

троичная проверка попадания точки в фигуру A	троичная проверка попадания точки в фигуру B

`: inFigA? {x,y,R}`
`C * E3 C * E2 C * +`
`{R^2,x^2+y^2} CMP { z= +1| 0|-1 }`
	`: inFigB? {x,y,W,H} E4 0 E3 E2`
	`{H,y,0,W,x} TSEG`
	`{ H,y,x?in[0,W]} E3 0 E3`
	`{x?in[0,W],0,H,y} TSEG`
	`{ x?in[0,W],y?in[0,H]} TMIN`
	`{ z=+1|0|-1 } ;`

Используя операции **NEG, TMIN** и **TMAX** можно выразить и более сложные троичные условия попадания точки в объединение (AUB) или пересечение двух фигур (A∩B), а также в фигуру, образованную фигурой A с вырезанной из неё фигурой B (A\B), или в фигуру, образованную фигурой B с вырезанной из неё

фигурой A (B\A):

троичная проверка попадания точки в фигуру AUB	троичная проверка попадания точки в фигуру A∩B

`: inFigA+B? {x,y,R,W,H} E3 C4 6 CT E3`
`{x,y,H,W,x,y,R} inFigA? 5 ET`
`{(x,y)inFigA?,y,H,W,x} E4 E3`
`{(x,y)inFigA?,x,y,W,H} inFigB?`
`{(x,y)inFigA?,(x,y)inFigB?} TMAX`
`{ z= (x,y)inFig(A+B) ? } ;`
	`: inFigA*B? {x,y,R,W,H} E3 C4 6 CT E3`
	`{x,y,H,W,x,y,R} inFigA? 5 ET`
	`{(x,y)inFigA?,y,H,W,x} E4 E3`
	`{(x,y)inFigA?,x,y,W,H} inFigB?`
	`{(x,y)inFigA?,(x,y)inFigB?} TMIN`
	`{ z= (x,y)inFig(A*B) ? } ;`

троичная проверка попадания точки в фигуру A\B	троичная проверка попадания точки в фигуру B\A

`: inFigA-B? {x,y,R,W,H} E3 C4 6 CT E3`
`{x,y,H,W,x,y,R} inFigA? 5 ET`
`{(x,y)inFigA?,y,H,W,x} E4 E3`
`{(x,y)inFigA?,x,y,W,H} inFigB?`
`{(x,y)inFigA?,(x,y)inFigB?} NEG TMIN`
`{ z= (x,y)inFig(A-B) ? } ;`
	`: inFigB-A? {x,y,R,W,H} E3 C4 6 CT E3`
	`{x,y,H,W,x,y,R} inFigA? NEG 5 ET`
	`{-(x,y)inFigA?,y,H,W,x} E4 E3`
	`{-(x,y)inFigA?,x,y,W,H} inFigB?`
	`{-(x,y)inFigA?,(x,y)inFigB?} TMIN`
	`{ z= (x,y)inFig(B-A) ? } ;`

Цикл с троичным условием

Такой цикл совмещает в себе проверку, надо ли прекращать его повторение, с выбором одного их двух действий, предусмотренных для исполнения в качестве тела цикла на очередном его шаге. Возможную полезность такого цикла демонстрировал ещё Эдсгер Дейкстра в своей книге «Дисциплина программирования» [6, с.70]. В ДССП для ТВМ цикл с троичным условием организуется (см. блок-схему варианта А на рис. 2) с помощью команды повторения **DW-+**, которая на очередном шаге проверяет, надо ли завершать цикл, и совмещает эту проверку с выбором на исполнение в качестве тела цикла одной из двух предусмотренных процедур в зависимости от знака значения, сформированного в вершине стека. Существенно, что 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.

В языках программирования для двоичных машин подобная конструкция цикла не даст никакого выигрыша, поскольку для его реализации на двоичной машине всё равно потребуется 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 в случае необходимости организации в программе подобного цикла с троичным условием программисту просто предлагается самому организовать его с помощью обычного цикла с (двоичным) условием и дополнительного условного оператора в теле цикла для выбора одного из действий.

