Бурцев А. А., Рамиль Альварес Х.

Сопрограммный механизм в системе структурированного программирования для троичной машины

Введение

В НИЛ троичной информатики ВМК МГУ создан программный имитатор (ТВМ) [1] троичной машины двухстековой архитектуры, ориентированной на поддержку структурированного программирования. А также построена кросс-система разработки на языке ДССП-Т структурированных программ для троичной машины, получившая название ДССП-ТВМ [2]. Язык ДССП-Т был разработан как троичный вариант языка ДССП [3] с учётом возможности его обработки не только интерпретатором, но и кросс-компилятором.

Впоследствии на основе ДССП-ТВМ был создан интерпретатор языка ДССП-Т (ДССП/ТВМ [4]), способный функционировать на ТВМ как резидентное ПО. Успешной разработкой такого интерпретатора была подтверждена пригодность ДССП-ТВМ для построения полноценного программного оснащения троичной машины, отвечающего задачам автоматизации программирования.

Для создания первичного системного ПО троичной машины потребуется также решать и другую группу задач системного программирования, связанную с управлением одновременной работой нескольких внешних устройств. Совокупность программ, отвечающих за решение этих задач, традиционно считается частью операционной системы. Для разработки программ такого характера, как правило, требуются особые средства программирования, с помощью которых можно легко выразить алгоритмическую сущность параллельной работы нескольких устройств цифровой машины.

В качестве такого особого средства может успешно использоваться сопрограммный механизм [5,п.3.4]. Это было подтверждено ещё Виртом при создании ряда системных программ на языке Модула-2 [6], а также опытом построения на основе такого механизма разнообразных мониторов управления параллельными процессами в ДССП различных версий [7,8]. Поэтому в качестве развития возможностей ДССП-ТВМ (и её языка ДССП-Т) в направлении поддержки разработки операционных систем было предложено дополнить её операциями сопрограммного механизма.

Далее в статье объясняются привлекательные возможности сопрограммного механизма как средства структурирования сложных программ, раскрываются особенности его реализации в ДССП для ТВМ, а также приводятся примеры программ, разработку которых удалось существенно упростить благодаря применению этого механизма.

1. Сопрограммный способ структурирования программ

После знаменитой статьи Э.Дейкстры "Notes on structured programming", русский перевод которой был опубликован в фундаментальной книге, посвящённой структурированному программированию [5,ч.1], механизм подпрограмм (процедур) был общепризнан в качестве обнадёживающего способа (средства), применение которого при разработке программы позволяет придать ей желаемую (иерархическую) структуру, облегчающую понимание сути заложенного в неё алгоритма и динамики её поведения.

Но такой способ структурирования программ можно применять, когда между вызывающей и вызываемой программами предполагается строгая подчиненность. Вызываемая обязана выполнить по требованию вызывающей всю назначенную ей работу от начала до конца и только после этого вернуть управление вызывающей в точку её вызова. Таким образом, вызывающая программа выступает в роли главной, а вызываемая - в роли подчиненной.

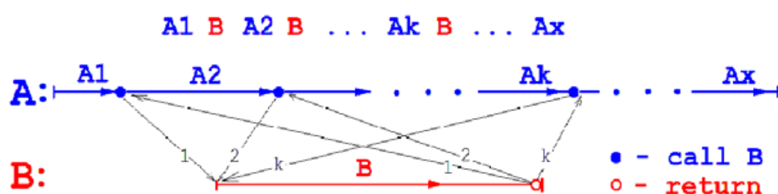
Заметим, однако, что такой способ взаимодействия программ не является единственно возможным. И в некоторых случаях взаимоотношение между двумя программными компонентами удобнее представлять не по принципу "главная/подчинённая", а по принципу равноправных партнёров, рассматривая их как **со**трудников, совместно решающих общую задачу, когда действия одного должны чередоваться с действиями другого. Для осуществления совместной деятельности такие программные компоненты должны регулярно предоставлять друг другу возможность выполнить каждому свою часть работы. Именно по такому принципу и осуществляют свое взаимодействие так называемые **со**программы (**со**трудничающие программы).

Поясним, в каком случае выгодно использовать сопрограммы. Допустим, программе требуется выполнять попеременно две различных по характеру группы действий: {Ai}{Bi}(i=1,2,..,k). Требуемую последовательность действий {A1,B1,A2,B2,...,Ак,Вк} можно, конечно, "запихнуть" в одну обычную программу, но такая программа будет совершенно нечитабельной: представьте себе, например, как труднее читался бы текст книги, в котором строки одной страницы чередуются попеременно со строками другой страницы.

