

Автоматизация проектирования и программирования

УДК 681.3-181.4:681.3.06

© 1990 г.

Ю. В. МАТИЯСЕВИЧ, д-р физ.-мат. наук

[Ленинградское отделение Математического института АН СССР им. В. А. Стеклова],

А. Н. ТЕРЕХОВ, канд. физ.-мат. наук

[НИИММ ЛГУ],

Б. А. ФЕДотов

[ЛНПО «Красная заря», Ленинград]

УНИФИКАЦИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ МИКРО-ЭВМ НА БАЗЕ ВИРТУАЛЬНОЙ МАШИНЫ

Описываются архитектура и система команд виртуальной машины, ориентированной на реализацию алгоритмических языков высокого уровня со статическим контролем типов (АДА, АЛГОЛ, МОДУЛА 2, ПАСКАЛЬ и т. д.) на микро-ЭВМ.

1. Введение

В настоящее время используются сотни различных типов микро-ЭВМ с разнообразной архитектурой, причем многие из них имеют высокое быстродействие, но малый объем прямоадресуемой оперативной памяти. Создание для всех микро-ЭВМ трансляторов или кросс-трансляторов с алгоритмических языков высокого уровня (АЯВУ) файловых систем, текстовых редакторов и других инструментальных средств — очень трудоемкая задача. В этих условиях можно использовать следующий технический прием: создать одну унифицированную архитектуру и систему команд некоторой виртуальной (выдуманной) машины, все инструментальные средства ориентировать только на эту виртуальную машину (ВМ), а на всех используемых микро-ЭВМ реализовать интерпретаторы ВМ [1]. При этом, кроме унификации программного обеспечения, обычно удается добиться экономии оперативной памяти в 3—5 раз ценой замедления счета в 2—4 раза.

Сама по себе идея использования ВМ не нова. Известны простейшие стековые машины, созданные Н. Виртом для реализации языка Паскаль [2]. Позже он же создал более современную ВМ, ориентированную на язык МОДУЛА 2. Эта ВМ была даже аппаратно реализована в виде ЭВМ ЛИЛИТ [3]. Аналогичная работа выполняется в ВЦ СО АН СССР [4].

Первые ВМ имели примитивное строение для облегчения реализации интерпретатора при переходе на новую ЭВМ, например широко известный Р-код [2] включал в себя загрузку в стек и выгрузку из него в память, четыре арифметических действия, условные переход и несколько (меньше 10) вспомогательных команд. Хотя система команд ЛИЛИТ [3] содержала

уже около 250 команд, но большая часть кодов была отдана под короткие команды, упакованные в одном байте с адресами (например, загрузка в стек локальной переменной с адресом 0, 1, ..., 15, то же — для глобальной переменной и т. д.). Действие, единое при использовании языка высокого уровня, выполняется несколькими мелкими командами, с другой стороны, слишком сильно влияние одного языка (в данном случае — МОДУЛЫ 2), поэтому даже для родственных языков многие детали архитектуры неудобны.

Данная статья является продолжением и развитием работы [5], в которой была предложена архитектура и система команд ВМ, ориентированной на класс АЯВУ со статическим контролем типов (ПАСКАЛЬ, МОДУЛА 2, АЛГОЛ 68, АДА и им подобные). Оказалось, что для этого класса языков можно унифицировать представление основных типов данных, схемы распределения памяти, организацию вызовов процедур, работу с массивами. Благодаря четкому разделению обязанностей и ответственности между транслятором с АЯВУ в коды ВМ и интерпретатором ВМ на конкретной ЭВМ удалось упростить структуру ВМ (например, то, что может быть проконтролировано во время трансляции, никак не контролируется во время счета). Кроме того, резко упростился процесс трансляции за счет использования обратной польской записи в объектном коде, стека для вычисления, специальных команд для наиболее употребительных конструкций АЯВУ, ортогональности системы команд ВМ.

Под ортогональностью системы команд понимается единообразие типов команд и типов адресации, отсутствие исключений и нестандартных действий. Например, если в команде пересылки может быть девять типов источника и три типа получателя, то всего должно быть 27 команд пересылки. Если есть три типа команды сложения ($A+B$, $A+:=B$, $A[I]+:=B$), то такие же типы команд должны быть и для умножения, для логических команд, и т. д. На ЕС ЭВМ, скажем, генерация формул в любом трансляторе занимает большую часть, так как сложение целых выполняется одним способом, сложение десятичных — другим, а логическое сложение — третьим; команды *АН*, *SH*, *MH* есть, а *DN* (деление полуслов) — нет. Ясно, что при ортогональном построении системы команд могут появиться редко используемые команды, но на это нужно идти ради упрощения трансляции.

