Бурцев А. А., Рамиль Альварес Х.

Реализация средств объектно-ориентированного программирования в кросс-компиляторе языка ДССП-Т

Введение

В НИЛ троичной информатики ВМК МГУ (ранее НИЛ ЭВМ) созданы программный комплекс ТВМ (Троичная Виртуальная Машина) [1], имитирующий функционирование современного варианта троичного процессора двухстековой архитектуры, а также ДССП-ТВМ [2] – система разработки программ для ТВМ на языке ДССП-Т – троичном варианте языка ДССП (Диалоговой Системы Структурированного Программирования) [3-7].

ДССП – интерпретируемая система программирования, а язык ДССП-Т реализуется кросс-компилятором. Увы, не все возможности, обеспечиваемые интерпретатором, удаётся реализовать кросс-компилятором. Поэтому выразительные средства языка ДССП-Т приходится существенно ограничивать (по сравнению с языком обычной ДССП).

Так, например, интерпретатор ДССП позволяет определить в программе новое слово, а затем сразу же его исполнить. Это даёт возможность определять в ДССП-программе особые слова, так называемые слова-компиляторы, которые можно тут же применять для формирования новых слов с компиляцией их тел. А кросс-компилятор, увы, не позволяет определять в программе такие компилирующие слова, которые могли бы сразу же исполняться в момент компиляции самой этой программы.

Но как раз именно такая возможность использовалась ранее в интерпретаторе ДССП в качестве инструмента построения новых типов данных. Без неё язык ДССП-Т (версии 2d) тяготился серьёзными недостатками в отношении типов данных. Во-первых, его бедный ассортимент встроенных типов данных (TRYTE TWORD) позволял оперировать только данными машинного уровня (трайтами и троичными словами). И во-вторых, что более досадно, в нём не было возможности построения новых типов данных, структурирования сложных высокоуровневых типов данных из имеющихся простых.

На ликвидацию этих недостатков языка ДССП-Т и была нацелена представляемая работа. В качестве поставленной задачи необходимо было обеспечить в кросс-компиляторе языка ДССП-Т

средства построения новых типов данных, эквивалентные возможностям объектно-ориентированного программирования (ООП).

Поскольку механизм обеспечения ООП-средств, предложенный ранее для интерпретатора ДССП/С [8,9], напрямую реализовать в кросс-компиляторе не представлялось возможным, предстояло разработать иной способ решения поставленной задачи, сохранив при этом совместимость предложенных ООП-средств на уровне языка ДССП.

Далее в статье сначала представляются предложенные в ДССП ООП-средства построения новых типов данных и рассматриваются особенности их реализации в ДССП-интерпретаторе. А затем анализируются проблемы возможной реализации таких ООП-средств кросс-компилятором ДССП-ТВМ и предлагаются способы их разрешения.

1. Средства построения новых типов данных в ДССП

Встраиваемые в язык ДССП-Т средства построения новых типов данных, эквивалентные возможностям ООП, были сначала предложены и реализованы для версии ДССП-32р, а затем и для версии ДССП/С [8], ядро которой в целях обеспечения мобильности системы разрабатывалось на языке Си. Синтаксические языковые конструкции этих ООП-средств, а также механизм их реализации в ДССП-интерпретаторе подробно описаны в [9]. Здесь лишь кратко поясним их так, чтобы можно было ознакомиться далее с проблемами их реализации в кросс-компиляторе.

Для объявления нового типа в ДССП предлагаются две конструкции: структура и класс.