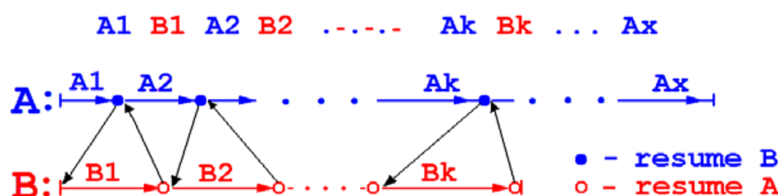
Распределим требуемые действия на две сопрограммы: пусть первая сопрограмма СА выполняет ряд действий: {A1,A2,...,Ак}, а

другая сопрограмма СВ выполняет ряд действий: {B1,B2,...,Bк}. Тогда для выполнения совместной работы: {A1,B1,A2,B2,...,Aк,Bк} потребуется лишь регулярно после выполнения очередного действия Ai передавать управление от сопрограммы СА к сопрограмме СВ на выполнение действия Bi, а после него - обратно от СВ к сопрограмме СА на выполнение действия, следующего в ней за Ai.

Передача управления от одной сопрограммы в другую с последующим возвратом существенно отличается от вызова подпрограммы (см. рис.1) в том, что передача производится не обязательно в начало другой сопрограммы, а в ту точку, где та сопрограмма ранее отдала управление. Это значит, что при передаче управления необходимо сохранить контекст текущей сопрограммы, чтобы восстановить его при последующем возврате в эту сопрограмму. Под контекстом же в таком случае понимается та информация, которую необходимо сохранять в момент приостановки процесса выполнения отдельной программы для его последующего корректного возобновления так, чтобы он мог развиваться далее, как будто бы и не приостанавливался вовсе.



а) Подпрограммный способ передачи управления



b) Сопрограммный способ передачи управления

Рисунок 1. Механизм подпрограмм (a) и механизм сопрограмм (b)

Одними из первых использовать сопрограммный механизм как удобный способ структурирования программ предложили Дал и Хоор ещё в своей знаменитой статье “Иерархические структуры программ”, русский перевод которой был опубликован в той же упомянутой ранее книге [5,ч.3]. В то время сопрограммный механизм уже появился в языке Симула-67, где был оценён и востребован при разработке программ для задач моделирования.

Анализируя приобретённый опыт применения сопрограммного механизма, Дал и Хоор охарактеризовали достаточно обширный класс задач из области программирования, решение которых можно значительно облегчить с помощью этого механизма. На конкретных примерах достаточно сложных программ они убедительно показали, как можно существенно упростить их разработку, если наряду с подпрограммным использовать ещё и сопрограммный способ для организации взаимодействия отдельных программных компонент.

Опираясь на мнение этих авторитетных специалистов, можно включить в рассматриваемый класс задач следующие виды программ.

1. Простейшим примером программы, требующей организации сопрограммной структуры, является игровая программа, которая "вычисляет" свой собственный ход, передаёт его противнику, вводит ответ, "вычисляет" следующий ход и т.д. до конца игры. Допустим, что уже разработаны две разные программы, умеющие играть в одну и ту же игру, и требуется определить, какая из них играет лучше. В таком случае полная программа игры совершенно естественным образом составляется из указанных двух игровых программ, но метод их структуризации отвечает скорее взаимоотношению сопрограмм, а не подпрограмм.

2. Другим примером сопрограммной структуризации является двухпроходной (и тем более многопроходной) компилятор для некоторого языка программирования. В результате первого прохода обычно выдаётся длинная последовательность сообщений, которая является входной информацией для второго прохода. Если размеры основной памяти достаточны для размещения полной программы, обеспечивающей реализацию обоих проходов, и нет необходимости плодить файлы временного характера, то можно, очевидно, провести всю компиляцию за один проход, в котором последовательность сообщений будет передаваться порциями из первого прохода во второй. В некоторых случаях можно, конечно, преобразовать один из проходов для использования его в качестве подпрограммы к другому, но если имеется возможность выбора, то лучше всё же рассматривать взаимодействие программ двух проходов по принципу сопрограмм.

3. В общем случае отдельной сопрограммой удобно представлять всякую завершённую независимую программу, команды ввода-вывода которой заменены на обращения к другим сопрограммам для производства или потребления информации. Поэтому сопрограммный способ лучше подходит и для построения программ, обрабатывающих последовательности элементов, получаемых в ходе нетривиального процесса генерации сложных объектов или путём обхода иерархических структур данных (списков, деревьев, графов).

