Сидоров С.А., Владимирова Ю.С.

Троичная виртуальная машина

Введение

Троичная виртуальная машина (ТВМ, Ternary Virtual Machine - TVM) представляет собой программный эмулятор архитектуры троичного компьютера, основанного на троичной симметричной системе счисления и наследующего некоторые принципы троичных ЭВМ «Сетунь» и «Сетунь-70» [1].

Появилась ТВМ в результате структурирования исходной задачи – реализации троичной версии Диалоговой системы структурированного программирования (ДССП) [2]. Выделение виртуальной машины в самостоятельную разработку позволило четко разделить деятельность разработчиков, сделав ее относительно независимой, а также более строго проработать и реализовать архитектуру троичного компьютера, которая сама по себе требует принятия решений по массе вопросов, которые в двоичных машинах давно решены.

Программный эмулятор работает на обычных двоичных компьютерах, полностью воссоздавая троичную среду на уровне машинных команд, регистрового файла, памяти и адресного пространства. Программист ТВМ имеет дело только с троичными объектами и не должен беспокоиться о представлении информации. В его распоряжении обычные для этого уровня регистры, память, стеки, прерывания и пр.

Для программирования ТВМ разработан простой транслятор с языка ассемблера троичного процессора, поддерживающий набор машинных команд и несколько необходимых директив для размещения данных и кода. Язык ассемблера используется как для непосредственного программирования ТВМ, в частности, ядра разрабатываемой троичной ДССП, так и в качестве промежуточного языка для кросс-компилятора троичной ДССП – кросс-компилятор генерирует текст на языке ассемблера, который затем транслируется в машинный код. Это позволяет радикально упростить кросс-компилятор ДССП и заметно облегчить внесение изменений в архитектуру ТВМ и реализацию ДССП, развязав их на уровне генерации текста.

Наконец, для удобства использования в ТВМ встроен интерактивный монитор, обеспечивающий взаимодействие оператора с виртуальной машиной в консольном режиме. Монитор предоставляет простые возможности загрузки программ в память ТВМ, передачи управления программе, просмотра и модификации памяти и регистров

виртуальной машины, задания точки останова, протоколирования работы и пр.

Таким образом, ТВМ включает собственно виртуальную машину, интерактивный монитор и транслятор с языка ассемблера, которые в совокупности образуют среду разработки и отладки программ, достаточно удобную для создания троичной версии ДССП.

Еще одна цель создания ТВМ состоит в исследовании способов возможной реализации троичной архитектуры в кремнии с предварительным моделированием в двоичных ПЛИС. В этом контексте тщательная проработка представления данных и алгоритмов выполнения арифметических и логических действий приобретает особое значение. Один из подходов к непосредственной реализации троичной архитектуры описан в [3].

Троичная симметричная система счисления

Троичная система счисления, как и двоичная, основана на позиционном принципе кодирования чисел, в которой вес n-ого разряда равен не 2^n, а 3^n. При этом сами разряды, в отличие от битов, не двухзначны (как биты), а трехзначны (их принято называть тритами – trit). Помимо 0 и 1 они допускают третье значение, которым в симметричной системе служит –1, благодаря чему единообразно представимы как положительные, так и отрицательные числа. Таким образом, в троичной симметричной системе счисления знаком числа оказывается цифра старшего из его значащих (ненулевых) разрядов. Проблемы чисел со знаком, не имеющей в двоичном коде совершенного решения, в троичном симметричном коде просто нет, чем и обусловлены его принципиальные преимущества:

- естественное и неизбыточное представление чисел со знаком;
- округление числа производится простым отбрасыванием младших разрядов;
- погрешности округления в процессе вычислений взаимно компенсируются;
- явное представление третьего (привходящего) логического значения, необходимого для адекватного представления логических отношений.

