Gem #140: Bridging the Endianness Gap

by Thomas Quinot—AdaCore

Let's get started!

This Gem presents a new implementation-defined attribute introduced in the GNAT Ada tool suite that aims to facilitate the peaceful coexistence of big-endian and little-endian data applications. This is particularly important, for example, when retargeting an application from legacy hardware using a given endianness to another platform of different endianness. If any data was stored in memory or persistent storage by the legacy system, or if interoperability with other subsystems is to be preserved, it is necessary for all data structures to have precisely the same representation on the two platforms.

Consider the following two-byte data structure and record representation clause:

```
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;
```

The representation of "December 12th, 2012" on a big-endian system is as follows:

0100000 1	100 01100
yyyyyyy m	mmm ddddd
65	140

On a little-endian system the same date is represented as:

0 0100000	01100 110
m yyyyyyy	ddddd mmm
32	102

One may believe that standard attribute Bit_Order will resolve this situation. Let's give it a try, and amend the declaration above with:

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

On a big-endian system this is the default, and we can verify that the representation is unchanged: the clause just restates the default explicitly (it is said to be *confirming*), and has no further effect.

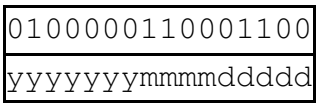
On a little-endian system, however, the result isn't the expected one. To understand why, one needs to look at how the Ada standard defines the interpretation of record representation clauses.

When the bit order is the default bit order, growing bit offsets simply correspond to going into successive (growing) addresses in memory. But when the bit order is the opposite value, bits are numbered "backwards" with respect to the machine's way of storing data as successive bytes, so additional rules are required to know what bits we need to look at.

It is important to keep in mind that bit offsets for a component in a record representation clause are always relative to some integer value (called "machine scalars" in the Ada RM), from which the component value is extracted using a shift and a mask operation.

To find out which machine scalar a given component belongs to, you must first identify the set of components that share the same byte offset. In our example, this would be all three components, since all are specified with a byte offset of 0. Next, consider the highest bit offset for all of these components. Here, it's 15. The size of the machine scalar is then the next larger power of two: 16 in this example. So, this means that all three components in our record are part of a single machine scalar which is a two-byte integer of size 16 bits. If you are on a little-endian machine, the low-order byte of this machine scalar is always the one stored at the lower address, and the high-order byte is the one stored at the higher address. And it is essential to note that this is independent of the bit order specified for the data structure.

So, if you are on a little-endian system, and you specify a High_Order_First bit order, the two-byte machine scalar value will be:



which when stored as two successive bytes in memory will correspond to:

10001100	01000001
140	65

It is interesting to note that this differs from *both* the native little-endian representation *and* the native big-endian representation. That's because the order in which the bytes that constitute machine scalars are written to memory is not changed by the Bit_Order attribute -- only the indices of bits within machine scalars are changed.

Now enter *Scalar_Storage_Order*

It is precisely in order to overcome this limitation of the language that a new attribute Scalar_Storage_Order was introduced in GNAT. The effect of this attribute is precisely to override the order of bytes in machine scalars for a given record type. So let's add another attribute definition:

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

This means that the bytes constituting the machine scalars must be swapped when stored in memory. We see that the memory representation then becomes (65, 140): it is now consistent with the native representation on a big-endian platform.

Existing code for a big-endian system can thus be ported to a little-endian system without any fuss, and without any change of data representation, just by adding appropriate attribute definitions on relevant record type declarations.

Compatibility with legacy toolchains

When retargeting an application with a change of endianness, it is convenient to use attribute Scalar_Storage_Order so that the new platform uses the same data representation as the old one. However, you might still want to be able to compile your application for your old target, with a legacy toolchain that might not support the newer attribute. In this case you can specify it using the alternate syntax:

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

Older toolchains, which know nothing about the new attribute (or about the new implementation defined pragma Attribute_Definition), will simply ignore it, whereas

newer compilers will treat this pragma as exactly equivalent to the corresponding attribute definition clause. The same application code can thus be compiled on both the legacy target with a legacy tool chain, and on the newer target (with different endianness) with a recent compiler, always using the same consistent data representation.

Demo program

Note: two versions of the demo program are provided below. Both produce the same result with current GNAT Pro releases. However, GNAT GPL 2012 was branched early during development of this feature and has an issue with the syntax used in the first vresion of the demo program. For that compiler release, you therefore need to use the endianness_demo_gpl2012.adb version, which uses an alternative syntax and produces the expected result.

When run on a little-endian machine, the attached demo program outputs:

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

On a big-endian machine, it shows:

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

endianness_demo.adb (http://www.adacore.com/uploads_gems/endianness_demo.adb)endianness_demo_gpl2012.adb (http://www.adacore.com/uploads_gems/endianness_demo_gpl2012.adb)

About the Author

Thomas Quinot holds an engineering degree from Télécom Paris and a PhD from Université Paris VI. The main contribution of his research work is the definition of a flexible middleware architecture aiming at interoperability across distribution models. He joined AdaCore as a Senior Software Engineer in 2003, and is responsible for the distribution technologies. He also participates in the development, maintainance and support of the GNAT compiler.

Last Updated: 10/13/2017

Posted on: 1/28/2013

QGen (/qgen)

Services (/services)

Support (/support)

Industries (/industries)

Automotive (/industries/automotive)

Avionics (/industries/avionics)

Rail (/industries/rail)

Air Traffic (/industries/atm)

Space (/industries/space)

Defense (/industries/defense)

Resources (/resources)

Books (/books)

Tech Papers (/tech-papers)

Documentation (/documentation)

Webinars (/webinars)

Devlog (/devlog)

Company (/company)

About AdaCore (/company/about)

Careers (/company/careers)

Contact (/company/contact)

Customer Support (/support)

Login to GNAT Tracker (/login)

Expert Support (/support)

Contact Us (/company/contact)

Pricing (/get-a-quote)

News (/news)

Press Releases (/press)
AdaCore in the Press (/in-the-press)
Events (/events)
Inside AdaCore (/newsletter)

Community (/community)
Download (/download)
Getting Started (/get-started)

Academia (/academia)
Overview (/academia)
Projects (/academia/projects)
Universities (/academia/universities)
GAP Login (/login?mode=gap)

About Ada (/about-ada)
The Ada Language (/about-ada)
Benefits and Features (/about-ada/benefits-and-features)
Ada Comparison Chart (/about-ada/comparison-chart)

Other AdaCore Sites
The AdaCore Blog (http://blog.adacore.com)
SPARK 2014 (http://www.spark-2014.org)
AdaCore U (http://u.adacore.com)
Make with Ada (http://makewithada.org)

(http://twitter.com/AdaCoreCompany)
(https://www.linkedin.com/company/39996)
(https://www.youtube.com/user/AdaCore05)

 (https://github.com/AdaCore)  (/rss)