Gem #140: Устранение разрыва порядка байтов. Как преодолеть Большой Байтовый Раздел?

Автор: Thomas Quinot — AdaCore

Краткое содержание: Два важнейших вопроса мучают всех разработчиков всю жизнь:

- Разбивать яйца всмятку с твердого или мягкого конца?
- Сохраняя многобайтовое значение, начинать с наиболее важного байта, или с наименее важного байта?

Более тридцати лет назад Danny Cohen (Дэнни Коен) заключил мир между большими эндианами и маленькими эндианами (с прямым порядком байтов и с обратным порядком байтов), но сфера программного обеспечения до сих пор кишит кодом, обменивающим байты только для того, чтобы обработать данные, производимые системами различного поведения. Могут ли языки программирования прийти на помощь?

Давайте начнём…

Этот Gem представляет новый определенный реализацией атрибут, представленный в GNAT комплекта инструментов Ada, который имеет целью упрощать мирное сосуществование применений данных с прямым порядком байтов и с обратным порядком байтов. Это особенно важно, например, при перенастройке приложения от устаревшего оборудования с помощью данного порядка байтов для другой платформы различного порядка байтов. Если какие-либо данные хранились в памяти или персистентном устройстве хранения данных унаследованной системой, или если совместимость с другими подсистемами должна быть сохранена, необходимо для всех структур данных иметь точно то же представление на этих двух платформах.

Рассмотрим следующую двухбайтовую структуру данных и предложение представления записи:

```
subtype Yr_Type is Natural range 0 .. 127;
subtype Mo_Type is Natural range 1 .. 12;
subtype Da_Type is Natural range 1 .. 31;

type Date is record
   Years_Since_1980 : Yr_Type;
   Month            : Mo_Type;
   Day_Of_Month     : Da_Type;
end record;

for Date use record
   Years_Since_1980 at 0 range 0  ..  6;
   Month            at 0 range 7  .. 10;
   Day_Of_Month     at 0 range 11 .. 15;
end record;
```

Представление "12-го декабря 2012" в системе с обратным порядком байтов следующие:

0100000 1	100 01100
yyyyyyy m	mmm ddddd
65	140

В системе с прямым порядком байтов та же дата представлена как:

0 0100000	01100 110
m yyyyyyy	ddddd mmm
32	102

Можно полагать, что стандартный атрибут `Bit_Order` разрешит эту ситуацию. Давайте попробуем и внесем поправки в декларацию выше с помощью:

```
for Date'Bit_Order use System.High_Order_First;
```

явное значение по умолчанию (говорят, что оно подтверждает), и больше не влияет.

Однако в системе с прямым порядком байтов результат не является ожидаемым. Чтобы понять почему, нужно взглянуть на то, как стандарт Ada определяет толкование положений представления записей.

Когда порядок битов является порядком битов по умолчанию, растущие смещения битов просто соответствуют вводу последовательных (растущих) адресов в памяти. Но когда битовый порядок имеет противоположное значение, биты нумеруются «в обратном направлении» относительно способа хранения данных в виде последовательных байтов, поэтому необходимы дополнительные правила, чтобы знать, на какие биты нам нужно смотреть.

Важно помнить, что смещения битов для компонента в предложении представления записи всегда относятся к некоторому целочисленному значению (называемому «машинными скалярами» в RM Ada), из которого значение компонента извлекается с использованием сдвига и маски. операция.

Чтобы выяснить, к какому машинному скаляру принадлежит данный компонент, вы должны сначала определить набор компонентов, которые имеют одинаковое смещение байтов. В нашем примере это будут все три компонента, так как все они указаны с байтовым смещением 0. Далее рассмотрим самое высокое битовое смещение для всех этих компонентов. Здесь - 15. Размер машинного скаляра - следующая большая степень двух: 16 в этом примере. Таким образом, это означает, что все три компонента в нашей записи являются частью скаляра одной машины, который представляет собой двухбайтовое целое число размером 16 бит. Если вы находитесь на машине с прямым порядком байтов, младший байт этого скаляра машины всегда тот, который хранится по младшему адресу, а старший бит - тот, который хранится по старшему адресу. И важно отметить, что это не зависит от порядка битов, указанного для структуры данных.