1.1. *Изменения.* По сравнению с [5] в описываемой ВМ изменились следующие аспекты.

а) Исключен регистровый стек. Вычисления арифметических выражений проводятся в стеке, совмещенном со статикой процедуры (т. е. памятью, в которой размещены идентификаторы и рабочие ячейки процедуры). Это позволило снять количественные ограничения на глубину стека, упростить схему вызова, уменьшить номенклатуру команд и размещать на вершине стека не только однословные значения.

б) Представление булевских и вещественных данных, кодировка литер и способ указания базовых регистров в описании ВМ не фиксируется. Считается, что для конкретного интерпретатора ВМ-машины должна производиться настройка транслятора за счет соответствующих сменных процедур.

в) Представление паспорта массива скрыто от транслятора. Концепция массива поддержана командами генерации массивов (возможно, многомерных), их присваивания, копирования, конструирования из отдельных элементов, вычисления границ массивов и адресов элементов. Таким образом, транслятор всегда оперирует со ссылкой на паспорт массива и не имеет

возможности оперировать с атрибутами паспорта в обход стандартных команд. В других известных ВМ работа с массивами поддержана слабо.

В настоящее время реализованы кросс-трансляторы с языка ассемблера ВМ, Паскаля и АЛГОЛА 68; ведутся работы по переносу этих трансляторов на ВМ методом раскрутки. Все точки, где нужно явно привязаться к архитектуре и ОС конкретной ЭВМ, явно выделены (таких точек очень немного). При переносе инструментального комплекса на новую ЭВМ их нужно перепрограммировать.

Во многих случаях использование ВМ выгодно не только по причинам унификации. Во многих задачах управления процессами и других задачах реального времени важна экономия оперативной памяти. Критическими же по времени являются не более 10% объема программ, поэтому в каждой разработке лучше использовать два кросс-транслятора: критичные по времени исполнения программы нужно транслировать непосредственно в коды спецЭВМ, а остальные программы — в коды виртуальной ЭВМ с последующим исполнением их через интерпретацию. Таким путем можно сэкономить сотни тысяч байтов оперативной памяти.

2. Основные элементы ВМ

Начнем с неформального описания архитектуры ВМ. Формальная модель, которая в настоящей статье не приводится, выполнена в виде реализации на языке АЛГОЛ 68 и может служить основной для реализации ВМ на различных объектных ЭВМ.

Архитектура ВМ включает в себя набор внутренних регистров и несколько типов памяти, а именно: программную память, рабочую память и стек вызовов.

2.1. Внутренние регистры ВМ-процессора. Имеется несколько регистров (указателей в программную и рабочую память, счетчиков), которые используются ВМ-процессором при интерпретации программного кода и непосредственно из ВМ-программы недоступны. К ним относятся: *G* — указатель на начало статике собственно программы; *L* — указатель на начало статике исполняемой в данный момент процедуры; *X* — указатель на конец текущей статике (вершину стека значений); *PC* — указатель на очередной байт интерпретируемого кода; *IR* — регистр исполняемой команды; *PN* — регистр номера текущей процедуры; *SP* — указатель вершины стека вызовов; *DT* — указатель вершины области массивов; *HT* — указатель вершины кучи; *LINE* — регистр привязки к программе на исходном языке.

В конкретной реализации ВМ некоторые регистры могут отсутствовать (например, *IR*, *LINE*) и могут быть введены дополнительные регистры.

2.2. Программная память. Эта память предназначена для хранения исполняемой ВМ-программы и доступна только для чтения. Функционально программная память делится на две части: интерпретируемый код и таблицу процедур.

Интерпретируемый код представляет собой однородный массив восьмибитовых байтов. Адресация интерпретируемого кода побайтовая. Никакого выравнивания ни для команд, ни для их операндов не требуется. Доступ к интерпретируемому коду осуществляется через указатель *PC*, который всегда указывает на очередной байт кода. При выборке этого байта происходит продвижение указателя на следующий байт.

Конкретные значения кодов команд на уровне описания ВМ не фиксируются. Если по каким-либо соображениям реализатора ВМ-интерпре-

татора не устраивает стандартная кодировка команд, он может выбрать другую.

Таблица процедур имеет один вход, на который может быть подано число в диапазоне от 0 до 255. Выходом таблицы является структура из трех полей: *EP* — адрес точки входа в процедуру (начальное значение *PC* для процедуры); *PL* — длина пакета параметров процедуры; *XLIM* — требуемый объем памяти в статике процедуры.