4. В виде взаимодействующих сопрограмм удобнее также представлять структуру программы, которой при своём выполнении

требуется попеременно исполнять серии действий разнообразного вида. Исполнение последовательности действий каждого вида целесообразно в таком случае реализовать в форме отдельного самостоятельного алгоритма, оформив его как сопрограмму. Например, таким способом можно существенно облегчить построение программы, управляющей совместной работой нескольких внешних устройств.

Преимущества сопрограммного способа структурирования продемонстрируем на примерах отдельных программ генерации последовательностей элементов, сравнивая построенные алгоритмы с их альтернативными вариантами, которые были бы получены в случае вынужденного построения программы традиционным способом, т.е. только с помощью подпрограмм.

Допустим, основной программе (Main) требуется поставлять на обработку элементы матрицы X[1:m,1:n], перебирая их в заданном порядке (см. рис. 2): просто по столбцам (вариант А) или “змейкой” по столбцам (вариант Б с чётным n). Требуемый порядок перебора матрицы оформим в виде отдельной программной компоненты, взаимодействие с которой предусмотрим двумя разными способами: как с подпрограммой или как с сопрограммой. И сравним варианты полученных алгоритмов (см. таблицу 1) с точки зрения ясности их структуры, объясняющей сущность их поведения. При этом для записи алгоритмов будем использовать нотацию языка Паскаль с дополнениями: заголовок сопрограммы начинается ключевым словом **coprogram**, а для переключения на сопрограмму CP применяется оператор **resume** CP .

При сопрограммном способе (см. правую колонку в таблице 1) для любого варианта удаётся наглядно отразить естественную структуру алгоритма сопрограммы GSeq перебора матрицы: в варианте А это два вложенных цикла **for** по i и по j, а в варианте Б - один внешний цикл **for** по i, в который встроены два цикла **for** с перебором по j сначала в прямом, а затем в обратном порядке.

При использовании же подпрограммного способа (см. левую колонку в таблице 1), никакой наглядной структуры, отражающей суть алгоритма перебора, невозможно увидеть в подпрограмме Get, отвечающей за последовательную генерацию элементов. И если для варианта А подразумеваемые циклы перебора еще можно “домылить”, то в варианте Б уже становится весьма затруднительным “увидеть” задуманные циклы за нагромождениями условных операторов в теле процедуры Get, а тем более понять, что по сути здесь осуществляется перебор матрицы “змейкой” по столбцам.

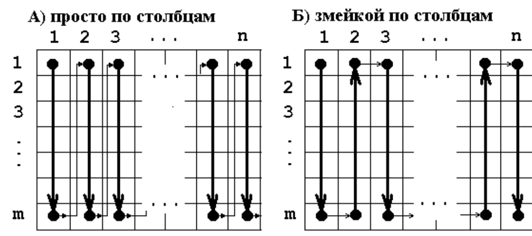


Рисунок 2. Варианты перебора элементов матрицы

const m=..; n= ..;{размеры матрицы} EOS=...;{признак конца псл-ти} **type** TItem= .. {тип элементов}; **var** X:array[1..m,1..n] of TItem; i,j:integer; CI: TItem; {текущий элемент} **procedure** GetI(**var** y:TItem); **begin resume** GSeq; y:=CI; **end**;	 **procedure** PutI(z:TItem); **begin** CI:=z; **resume** Main; **end**;
Вариант А. Перебор элементов матрицы просто по столбцам **:**	

procedure Init; **begin** i:=1; j:=1; **end**;
procedure Get(**var** y:TItem);
begin
if j<=m **then**
begin y:=X[j,i]; j:=j+1; **end**
else if i<n **then**
begin i:=i+1; y:=X[1,i]; j:=2; **end**
else y:=EOS;
end;
	
	coprogram GSeq;
	begin
	for i:=1 **to** n **do**
	for j:=1 **to** m **do**
	PutI(X[j,i]);
	PutI(EOS);{конец псл-ти}
	end.

Вариант Б. Перебор элементов матрицы “змейкой” по столбцам **:**	

procedure Get(**var** y:TItem);
begin
if i **mod** 2 =1 **then** {вниз}
if j<=m **then**
begin y:=X[j,i]; j:=j+1; **end**
else if i<n **then**
begin i:=i+1; y:=X[m,i]; j:=m-1;**end**
else y:=EOS
else {вверх} **if** j>0 **then**
begin y:=X[j,i]; j:=j-1; **end**
else
if i<n **then**
begin i:=i+1; y:=X[1,i]; j:=2; **end**
else y:=EOS;
end;
	coprogram GSeq;
	begin
	for i:=1 **to** n **div** 2 **do begin**
	for j:=1 **to** m **do**
	PutI(X[j,2*i-1]);
	for j:=m **downto** 1 do
	PutI(X[j,2*i]);
	end;
	PutI(EOS);
	end.