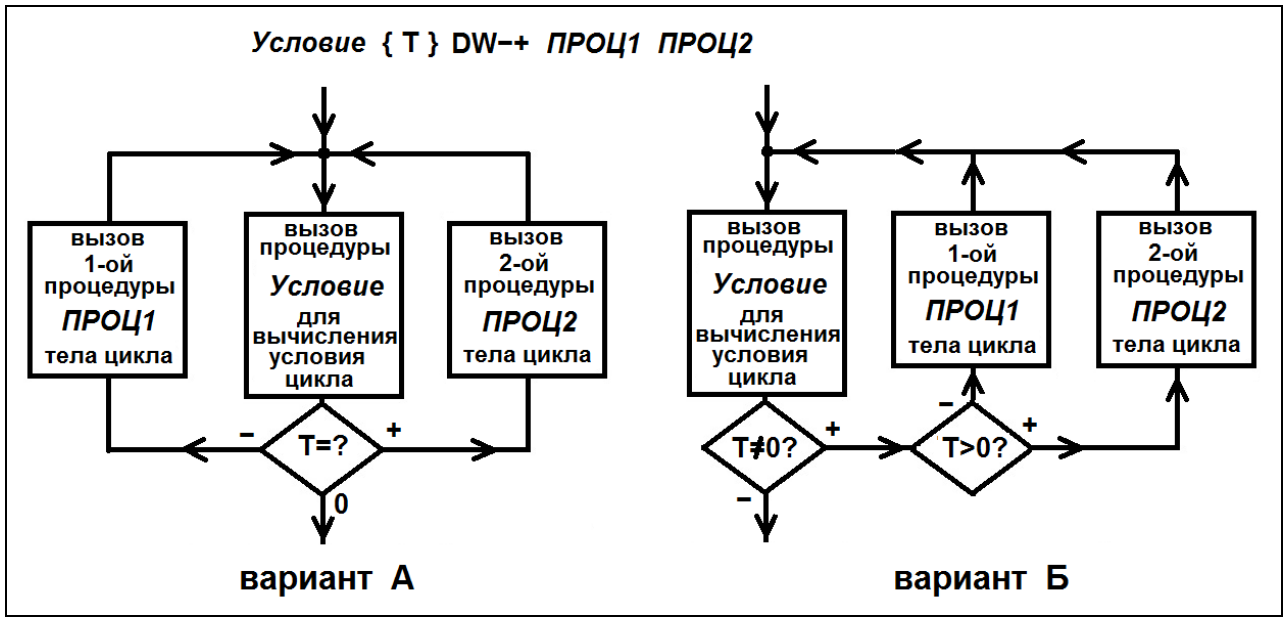

Рис. 2. Блок-схема выполнения цикла с троичным условием на троичной (А) и на двоичной машине (Б).

Продемонстрируем использование цикла с троичным условием в программе на языке ДССП-Т на примере решения рассмотренной ранее задачи вычисления наибольшего общего делителя двух натуральных чисел НОД(X,Y), сравнив его с аналогичным решением, в котором применяется команда цикла **DW** с двоичным условием и команда **BR-** условного выбора одной из двух процедур:

{ только на основе **DW** и **BR-** }	{ на основе **DW-+** }

`: NOD1 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} ;`
	`: 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} ;`

Обе этих процедуры вычисления НОД запускались на программном имитаторе троичной машины (ТВМ) с замером количества исполненных команд. В результате оказалось, что процедура **NOD2**, применяющая цикл с троичным условием, алгоритм которой выглядит проще, ещё и работает быстрее (примерно в полтора раза), так как исполняется за меньшее число команд ТВМ. Это значит, что на троичной машине реализация подобных процедур оказывается более эффективной, чем на двоичной.

Троичное множество

Возможность сохранять в одном трите не двоичное (+1| 0), а троичное значение (+1| 0 |−1) позволяет компактно представлять в троичной машине такие множества, в которых вопрос о вхождении каждого элемента может подразумевать не два (Да, Нет), а три варианта ответа: (Да, Нет, Неизвестно). Образно представим такое троичное множество некоторой замкнутой фигурой на плоскости с определённой границей. Тогда элементу, про который точно известно, что он входит в это множество, должна соответствовать на плоскости точка внутри этой фигуры. Элементу, про который точно известно, что он не входит в это множество, должна быть сопоставлена точка на плоскости вне этой фигуры. А элементу, о котором (на данный момент) нет точной информации, входит он в это множество или нет, можно сопоставить некоторую точку на границе этой фигуры.