Таблица процедур используется командой вызова, которая получает в качестве аргумента номер процедуры и через таблицу осуществляет вход в процедуру.

Такое решение позволяет без изменений перемещать по памяти интерпретируемый код или его составляющие процедуры, настраивая очевидным образом таблицу процедур. Более того, можно организовать динамическую подкачку процедур во время интерпретации программы.

На уровне описания архитектуры ВМ не фиксированы разрядность полей таблицы процедур и способ ее организации (в виде единой таблицы или же в виде нескольких согласованных таблиц).

2.3. *Рабочая память.* В отличие от программной памяти, рабочая память используется не только для чтения, но и для записи.

Наименьшим адресуемым элементом рабочей памяти является шестнадцатибитовое слово. Вся память представляет собой линейный массив таких слов:

При реализации ВМ на машине с байтовой организацией памяти слово ВМ может занимать в реальной памяти два соседних байта с номерами $2N$ и $2N+1$, а в качестве адреса может использоваться число $2N$.

Диапазон адресов ограничивается шестнадцатибитовым представлением адреса. Существует лишь один выделенный адрес, а именно: число 0, обращение по которому вызывает прерывание. Этот адрес используется для представления значения *NIL*.

Биты в слове нумеруются числами от 1 до 16. Порядок нумерации битов фиксируется реализатором ВМ-интерпретатора и должен быть согласован с представлением целых чисел.

Содержимое слова данных может интерпретироваться четырьмя способами:

- 1) как величина типа *bits*, т. е. как 16 независимых битов;
- 2) как величина типа *bool*, при этом вся информация хранится в одном бите. Значения остальных пятнадцати битов не существенны; 0 в значащем бите — это *false*, 1 — это *true*, номер значащего бита фиксирует реализатор ВМ;
- 3) как целое число со знаком, представленное в дополнительном коде; при этом бит 1 — знаковый, бит 16 — бит единиц, бит 15 — бит веса 2 и т. д.;
- 4) как адрес слова, т. е. число без знака.

Кроме того, некоторые слова могут содержать служебную информацию (например, паспорта массивов), необходимую интерпретатору, ВМ-программа непосредственно со служебной информацией не работает (ВМ-программа — это программа в кодах ВМ).

Пары слов могут интерпретироваться как числа с плавающей точкой, представление таких чисел фиксирует реализатор ВМ-интерпретатора.

Способ интерпретации слов данных определяется командой, которая обрабатывает эти слова. Так как автором ВМ-программы, как правило, является транслятор с АЯВУ со статическим контролем типов данных, необходимость в дополнительном контроле за правильностью использования данных не возникает.

Рабочая память делится на четыре области: область статик, область массивов, кучу и строковый пул. Взаимное расположение этих областей определяет реализатор ВМ.

Область статик — это связный участок памяти. Имеются три указателя на эту область: G всегда указывает в начало статик собственно программы, L указывает в начало статик исполняемой в данный момент процедуры (при исполнении собственно программы L совпадает с G) и X указывает на конец текущей статик. Указатель L переставляется при вызовах процедур и возвратах из них. Указатель X используется для реализации стека в текущей статике. Он изменяется командами, работающими с вершиной стека, и, явно, командой *SETX*.

Работа со стеком происходит в обычном магазинном режиме, т. е. команды могут считывать значение с вершины стека и записывать значение в стек. При считывании значения указатель X уменьшается на длину значения, при записи увеличивается, кроме того, указатель X уменьшается на одно слово при использовании косвенной адресации вершины стека. Контроль за возможным переполнением стека внутри текущей статик возложен на автора ВМ-программы.

Адресация области статик в командах производится относительно указателей L или G . В этом случае однобайтовый операнд команды содержит в одном из битов признак локальной или глобальной (0 соответствует L , а 1 — G) адресации и в остальных семи битах — смещение в словах относительно L или G . Какой именно бит используется в качестве признака, зависит от реализации ВМ-интерпретатора.

Динамический контроль за возможным исчерпанием области статик возложен на ВМ, а именно: при исполнении команды вызова ВМ из таблицы процедур получает информацию о требуемом объеме памяти в статике вызываемой процедуры. Если свободного места в области статик достаточно, то ВМ продолжает работу, в противном случае происходит АВОСТ.