Объявление нового типа как структуры начинается словом **STRUCT:** и завершается словом **;STRUCT** , а располагаемые между ними определения объектов данных (с помощью слов **VAR VCTR ARR**) объявляют в этом случае не самостоятельные переменные и массивы, а поля определяемой структуры:

```
STRUCT: <имя_структуры> {имя типа новой структуры}
 VAR <имя_поля>... VAR <имя_поля> {поля структуры}
;STRUCT {вместо VAR можно использовать VCTR, ARR }
```

Пример объявления нового типа для работы с величинами-датами:

```
STRUCT: TDATE {тип для представления даты}
 VAR .Year  VAR .Month  VAR .Day
;STRUCT
```

После объявления нового типа данных можно использовать его имя для объявления объектов (переменных и массивов) такого типа:

```
TDATE VAR Date  TDATE 9 VCTR Dates
```

Имеющиеся в языке ДССП-Т префиксные операции над переменными и массивами можно теперь применять и для доступа как ко всему объекту типа структуры целиком:

```
Date 5 ! Dates {копирование даты Dates[5]:=Date }
```

так и к отдельным его полям:

```
Date {'Date} .Year {Date.Year}
5 Dates ! .Year {Dates[5].Year:= Date.Year}
```

Ещё одним средством построения новых типов данных в языке ДССП-Т являются классы. Объявление нового типа как класса начинается со слова **CLASS:** и заканчивается словом **;CLASS** , а внутри такого объявления можно задавать не только поля (как и при объявлении структуры), но и методы класса:

```
CLASS: <имя_класса> {имя типа нового класса}
 VAR <имя_поля>... VAR <имя_поля> {объявление полей}
  METHOD <имя_метода> {объявление методов}
  METHOD= <прежнее_имя_метода> <новое_имя_метода>
 N METHOD# <новое_имя_метода> {N-номер прежнего метода}
;CLASS
```

Словом **METHOD** объявляется новая операция, которую можно будет совершать над объектом этого типа-класса помимо тех методов (префиксных операций **!** и **‘**), которые над ним уже предусмотрены изначально (как и над структурами). Слово **METHOD=** позволяет назначить новое имя методу, используя его прежнее имя, а слово **METHOD#** – задав номер метода.

Пример объявления типа-класса для работы с очередью:

```
:: CLASS:  QUEUE  { Класс ОЧЕРЕДЬ }
  {---  поля данных, составляющих очередь ---}
    VAR .cnt { кол-во элементов в очереди }
    VAR .n   { индекс-указатель начала очереди в массиве }
    VAR .k { индекс-указатель конца очереди в массиве }
 100 VCTR .MI { массив для хранения элементов очереди }
  {---  методы, представляющие операции над очередью ---}
 :: METHOD Init { инициировать, начать работу очереди }
 :: METHOD Show { показать на экране состояние очереди }
 :: METHOD Get { взять элемент из очереди на вершину стека }
 :: METHOD Put { поместить элемент из вершины стека в очередь}
 :: METHOD Empty? { проверить, пуста ли очередь }
 :: METHOD Full? { проверить, полна ли очередь }
 :: METHOD Done { завершить, закончить работу очереди }
 :: METHOD= ! => {новое имя метода для копирования очереди}
;CLASS
```

Для назначения процедуры, реализующей объявленный метод класса, предлагаются синтаксические конструкции:

```
<имя_класса> :M: <имя_метода> <тело_реализации_метода>... ;
<имя_класса> :M= <имя_метода> <имя_процедуры_метода>
```

Имя типа класса следует задать перед употреблением слова **:M:** (или **:M=**). Слово **:M:** создаёт тело новой процедуры, компилируя его как и слово **:** (до **;**), а слово **:M=** принимает имя уже существующей процедуры, чтобы назначать её в качестве исполнителя метода.

Примеры реализации некоторых методов класса QUEUE :

```
QUEUE :M: Init {Q} 0 C2 ! .cnt { Q.cnt:=0 }
       0 C2 ! .k 1 E2 ! .n { } ; { Q.k:=0; Q.n:=1 }
QUEUE :M= Empty? .Empty?
      : .Empty? {Q} .cnt 0 <= {1/0} ;
QUEUE :M: Get {Q}
          C .Empty? IF+ Empty! {проверка искл.ситуации}
          C .n C2 .MI E2 { y:= Q.M[Q.n] }
    {y,Q} C .cnt 1- C2 ! .cnt { Q.cnt:=Q.cnt-1 }
          C .n +1mod E2 {y,n+1,Q} ! .n
             {Q.n:=(Q.n+1)mod Maxlen } {y} ;
```