Таблица 1. Алгоритмы генерации последовательности

2. О возможности эффективной реализации сопрограммного механизма

К сожалению, предложенный упомянутыми авторитетными специалистами сопрограммный способ структурирования программ был оценён лишь в узких академических кругах и до сих пор не получил заслуженного признания и широкого распространения на практике.

Одной из причин тому, на наш взгляд, является, увы, отсутствие поддержки такого полезного механизма взаимодействия программ во многих языках, которые в своё время широко использовались (Алгол, Фортран, ПЛ/1), или ещё используются (Паскаль, Си) не только в обучении, но и в области практического программирования.

Возможно, такое пренебрежение механизмом сопрограмм объясняется сложностью или кажущейся неэффективностью его реализации на компьютерах, используемых в повседневной практике. И действительно, на машинах традиционной (фон-Неймановской) архитектуры, в которой характеристика состояния протекающего процесса исполнения программы (в каждый момент времени) включает совокупность значений обширного набора так называемых общих регистров, операция переключения контекста вряд ли может быть реализована достаточно эффективно, так как она требует сохранения в памяти и последующего восстановления всех этих регистров процессора. А с расширением набора таких общих регистров в компьютерах подобной архитектуры скорость исполнения операции переключения контекста ещё более замедляется. И вот таким недостатком архитектуры, за редким исключением, страдают, увы, и многие современные компьютеры.

Однако, операцию переключения контекста можно реализовать достаточно эффективно на машинах, процессоры которых содержат минимально необходимый набор общих регистров, а также в системах программирования, которые используют минимум процессорных регистров для организации исполнения разработанных в них программ.

Примером тому могут служить машины классической архитектуры PDP-11 (LSI-11)[9,гл.13], в которой (помимо регистра состояния PSW) было предусмотрено всего 8 общих регистров: R0-R5, указатель стека SP(R6) и счётчик команд PC(R7). И этого скромного ассортимента регистров, как ни странно, хватало для организации исполнения любых самых сложных программ, в том числе ряда популярных операционных систем и разнообразных систем автоматизации программирования.

Заметим, что секрет успешного применения этих машин феноминально прост. Всё дело в том, что реализованные для этих

машин системы программирования с популярными языками программирования Си и Паскаль предусматривали хранение обрабатываемых данных либо в глобальной памяти, либо в программно-организованном стеке. При этом часть общих регистров служила в качестве указателей обрабатываемых глобальных объектов (стеков, списков, кучи, области общих данных), а другая часть использовалась для временного размещения в них значений на период выполнения арифметических операций над данными, в основном хранящимися в памяти в форме специально организованных структур.

При таком порядке организации обрабатываемых данных в программе операцию переключения контекста, необходимую для обеспечения функционирования механизма сопрограмм, становится возможным реализовать достаточно просто. Ведь для этого требуется сохранять (и восстанавливать) лишь значения нескольких ключевых регистров, используемых в качестве указателей стеков и счётчика команд, а также регистры, применяемые для временного хранения значений, поскольку остальное содержимое контекста сопрограммы уже находится в памяти в организованном программой стеке (или стеках). Что и было подтверждено эффективной реализацией сопрограммного механизма в языке Модула-2 Вирта [6] для машин архитектуры PDP-11.

Для получения ещё более эффективного варианта функционирования сопрограммного механизма можно потребовать обеспечить аппаратную поддержку реализации операции переключения контекста. Например, как это впервые было реализовано в микропроцессорах семейства Texas Instruments [9,гл.16], где контекстное переключение можно было быстро осуществить путём изменения значения всего одного регистра, указывающего адрес размещения в быстродействующей памяти рабочей области, куда отображается содержимое всех общих регистров. В настоящее время аналогичные возможности реализованы в некоторых современных микропроцессорах в виде так называемых регистровых окон [10,п.2.4].

Наконец, следует обратить особое внимание на замечательный класс машин двухстековой архитектуры [11,12], в которых все операции по обработке данных осуществляются непосредственно над величинами, размещёнными в арифметическом стеке, а адреса возвратов из подпрограмм хранятся в управляющем стеке. Если содержимое их стеков отобразить в быстродействующую память, настроив на неё соответствующие регистры-указатели, то операцию переключения контекста в машинах такой архитектуры можно будет реализовать предельно эффективно, так как она будет заключаться лишь в перенастройке значений этих регистров-указателей на другой участок используемой под них памяти. Таким образом, к ряду преимуществ [11], которые стековые машины имеют по сравнению с машинами традиционной архитектуры, обременённых обширным набором общих

регистров, добавляется ещё одно – возможность обеспечения наиболее эффективного варианта реализации сопрограммного механизма.