Представление данных

Очевидно, для представления минимальной единицы троичной информации – трита – в двоичном компьютере требуется как минимум два бита. Для троичного процессора выбрано представление парой соответствующих битов в двух двоичных словах {P,M}, причем в первом слове P единицы стоят на тех местах, где в представлении

троичного числа стоят “1”, а в слове M – там, где в троичном числе стоят “–1”. Например, если обозначить “–1” как “–”, а “1” как “+“, то троичное число “00+0–0+–+” в двоичном компьютере будет выглядеть как {P,M}={001000101,000010010}.

Такое представление в большинстве случаев допускает параллельную обработку целых машинных слов и менее громоздко в программировании.

Архитектура троичного процессора

Хотя некоторые принципы архитектуры троичного процессора восходят к троичным ЭВМ «Сетунь» и «Сетунь-70», это не воспроизведение их архитектуры, а попытка создания новой, вобравшей основные преимущества прототипов (например, двухстековую организацию) и поддержку структурированного программирования, апробированного в ДССП.

Задуманная как составная часть проекта по разработке троичной ДССП, ТВМ на данном этапе не претендует ни на универсальность, ни на эффективность. Основная задача, решаемая с помощью ТВМ – оценка принимаемых решений и исследование вариантов подходов к реализации троичного компьютера.

Основные определения

Трит – минимальная единица троичной информации, принимает значения –1, 0 и 1.

Трайт – минимальная адресуемая единица информации для обмена с памятью. Состоит из 9 тритов, принимает 19683 различных значений в диапазоне [–9841, 9841]. Триты в трайте нумеруются справа налево, начиная с нуля и до 8.

В ЭВМ «Сетунь» использовались понятия короткого (9 тритов) и длинного (18 тритов) слова, в «Сетуни-70» размер трайта был равен 6 тритам, а слова – 18 тритов. Такой выбор был обусловлен разнообразными причинами, но пожалуй одной из главных была экономия весьма дорогостоящей в то время памяти. Сегодня все же представляется более логичным выбор размеров элементов данных кратными степени тройки.

Слово – группа из 3 смежных трайтов (т.е. 27 тритов). Принимает 7’625’597’484’987 различных значений, расположенных в диапазоне [–3’812’798’742’493, 3’812’798’742’493].

В ТВМ принята адресация little-endian, т.е. адресом слова является адрес младшего трайта. При этом слово, в т.ч. команда, может начинаться с любого адреса трайта.

Стек данных – структура данных с дисциплиной доступа «последним вошел – первым вышел» (LIFO). Элементом данных в стеке является слово. Первым элементом данных в стеке считается его ***вершина***, т.е. то, что было помещено в стек последним и будет извлечено первым. Далее следует второй элемент (***подвершина***), третий, и т.д.

Адресное пространство – диапазон адресов трайтов памяти. Адресное пространство ограничено диапазоном целых чисел, представимых одним словом: [–MAX, MAX], где MAX = 3’812’798’742’493. Адресное пространство единое для команд и данных. Регистры внешних устройств отображаются также в адресное пространство.

В отличие от привычных двоичных машин, нулевой адрес в адресном пространстве троичного процессора расположен посередине. Это обстоятельство явно используется для размещения программ и данных.

Общая характеристика архитектуры троичного процессора

В микропроцессоре используется троичная симметричная система счисления.

Для обработки данных используется стек данных. Арифметические и логические команды потребляют свои аргументы из стека данных, туда же засылают результаты. Есть модификации команд для работы с данными, заданными непосредственно в коде машинной команды, например, прибавление константы к значению, размещенному в вершине стека данных.

Второй стек, называемый управляющим или стеком адресов возврата (стеком возвратов), предназначен для сохранения адреса возврата из подпрограммы, сохранения контекста. Элементом стека возвратов также является слово.

Наряду со стеками в процессоре имеются регистры для хранения указателей стеков и их границ, флагов и прочей служебной информации, а также 4 регистра общего назначения. Регистры могут участвовать в формировании адреса данных в качестве индекса или базиса. Кроме пересылок данных в/из регистров, для них определено только прибавление некоторого значения, все прочие действия непосредственно над регистрами не предусмотрены.