Примеры объявлений объектов типа QUEUE :

```
 QUEUE VAR Q    { Q - отдельный экземпляр класса QUEUE }
 9 QUEUE VCTR VQ { массив очередей VQ[0:9] }
```

Примеры вызовов действий с объявленными объектами:

```
{вызов операций по имени объектов:} Q{адрес Q} 3 VQ {‘VQ(3)}
Q .cnt {Q.cnt}{получение значения кол-ва элементов очереди }
0 Q ! .cnt {Q.cnt:=0}{обнуление счетчика элементов очереди}
3 VQ ! Q { Q:= VQ[3] копирование очереди целиком }
3 VQ => Q {то же, но применяя новое обозначение метода }
Init Q { вызов метода для инициализации очереди Q }
77 Put Q { помещение в очередь Q числа 77 }
3 Get VQ {y} { взятие значения (y) из очереди VQ[3] }
```

Внутри объявления нового типа как класса с помощью слова **SUBCLASS** можно задать, какой уже известный тип использовать для наследования:

```
CLASS: <имя_класса> { имя типа нового класса }
 SUBCLASS <имя_наследуемого_класса>
 VAR <имя_поля>... VAR <имя_поля> {объявление новых полей}
 METHOD <имя_метода> {объявление новых методов}
 METHOD= <прежнее_имя_метода>    <новое_имя_метода>
 N METHOD# <новое_имя_метода> {N–номер прежнего метода}
;CLASS
```

Такое наследование означает, что у нового объявляемого типа изначально считаются уже заданными все поля и методы, которыми обладает наследуемый тип, после чего можно добавлять к его определению новые поля, задавать новые методы, а также переопределять исполнителей прежних методов.

Пример объявления класса-наследника:

```
{--- Объявление класса ДЕК ---}
:: CLASS:  DEQ  SUBCLASS QUEUE { наследует класс QUEUE }
 {--- методы , дополняющие операции над очередью ---}
 :: METHOD Put_ { поместить элемент в начало очереди }
 :: METHOD Get_ { взять элемент с конца очереди }
 :: METHOD Count { узнать количество элементов в очереди }
;CLASS
```

2. О реализации ООП-средств в ДССП-интерпретаторе

Определение нового типа данных (см. пример QUEUE) в ДССП с помощью компилирующих слов:

STRUCT: ;STRUCT CLASS: SUBCLASS METHOD ;CLASS

предполагает создание в словаре:

- слова с именем нового типа данных: QUEUE ;
- слов с именами новых полей: .cnt .n .k .MI;
- слов с именами новых методов:

```
Init Show Get Put Done => Empty? Full? Done.
```

Требуется построить тела для всех этих слов, а также:

- тело дескриптора для нового типа данных;
- тела процедур-исполнителей новых методов.

Слова с именами новых полей и их тела (дескрипторы) формируются почти так же, как это делается для обычных переменных. Но для полей используется иная функция вычисления адреса, которая использует значение вершины стека как базовый адрес структуры, добавляя к нему смещение, хранимое в дескрипторе данного поля.

Слова новых методов помечаются P-флагом как префиксные операции, а их тела формируются согласно единому образцу вида:

```
CALL_ExecMethod {адрес универсальной процедуры вызова метода}
.tword 8 { номер метода}
```

Строение тела слова имени типа (QUEUE) эквивалентно процедуре назначения системной переменной DTYPE адреса дескриптора типа QUEUE'Ptr :

```
: QUEUE QUEUE'Ptr ! DTYPE ;
```

Строение дескриптора типа детально изображено на рис.1. Оно состоит из полей, задающих адрес дескриптора наследуемого типа (**PredokPtr**),

количество методов (**MtdNum**), размер объекта (**Size**), за которыми располагается таблица методов, содержащая коды команд или адреса для вызова процедур-исполнителей методов.