Итак, если вы находитесь в системе с прямым порядком байтов и задаете порядок битов High_Order_First, двухбайтовое машинное скалярное значение будет:

0100000110001100
yyyyyyymmmmddddd

который при хранении в виде двух последовательных байтов в памяти будет соответствовать:

10001100	01000001
140	65

Интересно отметить, что это отличается и от собственного представления с прямым порядком байтов и от собственного представления с обратным порядком байтов. Поэтому порядок, в котором байты, которые составляют скаляры машины, записаны в память, не изменяется атрибутом Bit_Order - только индексы битов в скалярах машины изменяются.

Теперь введите Scalar_Storage_Order

Именно для того, чтобы преодолеть это ограничение языка, в GNAT был введен новый атрибут `Scalar_Storage_Order`. Эффект этого атрибута заключается в точном переопределении порядка байтов в машинных скалярах для данного типа записи. Итак, давайте добавим еще одно определение атрибута:

```
for Date'Scalar_Storage_Order use System.High_Order_First;
```

Это означает, что байты, составляющие скаляры машины, должны быть заменены при хранении в памяти. Мы видим, что тогда представление памяти становится (65, 140): теперь оно соответствует нативному представлению на платформе с прямым порядком байтов.

Таким образом, существующий код для системы с прямым порядком байтов можно перенести в систему с прямым порядком байтов без суеты и без каких-либо изменений в представлении данных, просто добавив соответствующие определения атрибутов в соответствующие объявления типов записей.

Совместимость с устаревшими наборами инструментов

При ретаргетинге приложения с изменением порядка байтов удобно использовать атрибут `Scalar_Storage_Order`, чтобы новая платформа использовала то же представление данных, что и старая. Тем не менее, вы все равно можете захотеть скомпилировать ваше приложение для вашей старой цели с помощью устаревшей цепочки инструментов, которая может не поддерживать новый атрибут. В этом случае вы можете указать его, используя альтернативный синтаксис:

```
pragma Attribute_Definition
   (Scalar_Storage_Order, Date, System.High_Order_First);
```

Более старые наборы инструментальных средств, которые ничего не знают о новом атрибуте (или о новой реализации определил прагму `Attribute_Definition`), просто проигнорируют его, тогда как более новые компиляторы будут рассматривать эту прагму как точно эквивалентный соответствующему пункту определения атрибута. Тот же код приложения может таким образом быть скомпилирован и на цели прежней версии с набором инструментальных средств прежней версии, и на более новой цели (с различным порядком байтов) с недавним компилятором, всегда с помощью того же последовательного представления данных.

Демонстрационная программа

Примечание: две версии демонстрационной программы обеспечиваются ниже. Оба приводят к тому же результату с текущим GNAT Pro выпуска. Однако GNAT GPL 2012 появился раньше, во время разработки этой функции и имеет проблему с синтаксисом, используемым в первой версии демонстрационной программы. Для того релиза компилятора поэтому необходимо использовать `endianness_demo_gpl2012.adb` версию, которая использует альтернативный синтаксис и приводит к ожидаемому результату.

Когда разработка ведется для машины с прямым порядком байтов, результаты выполнения демонстрационной программы следующие:

```
Default bit order: LOW_ORDER_FIRST
N      : 32 102
LE_Bits: 32 102
BE_Bits: 140 65
LE:      32 102
BE:      65 140
```

На машине с обратным порядком байтов программа показывает:

```
Default bit order: HIGH_ORDER_FIRST
N      : 65 140
LE_Bits: 102 32
BE_Bits: 65 140
LE:      32 102
BE:      65 140
```