Система прерываний векторная. Определены несколько внутренних исключительных ситуаций (переполнение/исчерпание стека, неопознанный код операции и т.п.), а также внешнее прерывание. Каждому событию сопоставлен адрес-вектор, откуда берется адрес обработчика исключения/прерывания.

Система команд включает команды пересылок данных, арифметические и логические операции, группу команд для работы со стеком, а также структурированные команды управления – вызов подпрограммы, команды ветвления и команду организации цикла.

В явном виде отсутствует команда goto. В то же время, поскольку регистр программного счетчика PC доступен для записи, действие команды перехода может быть выполнено засылкой в PC требуемого значения, в том числе вычисленного.

Регистры процессора

Все регистры троичного процессора имеют размер в одно слово. Часть тритов регистра может не использоваться, в этом случае оттуда считываются нули, при записи эти триты игнорируются.

Программно доступные регистры:

MEMCAP	- (read only) Размер памяти в трайтах
CSP	- Control Stack Pointer - указатель стека возвратов
CSPL	- CSP Low – нижняя граница стека адресов возврата
CSPH	- CSP High – верхняя граница стека адресов возврата
DSP	- Data Stack Pointer - указатель стека данных
DSPL	- DSP Low – нижняя граница стека данных
DSPH	- DSP High – верхняя граница стека данных
EXC	- Exception – вектор исключения (0 – исключения нет)
PC	- Program Counter – программный счетчик
STATUS	- маски и флаги прерываний и пр.
R0-R3	- регистры общего назначения
IVBASE	- базовый адрес векторов прерываний

Имеется также несколько программно недоступных регистров, играющих важную роль в организации исполнения команд.

IC	- Instruction Code – код инструкции, которая будет выполняться
ICL	- Instruction Code Lock – флаг блокировки IC

Структура стеков

Оба стека располагаются в основной памяти троичного компьютера. В регистрах процессора находятся указатели на вершину и верхняя и нижняя границы.

Стек данных растет в сторону увеличения адресов, стек возвратов – в сторону уменьшения. При попытке извлечь данные из пустого стека и при попытке положить данные в заполненный стек возникают исключения.

Элементы данных размером в один или два трайта укладываются в стек данных в младшие трайты вершины, извлекаются также из младших трайтов. Для стека возвратов возможен обмен только словами данных.

Форматы инструкций

Все инструкции троичного процессора занимают 1 слово. Это позволяет сделать существенно более простую реализацию и радикально сократить количество команд за счет разнообразия операндов.

Всего предусмотрено 3 формата команд троичного процессора, различающихся старшим тритом. При этом если старший трит равен 0, то это команда CALL безусловного вызова подпрограммы, а оставшиеся в коде команды 26 тритов представляют адрес подпрограммы. Хотя при такой кодировке часть адресного пространства недоступна для непосредственного вызова там подпрограммы, оставшегося диапазона 3^{26} (более 2500 Гигатрайт) более чем достаточно для хранения программ.

Все остальные команды кодируются в формате, где в старшем трите стоит значение 1. Под код операции отведено 5 тритов (243 команды, сейчас используется менее половины), 3 трита – под номер регистра, и остальные 18 тритов – два трайта – под непосредственное значение. Оно используется как слагаемое при вычислении адреса, как непосредственный операнд в командах обработки данных и т.п. Диапазон значений [–193`710`244, 193`710`244] достаточен для большинства случаев применения этих значений.

Команды, в старшем трите которых стоит значение –1, зарезервированы для будущих применений.