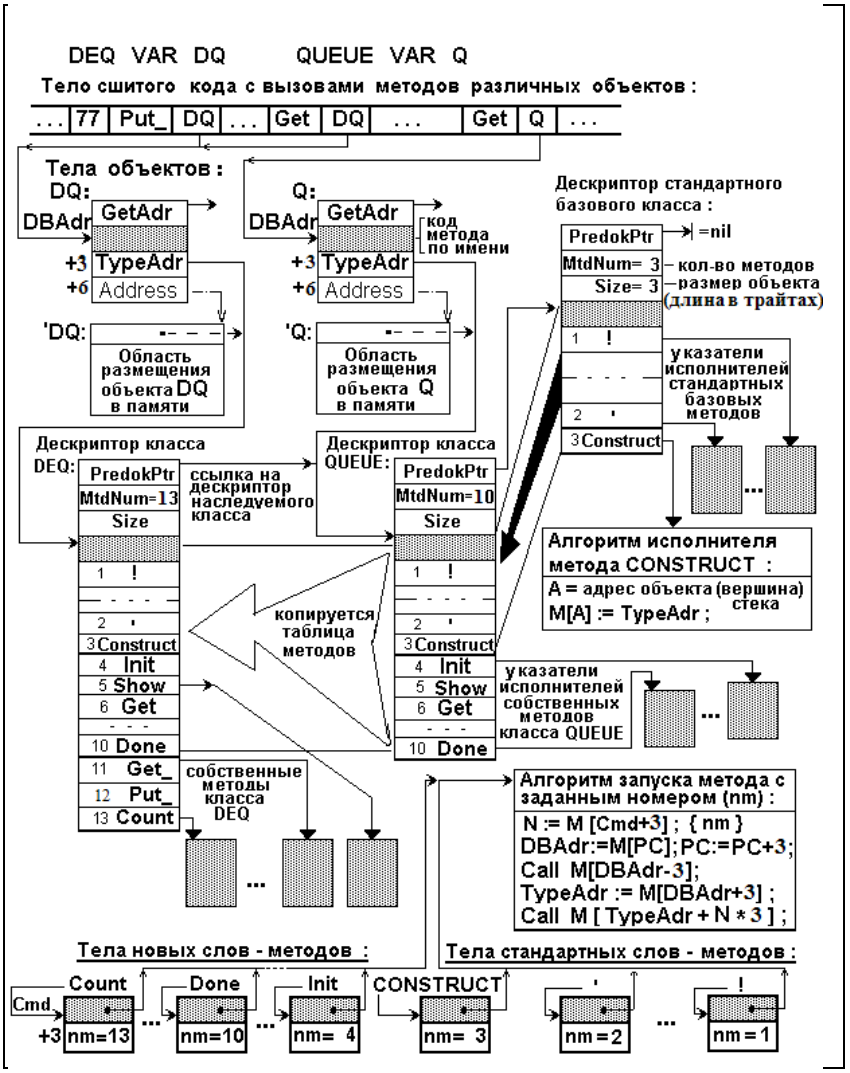

Рисунок 1. Механизм реализации методов класса

На рис.1 наглядно показано, как осуществляется вызов метода над объектом класса, какие при этом используются структуры данных (дескрипторы объектов-переменных, дескрипторы типов).

Для построения дескриптора нового типа (например, DEQ) необходимо сначала заполнить поля PredokPtr, NtdNum, Size, затем скопировать таблицу методов у дескриптора наследуемого типа, после чего дополнить эту таблицу ссылками на тела процедур-исполнителей новых методов.

Все описанные действия по построению необходимых структур данных достаточно легко осуществить в ДССП-интерпретаторе, т.к. во время его функционирования ему полностью доступна вся память, используемая ДССП-программой. А вот чтобы подготавливать требуемые структуры данных в ассемблерном коде, последовательно формируемом кросс-компилятором, потребовалось решить ряд проблем и существенно модифицировать работу кросс-компилятора.