Для работы с троичным множеством, число элементов которого не превосходит 27, в языке ДССП-Т можно объявить переменную, занимающую в памяти одно 27-разрядное троичное слово. Пронумеруем (от 0 до 26) все возможные элементы, которые могут входить в такое множество, и сопоставим каждому k-му элементу множества k-ый трит такого слова (k=0,1,..,26). Пустому множеству будет соответствовать 27-разрядное значение, все триты которого установлены в −1, полному множеству будет соответствовать значение, у которго все 27 тритов равны +1. Неопределённому состоянию множества можно сопоставить нулевое значение (все 27 тритов в нём равны 0).

Для заполнения множества потребуется уметь заносить в k-ый трит ставшую нам известной информацию о вхождении k-ого элемента в это множество. Потребуется также операция, позволяющая проверять состояние k-ого трита отведённого множеству троичного слова, чтобы узнать ответ на вопрос, входит ли k-ый элемент в множество (+1), не входит (−1) или про это (на данный момент) ничего неизвестно (0).

Такие операции в ДССП для троичной машины можно реализовать следующим образом:

```
{ взятие значения k-го трита вершины стека }
: T@I {T[27:0],k}
    NEG SHT #1 TMUL {T[k]} ;
{установка значения t для k-го трита вершины стека}
: T!I {T[27:0],t,k}
      E2 #1 TMUL C2 SHT E3
 {t<<k,k,T[27:0]} E2 Mask+++i0
     TMUL TADD {T[27:0]} {T[k]:=t} ;
```

```
{ троичное значение из одних +1 }
#14444444444444 VALUE Mask+++
{ Mi= маска, где M[i]=-1, M[j]=0 для j<>i }
: Mask000i- { i }
     .-. E2 SHT { Mi } ;
{ Mi= маска, где M[i]= 0, M[j]=+1 для j<>i }
: Mask+++i0 {i}
     Mask000i- Mask+++ TADD {Mi} ;
```

Допустим на плоскости имеется 27 точек (Xi,Yi) (i=0,1,..,26), координаты которых записаны в одноимённые вектора X[0:26] и Y[0:26]. И требуется сформировать два троичных множества TSA и TSB, в которых для каждой точки надо отметить, как эта точка расположена на плоскости относительно фигуры A — круга радиуса R=10 с центром в начале координат (в множестве TSA) и фигуры B — прямоугольника шириной **W**=16 и высотой **H**=12, левый нижний угол которого расположен в начале координат (в множестве TSB). Проверим каждую точку на попадание в каждую из фигур и отметим этот факт в соответствующем множестве:

```
{вектора координат точек} 26 VCTR X  26 VCTR Y
{ формирование множества TSA } VAR TSA
    { для каждой i–ой точки i=0,1,..,26 выполним: }
    TSA i X i Y {Xi ,Yi} 10 inFigA? i T!I ! TSA
{ формирование множества TSB } VAR TSB
    { для каждой i–ой точки i=0,1,..,26 выполним: }
    TSB i X i Y {Xi ,Yi} 16 12 inFigB? i T!I ! TSB
```

Далее со сформированными множествами можно выполнять разнообразные операции: проверять принадлежность элемента множеству, включать или исключать элементы из множества, а также получать из двух множеств новое множество путём их объединения, пересечения, вычитания одного множества из другого, используя соответствующие операции:

```
{ троичная проверка принадлежности k-го элемента множеству A} TSA k T@I
{ троичная проверка принадлежности k-го элемента множеству B} TSB k T@I
{  объявление нового множества } VAR TSC
{  пересечение множеств: C=A∩B } TSA TSB TMIN ! TSC
{  объединение множеств: C=AUB } TSA TSB TMAX ! TSC
{  вычитание множества B из A: C=A\B } TSA TSB NEG TMIN ! TSC
{  вычитание множества A из B: C=B\A } TSA NEG TSB TMIN ! TSC
```

Заключение

Целью написания статьи является знакомство с особенностями программирования в ДССП для троичной виртуальной машины (ТВМ). В статье на примерах конкретных программ, разработанных в ДССП, демонстрируется, какие новые возможности может предоставить программисту троичный компьютер. И тем самым ещё раз подчёркивается, что на определённом классе задач троичный компьютер по ряду параметров существенно превосходит двоичный, т.е. является более совершенным инструментом обработки информации.