3. Сопрограммы и параллельные программы

Следует признать, что в современных языках и системах программирования скромные возможности сопрограммного механизма затмеваются мощным аппаратом организации параллельных программ.

Заметим, однако, что для структурирования рассмотренных нами видов программ вовсе необязательно (да и не всегда целесообразно) прибегать к помощи средств обеспечения истинного или имитируемого параллелизма, чтобы не привносить в свои программы дополнительные проблемы, связанные с их непредсказуемым поведением в реальном времени.

Сопрограммный механизм позволяет легко отслеживать цепочку передач управления в программе, так как они всегда осуществляются по командам самой программы. Но этого нельзя гарантировать для параллельных программ, моменты переключений между которыми даже на одном и том же процессоре могут оказаться неподконтрольными самим программам. Поэтому при организации параллельных программ приходится ещё и решать специфические проблемы взаимного исключения, синхронизации исполнения ими отдельных участков своих алгоритмов.

Кроме того, параллельным программам может ограничиваться доступ к пространству памяти друг друга, и как следствие, запрещаться доступ к общим переменным, что влечёт необходимость серьёзной переделки построенных ранее алгоритмов. Подобных проблем, конечно же, не возникает при использовании сопрограммного механизма, ибо предполагается, что исполнение сопрограмм протекает в едином пространстве памяти (и имён переменных).

Можно сказать, что сопрограммный механизм и механизм организации параллельных программ являются разноуровневыми способами структуризации программ. Они не подменяют собой один другого и не исключают необходимости друг друга (ведь и кино не заменяет нам книги). Но оба этих механизма опираются на одну и ту же основу, которая их “роднит”. Такой основой является операция переключения (сохранения и восстановления) программного контекста.

Благодаря этому подмеченному “родству” оказалось возможным обеспечить организацию попеременного исполнения параллельных процессов на одном процессоре с помощью операции сопрограммной передачи управления. Именно такой путь реализации параллелизма был в своё время предложен Виртом в языке Модула-2, что позволило разрабатывать разнообразные программные модули-мониторы, управляющие развитием параллельных процессов, уже не на

языке ассемблера, а на языке высокого уровня. Таким образом, удалось существенно облегчить разработку мониторов-диспетчеров параллельных процессов (правда, лишь для однопроцессорных систем).

Аналогичный путь обеспечения параллелизма был в свое время предложен и для системы программирования ДССП.

4. Сопрограммный механизм в ДССП для ТВМ

ДССП может служить примером среды исполнения программ, характеризующейся двухстековой архитектурой. Поэтому реализация сопрограммного механизма в ней может быть осуществлена достаточно эффективно. Такая реализация была впервые осуществлена при разработке экспериментальной версии ДССП-РВ [7] для машин семейства PDP-11; затем она была выполнена для машин семейства Intel-x86 сначала в версии ДССП-32р (под управлением DOS-экстендера DOS4GW), а впоследствии и для мобильной версии ДССП/С [13]. На основе сопрограммного механизма в ДССП были разработаны (причём на языке самой системы!) разнообразные мониторы, управляющие развитием параллельных процессов [8].

В настоящее время сопрограммный механизм как полезный инструмент структуризации программ было решено реализовать и в версии ДССП-ТВМ, предназначенной для разработки программ, способных функционировать в среде имитационной модели создаваемой троичной машины (ТВМ).

4.1 Операции сопрограммного механизма в ДССП-ТВМ

Расширение языка ДССП-Т возможностями организации сопрограмм выполнено в версии ДССП-ТВМ, поддерживающей объектно-ориентированный стиль программирования [14]. Средства, обеспечивающие функционирование механизма сопрограмм, сосредоточены в отдельном модуле ДССП-библиотеки, который может быть подгружен командой: `LOAD` **`context.dsp`**.

В этом модуле объявлен новый тип данных **`CONTEXT`** как структура с полями, предусмотренными для сохранения значений регистров, характеризующих сопрограммный контекст. А также определены слова-операции над объектами этого типа.

Для представления каждой сопрограммы сначала требуется объявить в программе объект типа **`CONTEXT`** , например:

```
CONTEXT VAR CA  CONTEXT VAR CB  CONTEXT VAR CMAIN
```

Объявленные так слова-имена объектов (CA,CB,CMAIN) будут теперь засылать в стек адреса самих этих объектов, размещённых в памяти.

