Lady Ada Mediates Peace Treaty in Endianness War

Ada Europe 2013

Thomas Quinot

AdaCore

Berlin, 2013-06-11

AdaCore
The GNAT Pro Company

Overview

Re-targeting

Platform endianness can change when porting from legacy hardware, e.g. SPARC or PowerPC to Intel.

Need to preserve representations of...

- stored data
- interfaces to other legacy subsystems

... while minimizing code changes.

Interoperability

In the context of a distributed application, partitions must agree on data representation:

- data size
- endianness
- complex (constructed) data types
- padding
- ...

Three different representations involved

- Sender native
- Wire
- Receiver native

Ideally all agree... but how to describe a given fixed layout in a notation interpreted uniformly across all targets?

Overview

Representation clauses

- Refine type declarations
- Specify particular representation aspects
- Allow conformance to externally mandated requirement for representation (message format, shared data...)

Record representation clauses

For each component, define *position* of components:

- byte (storage element) position
- bit range

(but can't specify endianness, i.e. internal structure of the indicated bit slice).

Overview

Ada 83: Where is my bit 0?

A two-byte data structure

```
type R is record
   X : Character;
   Y : Boolean;
   Z : I7;          -- type I7 is range 0 .. 127
end record;
for R use record
   X at 0 range 0 .. 7;
   Y at 1 range 0 .. 0;
   Z at 1 range 1 .. 7;
end record;
```

Interpretation

	Low order first	High order first
X	first byte of representation	
Y	Least sig. bit of 2nd byte	Most sig. bit of 2nd byte
Z	*Shift_Right* (2nd byte, 1)	2nd byte *and* 2#0111_1111#

Work-around:

```
BE_Y at 1 range 7 * Bit_Order'Post (Default_Bit_Order);
```

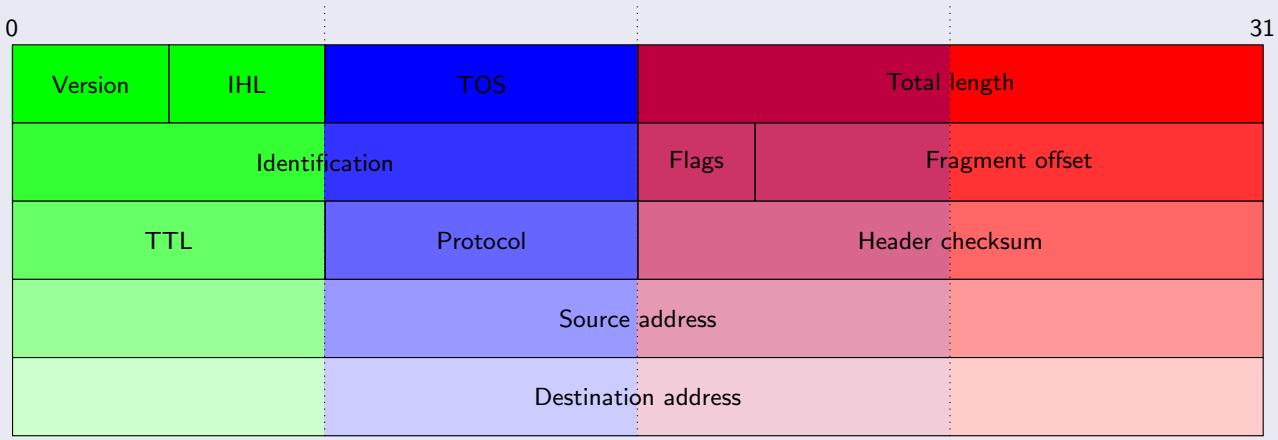
AdaCore
The GNAT Pro Company

Ada 95: Where is my bit 8?