3. О проблемах ООП-реализации в кросс-компиляторе

1. Компилятор функционирует в цикле обработки потока известных ему слов-компиляторов, для каждого из которых у него заранее заготовлена своя Си-процедура. Как же “научить ” компилятор понимать новые компилирующие слова, представляющие имена вновь определённых типов (например, QUEUE, DEQ) ? (проблема №1)

2. Компилятор последовательно (за один проход!) генерирует ассемблерный код, отправляя его в выходной файл. Как обеспечить копирование дескриптора наследуемого класса ? (проблема №2)

3. Дескриптор требуемого типа как и номер метода теперь невозможно напрямую взять из тела (как ранее), т.к. сформированные тела в ассемблерном коде помещаются в выходной результирующий файл. Как же узнать номер метода при определении процедуры его исполнения? (проблема №3)

4. Модификация кросс-компилятора для реализации средств построения новых типов данных

Рассмотрим, какие модификации пришлось сделать в кросс-компиляторе для разрешения этих проблем.

4.1 Дополнительный словарь имён типов

Для слов, представляющих имена новых типов, образуется дополнительный словарь, в каждом узле которого содержатся:

- 1) строка **IName** слова имени типа;
- 2) индекс **ICode** для доступа в таблицу к телу его дескриптора;
- 3) строка **ILabel** метки для ассемблерного тела дескриптора.

При этом изменяется алгоритм обработки компилирующих

слов: если встреченное компилирующее слово не найдено в основном словаре, то компилятор пытается найти это слово в дополнительном словаре типов. Если найдено слово типа, то его номер фиксируется в спецпеременной **nTYPE** так, чтобы использовать этот номер типа для последующих объявлений данных (переменных и массивов).

Когда в программе объявляется новый тип данных (см. например, объявление класса QUEUE) компилятор создаёт новое слово (QUEUE) и в своём словаре типов, и в ДССП-словаре компилируемой программы, формируя для него такое ассемблерное тело в выходном файле, чтобы при его вызове спецпеременная **DTYPE** получала значение адреса дескриптора этого типа:

```
call_LIT  ; tword TN1008  {{ '' QUEUE'Ptr
DPushI_42 ; Store         {{ 'DTYPE !W
RET                       {{ ;
```

При этом сам дескриптор типа формируется компилятором в отдельной области памяти (называемой далее таблицей дескрипторов типов) и будет выводиться в файл ассемблерного кода вместе с назначенной ему меткой (**ILabel**=TN1008) лишь по завершении компиляции всей ДССП-программы.

4.2 Таблица дескрипторов типов

Такая таблица создаётся в компиляторе как массив (**TypeBody**) значений, которые должны представлять тела дескрипторов объявленных типов сначала в компиляторе, а затем и в ассемблерном коде формируемой программы. Индекс-ссылка (**ICode**) из каждого узла слова имени типа указывает на расположение тела его дескриптора в этом массиве.

При наследовании типа его дескриптор копируется внутри этого массива. Затем в нём изменяются поля счётчика методов и размера объекта, а в его таблицу методов заносятся номера слов-исполнителей методов.

После завершения компиляции ДССП-программы итоговое содержимое массива отправляется в файл ассемблерного кода с учётом того, что элементы массива (long int) имеют разный смысл (см. табл.1). Заметим, что тело дескриптора нельзя выдавать в ассемблерный файл сразу же после его создания, поскольку при дальнейшей компиляции ДССП-программы в его таблице методов ещё будут корректироваться ссылки на тела процедур, назначаемых в качестве исполнителей этих методов. В массиве такими ссылками служат номера слов, а в ассемблерный код помещаются соответствующие их словам мнемокоды команд или метки, которые превращаются далее в адреса тел (или команды вызова) процедур-исполнителей методов.