Первоначально требуется проинициализировать контекст основной программы как сопрограммы, для чего используется команда:

`{ CMAIN } MAINCONTEXT { }`

Необходимо также привести в начальное состояние контекст каждой новой сопрограммы, для чего предусмотрена операция:

`{ PRA, CA, Addr, DSLen, CSLen } NEWCONTEXT { }`

где PRA – адрес процедуры, определяющей тело сопрограммы;

CA – адрес объекта, представляющего контекст этой сопрограммы;

Addr – адрес рабочего пространства, необходимого сопрограмме для своего функционирования (в нём выделяется место под стек операндов и стек возвратов);

DSLen,CSLen – длины областей под стеки (операндов и возвратов).

После этого можно осуществлять передачи управления от одной сопрограммы к другой. Для этого предусмотрена основная операция переключения контекста сопрограммы:

`{ CA,CB } TRANSFER { CZ }`

где CA – указатель контекста текущей сопрограммы;

CB – указатель контекста возобновляемой сопрограммы;

CZ – указатель контекста приостановленной сопрограммы.

Эта операция сохраняет контекст текущей сопрограммы CA и возобновляет сопрограмму CB, передавая в стеке указатель контекста прерванной сопрограммы CZ, от которой перешло управление к данной. Заметим, что CZ может не совпадать с CB, например, в случае передачи управления по такой цепочке: CA -> CB -> CZ -> CA .

Предусмотрены также операции возобновления сопрограммы, в которых не требуется указатель контекста текущей сопрограммы:

`{ CB } RESUME { }   { CB } RESUME= { CZ }`

При использовании этих операций требуемый указатель автоматически сохраняется в специально предусмотренной переменной Cntxt, но инициализация сопрограммного механизма должна осуществляться уже другой командой:

`{ CMAIN } INITCONTEXTS { }`

Заметим, что реализацию этих дополнительных операций можно обеспечить с помощью описанных выше основных операций:

```
TWORD VAR Cntxt {контекст текущей сопрограммы}
:: : INITCONTEXTS {CMain} C MAINCONTEXT ! Cntxt {};
:: : RESUME= {CB} Cntxt E2 C ! Cntxt {Cntxt:=CB}
              {OldCntxt, CB } TRANSFER { CZ } ;
:: : RESUME {CB} RESUME= {CZ} D { } ;
```

4.2 Специфика реализации операции контекстного переключения в ДССП для ТВМ

Операция передачи управления сопрограмме (TRANSFER) требует сохранения и последующего восстановления контекста выполняющейся ДССП-программы. Такой контекст в текущей версии ДССП-ТВМ полностью определяется содержимым двух стеков: арифметического и управляющего.

В ТВМ стеки исполняемой программы отображены в память так, что регистры-указатели DSP-DSPL-DSPH определяют расположение вершины, дна и "потолка" арифметического стека, а регистры CSP-CSPL-CSPH – управляющего стека соответственно. Поэтому для реализации операции переключения контекста в ДССП-ТВМ достаточно сохранять имеющиеся значения только этих регистров-указателей стеков (в полях объекта-структуры CA) и восстанавливать их новые значения (из полей объекта CB). Именно эти действия и выполняет операция TRANSFER, реализованная в программном модуле "context.dsp" на языке ДССП-Т как слово, определённое в ассемблерном коде (т.е. между компилирующими словами :ASM и ;ASM).

Замечание. Хотя при функционировании ДССП-программы используются ещё и четыре общих регистра R0-R3 ТВМ, при реализации операции переключения контекста их можно не сохранять, ибо они используются лишь как временные на этапе исполнения ДССП-команд, реализованных в машинном коде. Конец замечания.

При реализации операции TRANSFER обнаружился досадный недостаток системы команд ТВМ. Оказалось, что нельзя по отдельности изменять значения регистров-указателей одного и того же стека, поскольку в ТВМ встроен контроль их правильности, подразумевающий соблюдения условия: DSPL<=DSP<=DSPH для арифметического стека и условия: CSPH<=DSP<=DSPL для управляющего стека.

Поэтому было предложено добавить в ТВМ новые команды для совместного группового сохранения в памяти (**SVDS**, **SVCS**) и загрузки из памяти (**LDDS**, **LDCS**) значений триады регистров-указателей, определяющих расположение в памяти арифметического и управляющего стека соответственно.

Наконец, отметим, что такую операцию переключения контекста целесообразно (при дальнейшем развитии ТВМ) реализовать всё же одной машинной командой. Это позволило бы не только ускорить переключения между сопрограммами, но и обеспечить неделимость исполнения этой операции на случай возникновения возможных прерываний.