- Ada 95 RM defines interpretation of storage place in terms of the *sequence of bits* that represent a component
- Fine with native bit order: bit 8 is bit 0 of the next byte
- With reverse bit order, no notion of a sequence of bits (bits are not contiguous as numbered anymore when multi-byte values are interpreted as integers)
- Ada 95 AI-133 (binding interpretation)

Ada 2005: Putting the pieces together

IPv4 datagram header



- "Big-endian" protocol: layout is specified in terms of big-endian 32-bit quantities
- Lends naturally to Ada rep clause on big-endian platforms

Rep clause and interpretation

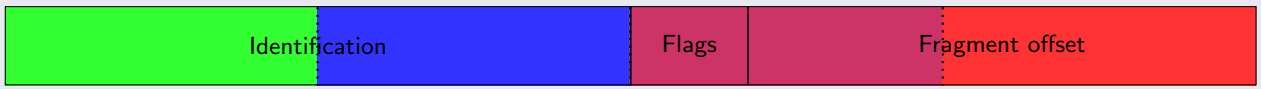
Natural Ada rep clause on BE platforms

```
for IP_Header use record
  Version    at 0 range  0 ..  3;
  IHL        at 0 range  4 ..  7;
  TOS        at 0 range  8 .. 15;
  Length     at 0 range 16 .. 31;

  Ident      at 4 range  0 .. 15;
  Flags      at 4 range 16 .. 18;
  Frag_Offs  at 4 range 19 .. 31;

  --  ...

end record;
```

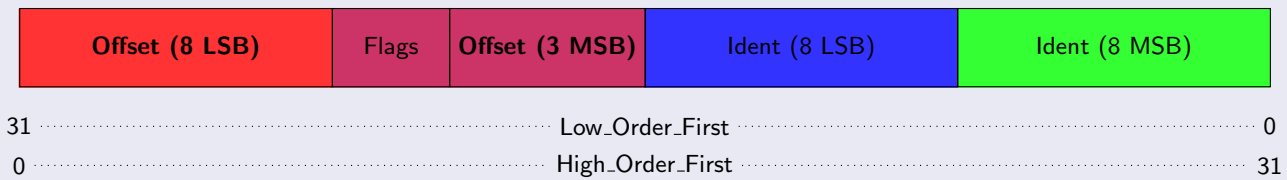
- Components are grouped according to storage position (byte offset)
- All components at a same offset are part of a single *machine scalar*
- Fields are extracted by shift+masking operations on enclosing *machine scalars*

IPv4 header on a little-endian platform

Transmitted data



Corresponding machine scalar on a *little endian* machine



All fields have different bit ranges (unless overriding default *bit order*)

Ident requires byte swapping

Offset crosses a byte boundary ⇒ becomes non-contiguous,
cannot be specified by any Ada rep clause
regardless of bit order overriding

Endianness: issues summary

Ada 83

Storage place attributes are endianness-dependent.

Ada 95

- Bit_Order attribute identifies where bit 0 is.
- Meaning of component storage place that crosses byte boundary with reverse storage order?

Ada 2005

- Storage places are relative to the underlying machine scalar
- Bit-order attribute changes bit numbering, *not* byte ordering
- Byte swapping required in the presence of multi-byte components

Overview

Explicit byte-swapping

Per-component

- C-like hton* / ntoh* functions
- GNAT provides efficient implementations in GNAT.Byte_Swapping (based on GCC built-ins)

Wholesale

- Revert byte order at the level of a complete composite structure
- Must generate a complete new record representation clause by hand (cumbersome/error prone)

Arithmetic notation

```
BE_Long := Bytes (0) * 16_777_216
         + Bytes (1) *     65_536
         + Bytes (2) *        256
         + Bytes (3);
```

Code generation from data model

Tranxgen produces SPARK accessors from XML specification of a data structure:

Example Tranxgen data model

```
<package name="Date_And_Time_Pkg">
 <message name="Date_And_Time">
  <field name="Years_Since_1980" length="7" />
  <field name="Month"            length="4" />
  <field name="Day_Of_Month"     length="5" />
  <field name="Hour"             length="5" />
  <field name="Minute"           length="6" />
  <field name="Two_Seconds"      length="5" />
 </message>
</package>
```

- Spec is implicitly for a big-endian structure (tool was developed as part of producing a certifiable TCP/IP stacl)
- Generated code uses arithmetic notation for endianness independence

Overview

The *Scalar_Storage_Order* attribute

- Implementation-defined representation attribute (GNAT)
- When accessing a scalar component of a record,
  byte swap the underlying *machine scalar*
- Byte swapping is transparently generated by the compiler