Область массивов предназначена для динамически создаваемых объектов, время жизни которых ограничено, например, для массивов, создаваемых командой *GENR*. При выходе из подпрограммы или блока, где был захват памяти в области массивов, эта память освобождается, но будет ли она использоваться повторно, зависит от реализации ВМ.

В отличие от области массивов, память, захваченная в куче, не освобождается до конца интерпретации.

В строковом пуле размещаются строковые константы во время загрузки ВМ-программы. Память, отведенная для них, не освобождается до окончания интерпретации.

Значение любого слова из области статик, области массивов, кучи и строкового пула может быть считано по его абсолютному адресу.

2.4. *Стек вызовов.* Этот стек используется командами вызова процедуры, входа в блок и структурное предложение, организации цикла. Он служит для сохранения и восстановления указателей текущей статик (L) или вершины стека значения (X); адреса возврата из процедуры (PC) или адреса выхода из структурного предложения.

Структура ячейки стека вызовов определяется реализатором ВМ-интерпретатора и должна включать поля для сохранения указателей L или X , PC .

Работа со стеком вызовов происходит в основном в традиционном стиле. Кроме того есть две команды одна из которых сбрасывает одну позицию со стека вызовов, другая опустошает его целиком.

3. Система команд

Система команд ВМ охватывает все основные конструкции класса статических АЯВУ. Она включает следующие группы команд: пересылки слова; пересылки группы слов; арифметические и логические операции над словами; вещественную арифметику; управление исполнением; работу с массивами.

Команды ВМ записываются традиционным образом: один байт занимает код команды, за кодом могут следовать один и более операндов. Типы операндов определяются кодом команды. Операнд может состоять из несколько подряд идущих байтов. Операндом команды может быть: однобайтовое смещение относительно указателя L или G , задающее адрес; одно-, двух- или четырехбайтовый непосредственный операнд; одно- или двухбайтовое смещение относительно PC , определяющее метку в программном коде; однобайтовый спецификатор длины значения; однобайтовый спецификатор количества операндов команды.

3.1. *Примеры команд.* Рассмотрим в качестве первого примера команды $LOOP$ и OD , предназначенные для организации циклов со счетчиком. Команда $LOOP$ имеет четыре операнда: F — целое число, нижний предел цикла, B — целое число, шаг цикла; T — целое число, верхний предел цикла, END — метка, адрес выхода из цикла.

Последний операнд всегда считывается из ВМ-программы. Первые три операнда могут либо считываться со стека, либо принимать значение по умолчанию, так что всего имеется восемь разновидностей команды $LOOP$; по умолчанию $F=1$, $B=1$.

При исполнении команды $LOOP$ на вершине стека образуется область управления циклом (ОУЦ), занимающая три слова. Первое из них содержит текущее значение переменной цикла. Два других слова непосредственно из программы не адресуются, они должны содержать служебную информацию, необходимую команде OD для повторения или завершения цикла, например значения B и T . Вместо T может храниться K — оставшееся количество повторений цикла. В момент исполнения команды $LOOP$

$$K := (T - F) \% B + 1 \quad (\text{при } B \neq 0).$$

Если $K \leq 0$, то управление сразу передается на метку END . Для обеспечения работы команды OD команда $LOOP$ также кладет на стек возвратов текущие значения указателей X и PC .

Команда OD , используя значение X со стека возвратов, получает доступ к ОУЦ и определяет, следует цикл повторять или нет. В первом случае модифицируется значение переменной цикла и, при необходимости, значения остальных слов ОУЦ. После этого управление передается по значению PC , находящемуся на стеке возвратов, при этом ни значение PC , ни значение X со стека возвратов не снимаются. При завершении цикла управление передается на метку END , являющуюся операндом команды $LOOP$. Адрес команды $LOOP$ определяется по значению PC из стека возвратов, после чего PC и X снимаются со стека возвратов.

Отметим, что в отличие от АЯВУ команда OD не является синтаксически привязанной к команде $LOOP$. Привязка осуществляется только динамически. Это позволяет, например, реализовать цикл вида

<u>for</u>	<i>I</i>	<u>from</u> <i>F</i> <u>by</u> <i>B</i> <u>to</u> <i>T</i>
<u>do</u>		
	<u>if</u>	условие
	<u>then</u>	ветка 1
	<u>else</u>	ветка 2
	<u>fi</u>	
<u>od</u>		

в виде следующей ВМ-программы:

```

                                <вычисление F>
                                <вычисление B>
                                <вычисление T>
                                LOOP.. END
                                <условие>
                                BRF. ELSE
                                <ветка 1>
                                OD
ELSE:                          <ветка 2>
                                OD
                                END

```

Здесь *LOOP..* — разновидность команды *LOOP* без значений по умолчанию, *BRF.* — команда условного (по значению false на вершине стека) перехода.