5. Примеры разработок ДССП-программ на основе сопрограммного механизма

Реализованный в ДССП-ТВМ сопрограммный механизм был опробован при создании ряда полезных программ, разработку которых удалось существенно облегчить благодаря применению этого механизма. Кратко охарактеризуем некоторые из них.

5.1 Обработка зашифрованного текста

На основе сопрограммного механизма в ДССП-ТВМ была реализована программа, аналогичная рассмотренной в п.1. Она осуществляет обработку текста, зашифрованного по так называемому методу транслитерации: предполагается, что поступающие из файла символы сначала записываются в массив по строкам, а выбираются из него на дальнейшую обработку по столбцам (или “змейкой” по столбцам). Для основной ДССП-программы, занятой непосредственно анализом и обработкой поступающих символов, требовался своеобразный “помощник”, который взял бы на себя всю работу по расшифровке принимаемого из файла текста. Такой “помощник” и был реализован в виде отдельной сопрограммы.

5.2 Игровая программа угадывания числа

Игровая программа угадывания девятеричного числа (своеобразный троичный аналог известной игры “Быки и коровы”) построена в ДССП-ТВМ как совокупность четырёх сопрограмм, “роли” между которыми распределены следующим образом (см. рис.3).

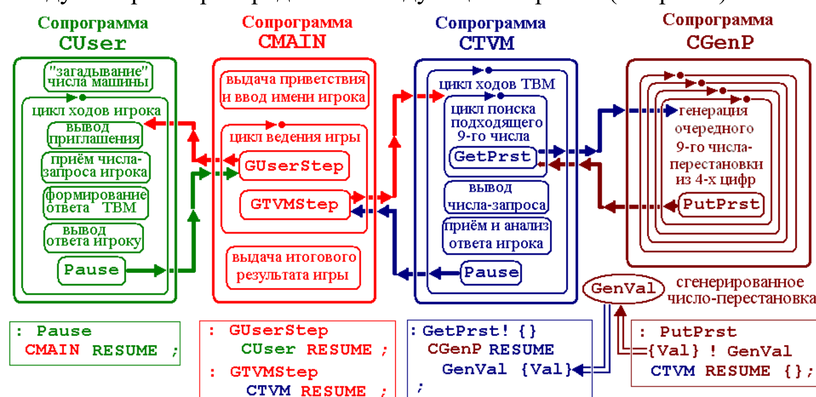


Рисунок 3. Сопрограммная структура игровой программы

Основная сопрограмма CMAIN организует диалог с пользователем и взаимодействие с сопрограммами, выполняющими ходы игрока и машины (ТВМ). Сопрограмма CUser отвечает на ходы-запросы игрока. Сопрограмма CTVM осуществляет основную игровую деятельность машины, т.е. ведёт поиск загаданного числа, формирует ходы-запросы машины, получает ответы игрока на них, корректируя по ним дальнейший поиск решения. Для получения очередного числа-претендента она обращается к вспомогательной сопрограмме CGenP, которой поручено перебирать все числа, являющиеся правильными перестановками 9-чных цифр (012345678) длиной 4. Для генерации всех таких чисел-перестановок сопрограмма CGenP организует 4 вложенных цикла (для перебора по каждой требуемой цифре).

5.3 Простой монитор управления процессами

На основе сопрограммного механизма в ДССП-ТВМ создан модуль-монитор "**spmon.dsp**", позволяющий организовать исполнение параллельных процессов на одном процессоре по круговой очереди. Главная особенность этого монитора заключается в том, что он предоставляет запущенным процессам возможность самим регулировать переключения процессора между ними, не требуя для осуществления таких переключений регулярных прерываний от таймера. Для этого каждый процесс обязан периодически выполнять специальную операцию (см. далее **PAUSE**), чтобы уступить процессор следующему процессу по круговой очереди (см. рис.4).

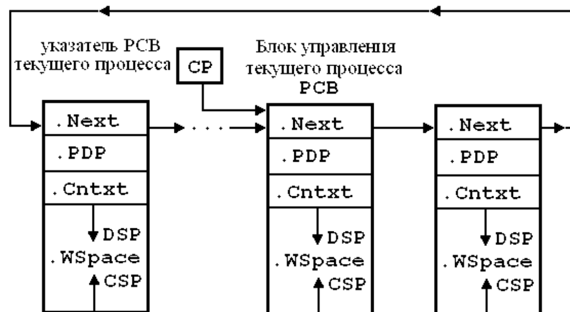


Рисунок 4. Круговая очередь процессов

Основные операции монитора над процессами, объявленными в ДССП-программе как объекты класса **PROCESS** :

```
PROCESS VAR PR
```

реализованы в нём как методы (т.е. в объектно-ориентированном стиле):

1) создать процесс: **CREATE** PR PB { PB – процедура тела процесса } ;
2) запустить процесс: **START** PR ;
3) остановить процесс: **FINISH** PR .