Благодарности

Автор выражает благодарность Сидорову С.А. за реализацию в ТВМ команды цикла с троичным условием, являющейся аналогом новой команды цикла **DW−+** языка ДССП-Т.

Литература

1. Бурцев А.А., Сидоров С.А. Программный комплекс ДССП-ТВМ для структурированного программирования троичной [виртуальной] машины. // Современные информационные технологии и ИТ-образование. — 2015. — Т. 2, № 11. — С. 371–379.
2. А. А. Бурцев, С. А. Сидоров. Троичная виртуальная машина и троичная ДССП. // Программные системы: теория и приложения. — 2015. — Т. 6, №:4 (27). — С. 29–97. [электронный ресурс] URL: http://psta.psiras.ru/read/psta2015_4_29-97.pdf.
3. Владимирова Ю.С. Введение в троичную информатику: Учебное пособие. — М.: АРГАМАК-МЕДИА, 2015. — 160 с.
4. Баранов С.Н., Ноздрунов Н.Р. Язык ФОРТ и его реализации.— Л.: Машиностроение, 1988.— 157 с.
5. Брусенцов Н.П., Рамиль Альварес Х. Структурированное программирование на малой цифровой машине. // Вычислительная техника и вопросы кибернетики. Вып. 15. М.: Изд-во МГУ, 1978, с.3-8.
6. Дейкстра Э. Дисциплина программирования. — М.: Мир, 1978. — 275 с.

References

1. Burtsev A.A., Sidorov S.A. Programmnyj kompleks DSSP-TVM dlja strukturirovannogo programmirovanija troichnoj [virtual'noj] mashiny. // Sovremennye informacionnye tehnologii i IT-obrazovanie. — 2015. — T. 2, № 11. — S. 371–379.
2. Aleksey Burtsev, Sergey Sidorov. “Ternary virtual machine and ternary DSSP”, Program systems: theory and applications, 2015, 6:4(27), pp. 29–97. (In Russian.) URL http://psta.psiras.ru/read/psta2015_4_29-97.pdf
3. Vladimirova Ju.S. Vvedenie v troichnuju informatiku: Uchebnoe posobie. — M.: ARGAMAK-MEDIA, 2015. — 160 s.
4. Baranov S.N., Nozdrunov N.R. Jazyk FORTH i ego realizacii.— L.: Mashinostroenie, 1988.— 157 s.
5. Brusentsov N.P., Ramil' Al'vares H. Strukturirovannoe programmirovanie na maloj cifrovoj mashine. // Vychislitel'naja tehnika i voprosy kibernetiki. Vyp. 15. M.: Izd-vo MGU, 1978, s.3-8.
6. Edsger W. Dijkstra. A discipline of programming. — Prentice-Hall, Inc. Englewood Cliffs, N.J., 1976.

Поступила: 15.10.2017

Об авторах:

Бурцев Алексей Анатольевич, кандидат физико-математических наук, доцент, старший научный сотрудник отдела архитектур высокопроизводительных микропроцессоров отделения разработки вычислительных систем Федерального Научного Центра «Научно-исследовательский институт системных исследований», burtsev@niisi.msk.ru, доцент кафедры «программное обеспечение, информационные технологии и прикладная математика» Калужского филиала Московского государственного технического университета им. Н.Э. Баумана, burtsev59.aa@gmail.com

Note on the authors:

Burtsev Alexey, Candidate of physical and mathematical sciences, associate professor, high research associate of the department of architecture of high-performance microprocessors of the separation of development of computing systems, Federal Scientific Center Scientific Research Institute for System Analysis of the Russian Academy of Sciences (FSC SRISA RAS), burtsev@niisi.msk.ru, associate professor of the department "software, information technologies and applied mathematics" of the Kaluga branch of Bauman Moscow State Technical University, burtsev59.aa@gmail.com