В качестве другого примера рассмотрим команды для работы с массивами. Имеется 10 таких команд.

Команды *GENR* и *HGENR* служат для генерации массивов (*GENR* — в области массивов, *HGENR* — в куче). Обе команды имеют три однобайтовых операнда в тексте ВМ-программы: *LEN* — целое число, длина элемента массива, *N* — целое число, размерность массива, *K* — целое число, количество генерируемых массивов.

Кроме того, со стека снимаются $2N$ чисел — граничных пар по каждой размерности. После исполнения команд на стек кладутся *K* адресов паспортов массивов. Структура паспортов на уровне описания ВМ никак не регламентируется. Паспорт должен давать доступ к информации, необходимой для работы остальных команд.

Команда *SL* по адресу паспорта массива и индексом элемента вычисляет адрес элемента, при этом контролируется невыход значений индексов за пределы диапазона, определяемого граничными парами. Имеется 14 разновидностей команды *SL*, которые могут брать адрес массива со стека или области статик, а индексы — со стека или (для одно- и двумерных массивов) из области статик.

Команда *ASSR* осуществляет присваивание массивов, при этом контроль за совпадением размерностей и длин элементов массивов возложен на автора ВМ-программы, а проверка совпадения границ осуществляется ВМ.

Команды *LWB* и *UPB* вычисляют границы массивов по адресу паспорта и номеру измерения. Команда *COPYR* создает копию массива.

Перечисленные выше команды допускают массивы произвольной размерности. Имеются три специальные команды для работы с одномерными массивами.

Команда *ROW* создает одномерный массив с границами $1:N$ и заполняет его N значениями, снимаемыми со стека; значение N и длина элементов массива — операнды из ВМ-программы.

Команда *CMPR* сравнивает одномерные массивы из однословных элементов, причем границы массивов могут не совпадать. В результате работы команды на стек кладутся два «представителя» сравниваемых массивов. Это могут быть их длины, если один массив является началом другого, или первые несовпадающие элементы массивов. Основное назначение команды — лексикографическое сравнение строк.

Команда *STRING* с одним однобайтовым операндом в тексте ВМ-программы создает одномерный массив литер. Значение операнда — это номер строки литер в специальном строковом пуле. Каждая процедура и собственно программа имеют независимую нумерацию строк.

4. Количественные оценки

Как уже говорилось, одной из основных целей использования ВМ является существенное уменьшение длины командного кода, эта характеристика особенно важна для программного обеспечения микро-ЭВМ. Чтобы получить достаточно достоверные соотношения, сравнение проводилось на большом пакете рациональной арифметики (47 процедур на АЛГОЛЕ 68 общим объемом 242 строки исходного текста). При трансляции этого пакета в коды ЕС ЭВМ, СМ 4 и ВМ были получены объектные коды общим объемом 4368, 3074 и 836 байтов соответственно, т. е. код ВМ короче в 3,7 раза, чем СМ 4, и в 5,2, чем ЕС. Отметим дополнительно, что эти цифры получены при отключенном контроле индексов для ЕС и СМ 4, а в ВМ контроль встроенный. Если, например, на ЕС потребовать контроль индексов, то получится код длиной 6696 байтов, т. е. в восемь раз длиннее кода ВМ.

СПИСОК ЛИТЕРАТУРЫ

1. *Stramm B., Hughes D.* A virtual machine design for nest free programming // *INFOR*. 1986. V. 24. № 1. P. 45–58.
2. *Nori K. V.* The PASCAL (*p*) compiler implementation notes // *Institute für Informatik. Eidgenössische technische Hochschule, Zurich. Tech. Rep. July 1976.*
3. *Wirth N.* Personal computer LILITH // *Zurich: Swirss Federal Institute of Technology Rep. 40. 1981.*
4. *Кузнецов Д. Н., Тарасов Е. В., Недоря А. Е.* Процессор КРОНОС как транспьютерный узел макета системы «МАРС». Препринт. Новосибирск: ВЦ СО АН СССР, 1987.
5. *Магиясевич Ю. В., Терехов А. Н.* 16-разрядная ЭВМ, ориентированная на АЯВУ // Программирование микропроцессорной техники (сб. трудов целевой подгруппы по технологии программирования микропроцессорной техники). Таллин: Ин-т кибернетики АН ЭССР, 1984. С. 68–72.

Поступила в редакцию 05.11.88