Связанный со статьёй текст программы

File: endianness_demo.adb

```
 1. pragma Ada_2005;
 2. with System.Storage_Elements; use System.Storage_Elements;
 3. with Ada.Text_IO; use Ada.Text_IO;
 4.
 5. procedure Endianness_Demo is
 6.
 7.    -------------------------
 8.    -- Common declarations --
 9.    -------------------------
10.
11.    subtype Yr_Type is Natural range 0 .. 127;
12.    subtype Mo_Type is Natural range 1 .. 12;
13.    subtype Da_Type is Natural range 1 .. 31;
14.
15.    type Date is record
16.       Years_Since_1980 : Yr_Type;
17.       Month            : Mo_Type;
18.       Day_Of_Month     : Da_Type;
19.    end record;
20.
21.    for Date use record
22.       Years_Since_1980 at 0 range 0  ..  6;
23.       Month            at 0 range 7  .. 10;
24.       Day_Of_Month     at 0 range 11 .. 15;
25.    end record;
26.
27.    ------------------------------------------------------------
28.    -- Derived types with different representation attributes --
29.    ------------------------------------------------------------
30.
31.    --  Bit order only
32.
33.    type Date_LE_Bits is new Date;
34.    for Date_LE_Bits'Bit_Order use System.Low_Order_First;
35.
36.    type Date_BE_Bits is new Date;
37.    for Date_BE_Bits'Bit_Order use System.High_Order_First;
38.
39.    --  Bit order and scalar storage order (note: if the latter is specified,
40.    --  it must be consistent with the former).
41.
42.    type Date_LE is new Date;
```

```
43.    for Date_LE'Bit_Order use System.Low_Order_First;
44.    for Date_LE'Scalar_Storage_Order use System.Low_Order_First;
45.
46.    type Date_BE is new Date;
47.    for Date_BE'Bit_Order use System.High_Order_First;
48.    for Date_BE'Scalar_Storage_Order use System.High_Order_First;
49.
50.    ---------------------------
51.    -- Show bits at address A --
52.    ---------------------------
53.
54.    procedure Show (A : System.Address) is
55.       Arr : Storage_Array (1 .. 2);
56.       for Arr'Address use A;
57.       pragma Import (Ada, Arr);
58.    begin
59.       for J in Arr'Range loop
60.          Put (Arr (J)'Img);
61.       end loop;
62.       New_Line;
63.    end Show;
64.
65.    D_N  : Date    := (32, 12, 12);
66.    --  Native storage (no attribute specified)
67.
68.    D_LE_Bits : Date_LE_Bits := (32, 12, 12);
69.    D_BE_Bits : Date_BE_Bits := (32, 12, 12);
70.
71.    D_LE : Date_LE := (32, 12, 12);
72.    D_BE : Date_BE := (32, 12, 12);
73.
74. begin
75.    Put_Line ("Default bit order: " & System.Default_Bit_Order'Img);
76.
77.    Put ("N      :"); Show (D_N 'Address);
78.
79.    Put ("LE_Bits:"); Show (D_LE_Bits'Address);
80.    Put ("BE_Bits:"); Show (D_BE_Bits'Address);
81.
82.    Put ("LE:     "); Show (D_LE'Address);
83.    Put ("BE:     "); Show (D_BE'Address);
84. end Endianness_Demo;
```