Выполнение инструкций

Выполнение инструкций троичного процессора определяется работой простого автомата, описываемого следующим алгоритмом:

```
NEXT_INSTRUCTION:
// Это исключение (прерывание)?
if ((ICL==0) && ((EXC<-3) || ((EXC==-3) &&
(STATUS[IE]==1)))
   {
   pushcs(STATUS); // сохранить STATUS
   STATUS[IE]=0;   // запретить прерывания
   pushcs(PC);     // сохранить PC
   PC = m(IVBASE + EXC); // извлечь адрес обработчика
```

```
   EXC = 0;
   }
else  // Исключения нет
   {
   if (ICL == 1) // IC locked, исполняем инструкцию из IC
      ICL=0;
   else      // Как обычно берем очередную инструкцию
      { IC = m(PC); PC = PC+3; }
   }
// здесь IC и PC имеют правильное значение
execute;
```

Набор инструкций троичного процессора

Далее приведен обзор набора инструкций троичного процессора ТВМ.

Управляющие инструкции представлены вызовами подпрограмм, условными и безусловными, командой организации цикла и рядом дополнительных команд.

Безусловный вызов подпрограммы выполняет команда CALL, в качестве операнда потребляющая прямой адрес подпрограммы. В языке ассемблера это обычная метка.

Возврат из подпрограммы обеспечивает команда RET без аргументов. Для возврата из обработчика исключения (прерывания) должна применяться команда RFE, т.к. при входе в обработчик в стек засылается также содержимое регистра STATUS.

Имеется также возможность выполнить любую команду троичного процессора, задав ее код в стеке данных; это делает инструкция EXEC.

Команды ветвления по условию фактически выполняют условное выполнение одной из инструкций, стоящих следом. Как и в языке ДССП, предусмотрены команды ветвления по знаку операнда в вершине стека данных на одну, две и три ветви – IF*, BR*, BRS соответственно, где * означает буку P – plus, N – minus или 0. Например, запись

```
[x]  BRM  p1  p2  p3
```

означает, что перед выполнением инструкции BRM (ветвиться по минусу) в стеке находится значение x (в квадратных скобках в качестве комментария указывается содержимое стека); инструкция BRM извлекает это значение и анализирует его. Если x<0, то выполняется инструкция p1, иначе p2. В любом случае, после выполнения p1 или p2 будет выполняться p3.

На месте p1 и p2 могут стоять любые инструкции троичного процессора, за исключением инструкций ветвления и цикла. Так, если фрагмент программы будет выглядеть так

```
[x]  BRM  CALL_p1  CALL_p2  p3
```

то будет выполняться условный вызов подпрограммы p1 или p2.

Команда организации цикла устроена аналогично:

```
[cond]  DW  p
```

Здесь cond – некоторое условие (нулевое или ненулевое значение). Инструкция DW извлекает его из стека и проверяет. Если оно не равно нулю, то выполняется команда p (как правило, это вызов подпрограммы), которая должна оставить после себя в стеке новое значение cond. После выполнения p управление снова попадает на DW. Если же значение cond было нулевым, то выполняется инструкция, стоящая вслед за p.

Таким образом, единственная команда цикла обеспечивает построение циклов всех видов – бесконечных, со счетчиком и с проверкой условия.

Целочисленные арифметические инструкции обычные – сложение, вычитание, умножение и деление, а также сравнение, максимум и минимум. Особенность их в том, что операнды эти инструкции изымают из стека данных, и засылают обратно результат.

Следует отметить, что в симметричной системе счисления правила вычисления знаков частного и остатка от деления, а также их значений иные, чем в привычной. Они вычисляются по следующему алгоритму (на языке Си):

```
// Деление n/m
// n = q*m + r, q – частное, r - остаток
q = n/m;
r = n%m;
m = abs(m);
if ( 2*abs(r) > m )
   {
   if (q < 0) q=q-1; else q=q+1;
   if (r < 0) r=r+m; else r=r-m;
   }
```

Как и в любом процессоре, значительное место в наборе команд занимают ***инструкции пересылки данных***. Троичный процессор обеспечивает пересылку данных в/из памяти, стеков, регистров, а также засылку непосредственно заданных в коде команды значений. При обмене с памятью поддерживается чтение и запись одного, двух и трех трайтов, при пересылках стек-стек и стек-регистр используется только формат слова. Засылка значения в стек и извлечение из стека сопровождается изменением указателя вершины стека, при этом

контролируется нахождение указателя в заданных пределах. При выходе за границы возникает исключительная ситуация.