<table>
<tr><td></td><td>в массиве:</td><td>на ассемблере:</td><td></td></tr>
<tr><td>-3</td><td>номер типа-предка</td><td>.tword TN1007</td><td>адрес дескриптора-предка</td></tr>
<tr><td>-2</td><td>счётчик методов</td><td>.tword 10</td><td>числовое значение</td></tr>
<tr><td>-1</td><td>размер объекта</td><td>.tword 315</td><td>числовое значение</td></tr>
<tr><td>0</td><td rowspan="7">Таблица методов: содержит номера слов процедур, назначенных исполнителями методов</td><td>TN1008: NOP</td><td rowspan="7">Таблица методов: содержит ассемблерные коды команд исполнителей методов или команды (адреса) вызовов их процедур</td></tr>
<tr><td>1</td><td>call_StructCopy</td></tr>
<tr><td>2</td><td>NOP</td></tr>
<tr><td>3</td><td>call_PN2232</td></tr>
<tr><td>...</td><td>...</td></tr>
<tr><td>10</td><td>call_PN2373</td></tr>
</table>

Таблица 1. Строение тела дескриптора типа

4.3 Поле номера метода в узле слова ДССП-словаря

Номера слов, назначаемых в качестве исполнителей методов, требуется вносить в таблицу методов дескриптора типа по мере определения тел их процедур: `QUEUE :M: Get ... ;`

Для этого необходимо по имени метода (`Get`) узнавать его номер. В интерпретаторе для этого достаточно “заглянуть” в память на второе слово тела метода (см. рис.1). А в кросс-компиляторе для решения этой проблемы (№3) потребовалось добавить дополнительное поле в узел слова ДССП-словаря, чтобы хранить в нём номер метода вместе с флагом-признаком метода.

Это позволило получить возможность корректировать поля таблицы методов в дескрипторе типа, применяя такой алгоритм:

1) после выполнения слова с именем типа `QUEUE` спецпеременная **nTYPE** получила номер слова-типа nTYPE:= nQUEUE;
2) по этому номеру из узла словаря типов получаем ссылку ICode на его дескриптор в массиве TypeBody;
3) по имени метода (Get) получаем (из поля его узла ДССП-словаря) номер метода **nMtdGet**;
4) изменяем элемент таблицы методов в массиве тел дескрипторов TypeBody[ICode+nMtdGet], записывая в него номер слова, назначенного в качестве исполнителя метода QUEUE::Get.

Заключение

В результате осуществлённой модификации кросс-компилятора в языке ДССП-Т (версии 3а) обеспечены возможности построения новых типов данных, эквивалентные средствам объектно-ориентированного программирования.

На их основе в ДССП-библиотеке были определены новые типы данных: **ACT** для объявления процедурных переменных; **TSET** для работы с нумерованными тритами троичного слова; а также **TCLASS** как базовый тип для всех классов с методом **CONSTRUCT**.

Литература

1. 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.
2. 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.
3. Брусенцов Н.П., Златкус Г.В, Руднев И.А. ДССП - диалоговая система структурированного программирования. // Программное оснащение микрокомпьютеров. М.: Изд-во МГУ, 1982, с.11-40.
4. Брусенцов Н.П., Захаров В.Б., Руднев И.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.
5. Сидоров С.А., Шумаков М.Н. ДССП как открытая система. // Дискретные модели. Анализ, синтез и оптимизация. Спб.: СпбГУ, 1998. с.191-201.
6. 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.
7. 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.
8. Бурцев А.А., Франтов Д.В., Шумаков М.Н. Разработка интерпретатора сшитого кода на языке Си. // Вопросы кибернетики. Сб. статей под ред. В.Б.Бетелина. М., 1999. с.64-76.
9. Бурцев А.А., Рамиль Альварес Х. Средства объектно-ориентированного программирования в ДССП. // Программные системы и инструменты. Тематический сборник №4, М.: Изд-во факультета ВМК МГУ, 2003. с.166-175.

Опубликовано: Программные системы и инструменты № 13. Под ред. Л.Н. Королева. – М.: Издательский отдел ВМиК МГУ, 2012. С. 27-36.