File: endianness_demo_gpl2012.adb

```
 1. pragma Ada_2005;
 2. with System.Storage_Elements; use System.Storage_Elements;
 3. with Ada.Text_IO; use Ada.Text_IO;
 4.
 5. procedure Endianness_Demo_GPL2012 is
 6.
 7.    ------------------------
 8.    -- Common declarations --
 9.    ------------------------
10.
```

```
11.    subtype Yr_Type is Natural range 0 .. 127;
12.    subtype Mo_Type is Natural range 1 .. 12;
13.    subtype Da_Type is Natural range 1 .. 31;
14.
15.    type Date is record
16.       Years_Since_1980 : Yr_Type;
17.       Month            : Mo_Type;
18.       Day_Of_Month     : Da_Type;
19.    end record;
20.
21.    for Date use record
22.       Years_Since_1980 at 0 range 0  ..  6;
23.       Month            at 0 range 7  .. 10;
24.       Day_Of_Month     at 0 range 11 .. 15;
25.    end record;
26.
27.    -------------------------------------------------------------
28.    -- Derived types with different representation attributes --
29.    -------------------------------------------------------------
30.
31.    --  Bit order only
32.
33.    type Date_LE_Bits is new Date;
34.    for Date_LE_Bits'Bit_Order use System.Low_Order_First;
35.
36.    type Date_BE_Bits is new Date;
37.    for Date_BE_Bits'Bit_Order use System.High_Order_First;
38.
39.    --  Bit order and scalar storage order (note: if the latter is specified,
40.    --  it must be consistent with the former).
41.
42.    type Date_LE is new Date;
43.    for Date_LE'Bit_Order use System.Low_Order_First;
44.    for Date_LE'Scalar_Storage_Order use System.Low_Order_First;
45.
46.    type Date_BE is new Date;
47.    for Date_BE'Bit_Order use System.High_Order_First;
48.    for Date_BE'Scalar_Storage_Order use System.High_Order_First;
49.
50.    ----------------------------
51.    -- Show bits at address A --
52.    ----------------------------
53.
54.    procedure Show (A : System.Address) is
55.       Arr : Storage_Array (1 .. 2);
56.       for Arr'Address use A;
57.       pragma Import (Ada, Arr);
58.    begin
59.       for J in Arr'Range loop
60.          Put (Arr (J)'Img);
61.       end loop;
62.       New_Line;
63.    end Show;
64.
65.    D_N  : Date    := (32, 12, 12);
```

```
66.     --  Native storage (no attribute specified)
67.
68.     D_LE_Bits : Date_LE_Bits := (32, 12, 12);
69.     D_BE_Bits : Date_BE_Bits := (32, 12, 12);
70.
71.     D_LE : Date_LE;
72.     D_BE : Date_BE;
73.
74. begin
75.     Put_Line ("Default bit order: " & System.Default_Bit_Order'Img);
76.
77.     Put ("N      :"); Show (D_N 'Address);
78.
79.     Put ("LE_Bits:"); Show (D_LE_Bits'Address);
80.     Put ("BE_Bits:"); Show (D_BE_Bits'Address);
81.
82.     --  Note: in GNAT GPL 2012, intialization of a non-default-
endianness
83.     --  record from an aggregate of whole-record assignment is not
done
84.     --  correctly, so perform per-component assignments here. This
has
85.     --  been fixed in GNAT Pro 7.1 and the fix will also be in GNAT
GPL 2013.
86.
87.     D_LE.Years_Since_1980 := 32;
88.     D_LE.Month            := 12;
89.     D_LE.Day_Of_Month     := 12;
90.
91.     D_BE.Years_Since_1980 := 32;
92.     D_BE.Month            := 12;
93.     D_BE.Day_Of_Month     := 12;
94.
95.     Put ("LE:     "); Show (D_LE'Address);
96.     Put ("BE:     "); Show (D_BE'Address);
97. end Endianness_Demo_GPL2012;
```

Файлы примеров Ada Gems распространяются AdaCore и могут быть использованы или изменены для любых целей без ограничений.

Об авторе

Автор: Thomas Quinot - AdaCore

Thomas Quinot (Фома Аквинский) имеет степень инженера в Télécom Paris и PhD (степень доктора философии) в Université Paris VI (Парижском университете VI). Основным вкладом его исследовательской работы является определение гибкой архитектуры промежуточного программного обеспечения, направленной на взаимодействие между моделями распространения. Он присоединился к AdaCore в качестве Senior Software Engineer (старшего инженера-программиста) в 2003 году и отвечает за технологии распространения. Он также участвует в разработке, обслуживании и поддержке компилятора GNAT.

Last Updated: 10/13/2017
Posted on: 1/28/2013

Обсуждение…

- **Simon Wright**

Jan 28th, 2013

This may depend on the reader’s background - but I’ve never come across a bit/byte diagram for little-endian machines where the least significant byte is on the left! The way the first little-endian diagram has the ‘m’ bits apparently non-contiguous looks weird.

Also, mis-spelling of the first occurrence of pragma Attribute_Definition.

- **Simon Wright**

Jan 29th, 2013

With GNAT GPL 2012 on Mac OS X Mountain Lion, the BE output isn’t as stated.