Стековая организация процессора требует набора команд для ***манипуляции содержимым стека данных***. Эти команды были хорошо проработаны в ДССП, здесь они воспроизведены в чуть более общем виде: имеется команда удаления n позиций из стека данных, копирования значения с указанной глубины, обмена вершины и любого элемента стека. Есть варианты этих команд с заданием параметра в вершине стека, т.е. вычисляемого.

К ***логическим инструкциям*** традиционно относят сдвиги и поразрядные операции. В троичном процессоре предусмотрены сдвиги (причем специальные арифметические сдвиги в симметричной системе счисления не требуются), поразрядные сложение, умножение, минимум и максимум.

При разработке программ для троичного процессора необходимо обеспечить возможность ввода-вывода, по крайней мере взаимодействие с консолью и чтение-запись файлов. В дальнейшем предполагается разработка интерфейса ТВМ к обычным внешним устройствам (очевидно, двоичным), а на сегодняшний день в троичном процессоре реализованы временные команды, поддерживающие элементарные возможности ввода- вывода: распечатка вершины стека, печать символа и строки, работа с файлами

Интерактивный монитор

Интерактивный монитор в составе ТВМ выполняет роль «пульта управления» виртуальной машиной. Аналогичные возможности как правило предоставляются встроенным программным обеспечением компьютеров, в данном же случае функциональность интерактивного монитора реализована в самой эмулирующей программе.

Монитор выполняет команды пользователя, вводимые с клавиатуры, и отображает результаты на экране. Команды монитора обеспечивают основные действия по загрузке, выполнению и отладке программ.

Загрузка программы в ТВМ выполняется из файла, подготовленного компилятором с языка ассемблера, причем загрузить программу можно как при старте ТВМ, так и уже в интерактивном режиме.

Для отладки программ предусмотрены команды просмотра и изменения значения регистров троичного процессора и памяти в различных форматах, а также установка и снятие точек останова, старт и продолжение выполнения программы, пошаговое исполнение.

Имеется возможность записи протокола отладки в файл для последующего анализа и выполнения последовательности команд из файла, что существенно ускоряет процесс отладки.

Ассемблер троичного процессора

Язык ассемблера предоставляет возможность программирования в символьных обозначениях для троичного процессора ТВМ. Программа на языке ассемблера представляет собой последовательность инструкций в свободном формате. Разделителем инструкций служит пробел или точка с запятой. В случае инструкций переменной длины завершающая точка с запятой обязательна.

Инструкции и операнды отделяются по крайней мере одним пробелом (символом табуляции, концом строки) или знаком “_” (подчеркивание). Таким образом, команды, состоящие из нескольких слов, можно визуально объединять знаком подчеркивания, а в других случаях использовать пробел.

Числа могут быть представлены в троичной, девятеричной, десятичной и шестнадцатеричной системах счисления, в т.ч. со знаком минус.

В языке ассемблера предусмотрены метки – символические обозначения адресов. Метка – имя, за которым без пробела стоит двоеточие. Метки могут располагаться в произвольном месте программы между инструкциями.

Набор директив транслятору традиционен: размещение кода программы, размещение данных, определение текстовых строк и пр.

В качестве результата трансляции выдается файл с троичным кодом программы и текстовый файл с дизассемблированной программой.

Литература

1. 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.

2. Брусенцов Н.П., Захаров В.Б., Руднев С.А., Сидоров С.А., Чанышев Н.А. Развиваемый адаптивный язык РАЯ диалоговой системы программирования ДССП. М.: Изд-во Моск. Ун-та, 1987 г. – 80 с.

3. Jeff Connelly. Ternary Computing Testbed. 3-Trit Computer Architecture. Computer Engineering Department, California Polytechnic State University of San Luis Obispo, 2008.

Опубликовано: Программные системы и инструменты. Тематический сборник № 12. М.: Изд-во факультета ВМиК МГУ, 2011. С. 46-55.