Над текущим процессом можно также выполнять операции:

4) остановить процесс: **STOP** ;
5) задержать процесс на заданное время: {Dt} **DELAY** ;
6) задержать процесс до заданного времени: {Wt} **AFTER** ;
7) приостановить процесс с возобновлением другого: **PAUSE** .

Перед запуском процессов необходимо проинициализировать работу монитора командой **INITPROCESS**. Далее переключения между запущенными процессами осуществляются путём исполнения ими операций, вызывающих сопрограммную передачу управления:

```
CLASS: PROCESS
  VAR .Next VAR .PDP CONTEXT VAR .Cntxt    ...
;CLASS
 : PAUSE NextReady RunReady ;
 : NextReady CP .Next {PCB};
 : RunReady {PCB} C ! CP {PCB} .Cntxt RESUME { } ;
```

Заключение

Таким образом, в процессе построения для троичной машины системы структурированного программирования, пригодной для разработки её системного ПО, проведён ряд результативных работ.

Исследована возможность реализации в ТВМ операции контекстного переключения. Система команд ТВМ дополнена необходимыми командами для совместной загрузки и сохранения регистров-указателей стеков.

В ДССП-ТВМ встроен сопрограммный механизм, на основе которого создан простой монитор, обеспечивающий попеременное исполнение параллельных процессов на одном процессоре без прерываний от таймера. А также разработан ряд других полезных программ, иллюстрирующих приёмы применения этого механизма.

Литература

Сидоров С.А., Владимирова Ю.С. Троичная виртуальная машина. // Программные системы и инструменты. Тематический сборник №12, М.: Изд-во факультета ВМиК МГУ, 2011. с.46-55.

Бурцев А.А., Рамиль Альварес Х. Кросс-система разработки программ на языке ДССП для троичной виртуальной машины // Программные системы и инструменты. Тематический сборник №12, М.: Изд-во факультета ВМиК МГУ, 2011. с.183-193.

Бурцев А.А., Сидоров С.А. История создания и развития ДССП: от "Сетуни-70" до троичной виртуальной машины. // Вторая Международная конференция "Развитие вычислительной техники и её программного обеспечения в России и странах бывшего СССР" SORUCOM-2011 (12-16 сентября 2011 г., г. Великий Новгород, Россия): Труды конференции. В.Новгород: Изд-во НовГУ, 2011. с. 83-88.

Бурцев А.А., Бурцев М.А. ДССП для троичной виртуальной машины. // Труды НИИСИ РАН, т.2, №1. М.: Изд-во НИИСИ РАН, с.73-82.

Дал У., Дейкстра Э., Хоор К. Структур(ирован)ное программирование. М.: Изд-во Мир, 1975.

Вирт Н. Программирование на языке Модула-2. М.: Изд-во Мир, 1987.

Борисов А.В. Диалоговая система структурированного программирования в реальном времени – ДССП-РВ. // Диалоговые микрокомпьютерные системы. М.: Изд-во МГУ, 1986, с.51-62.

Бурцев А.А., Шумаков М.Н. Сопрограммный механизм в ДССП как основа для построения мониторов параллельных процессов. // Вопросы кибернетики. Сб. статей под ред. В.Б.Бетелина. М., 1999. с.45-63.

Уокерли Дж. Архитектура и программирование микро-ЭВМ. М.: Изд-во Мир, 1984.

Корнеев В.В, Киселев А.В. Современные микропроцессоры. – М.: НОЛИДЖ, 2000.

Брусенцов Н.П. Стековые машины с изменяемой адресностью команд. // Вычислительная техника и вопросы кибернетики. Вып. 13. М.: Изд-во МГУ, 1977, с.97-112.

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.

Бурцев А.А. ДССП – среда структурированной разработки программ как сложных систем. // Вторая Международная конференция “Системный анализ и информационные технологии” САИТ-2007 (10-14 сентября 2007 г., Обнинск, Россия): Труды конференции. Изд-во ЛКИ, 2007. т. 2. с.190-194.

Бурцев А.А., Рамиль Альварес Х. Реализация средств объектно-ориентированного программирования в кросс-компиляторе языка ДССП-Т. // Программные системы и инструменты. Тематический сборник №13, М.: Изд-во факультета ВМК МГУ, 2012. с.28-37.

Опубликовано: Программные системы и инструменты № 14. Под ред. Л.Н. Королева. – М.: Издательский отдел ВМиК МГУ, 2013. С. 199 208.