(В GNAT GPL 2012 на Mac OS X Mountain Lion вывод BE не соответствует заявленному.)

$ ./endianness_demo
Default bit order: LOW_ORDER_FIRST
N : 32 102
LE_Bits: 32 102
BE_Bits: 140 65
LE: 32 102
BE: 32 102

- **Thomas Quinot**

Jan 30th, 2013

Simon,
Thanks for catching the typo—fixed!

Let me clarify the diagram. The two halves represent two succesive positions in memory, numbered in the ``natural’’ way: the lower address left, the higher address right. And since we are on a little-endian machine, of course the lower order byte is stored first, i.e. left. Within each cell of the box, the binary values are shown in the usual notation: most significant bit on the left.

The bits of ‘m’ appear non-contiguous on this diagram, but in reality there is no notion of contiguity of the sequence of bits across different bytes: memory stores a sequence of bytes, not a sequence of bits. It might be clearer to visualize the diagram as:

(Спасибо, что поймали опечатку - исправлено!

Позвольте мне уточнить диаграмму. Две половины представляют две последовательные позиции в памяти, пронумерованные «естественным» способом: нижний адрес слева, верхний адрес справа. И

поскольку мы находимся на машине с прямым порядком байтов, конечно, младший байт сохраняется первым, то есть слева. Внутри каждой ячейки поля двоичные значения показаны в обычной записи: самый значимый бит слева.

Биты «m» кажутся несмежными на этой диаграмме, но в действительности нет понятия непрерывности последовательности битов в разных байтах: в памяти хранится последовательность байтов, а не последовательность битов. Может быть более понятным представить диаграмму в виде:)

```
Address | Fields    | Binary value
--------+-----------+-------------
n       | m yyyyyyy | 0 0100000
n+1     | ddddd mmm | 01100 110
```

- **Thomas Quinot**

Feb 1st, 2013

Simon,

Indeed we have observed the behaviour you describe with GNAT GPL 2012. This release was branched at a point in time where development and field testing of the Scalar_Storage_Order attribute was still in progress. However we opted to include it as it was at the time to give users a chance to experiment with this preliminary version.

This issue of course has been fixed since then in GNAT Pro, and was not present anymore where Scalar_Storage_Order was officially announced on 2012-08-04. An alternate version of the test case, which works with GNAT GPL 2012 as well, is now available above. The next GPL release (GNAT GPL 2013, which should be released in June) will also include the fix as well, and will produce the same result with both test cases.

(Действительно, мы наблюдали поведение, описанное вами в GNAT GPL 2012. Этот выпуск был разветвлен в тот момент, когда разработка и полевое тестирование атрибута Scalar_Storage_Order еще продолжалось. Однако мы решили включить его, как это было в то время, чтобы дать пользователям возможность поэкспериментировать с этой предварительной версией.

Эта проблема, конечно, была исправлена с тех пор в GNAT Pro и больше не присутствовала там, где Scalar_Storage_Order был официально объявлен 2012-08-04. Альтернативная версия тестового примера, которая также работает с GNAT GPL 2012, теперь доступна выше. Следующий выпуск GPL (GNAT GPL 2013, который должен быть выпущен в июне) также будет включать исправление и даст одинаковый результат в обоих тестовых примерах.)

Thomas.

- **Simon Wright**

Feb 1st, 2013

Thomas,

Thanks for the updated demo, which does indeed behave as expected with GNAT GPL 2012.

Just for info, GCC 4.8 experimental r195611 still gives the same wrong answer.

(Спасибо за обновленную демонстрацию, которая действительно ведет себя так, как ожидалось с GNAT GPL 2012.

Просто для информации, GCC 4.8 experimental r195611 все еще дает тот же неправильный ответ.)

- **Thomas Quinot**

  Feb 1st, 2013

The results you are seeing on GCC 4.8 are expected as the required changes in the back-end have not been merged at the FSF yet.

(Результаты, которые вы видите в GCC 4.8, ожидаются, поскольку необходимые изменения в серверной части еще не объединены с FSF.)