Case of non-scalar components

- Effect of attribute is defined for scalar components only
- Ensures no double swapping occurs
- Nested structures should have their own Scalar_Storage_Order attribute
- Consistency required except if inner structure falls on byte boundary

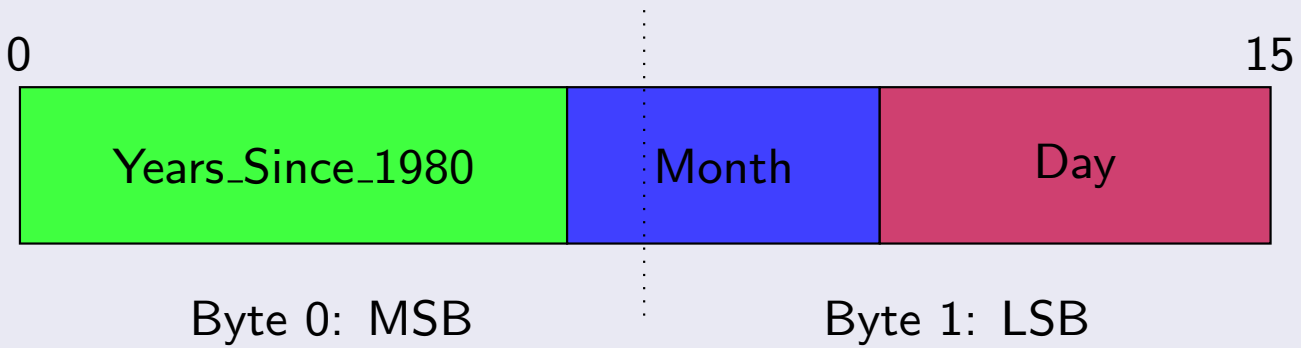
Data structure declaration

Structure similar to MS-DOS FAT timestamps

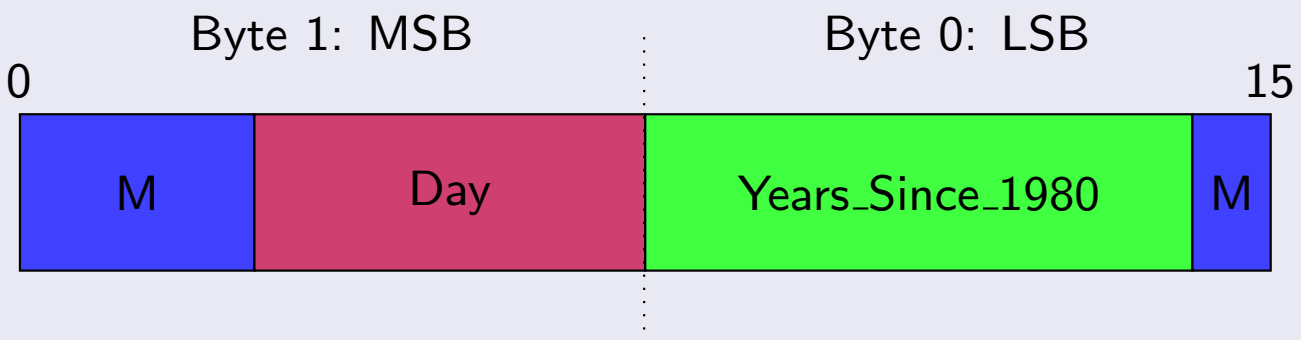
```
type Date_And_Time is record
   Years_Since_1980 : Yr_Type;
   --  7 bits

   Month            : Mo_Type;
   --  4 bits

   Day_Of_Month     : Da_Type;
   --  5 bits

   Hour             : Ho_Type;
   Minute           : Mi_Type;
   Two_Second       : S2_Type;
end record;

for Date_And_Time use record
   Years_Since_1980 at 0 range 0  .. 6;
   Month            at 0 range 7  .. 10;
   Day_Of_Month     at 0 range 11 .. 15;
   Hour             at 2 range 0  .. 4;
   Minute           at 2 range 5  .. 10;
   Two_Second       at 2 range 11 .. 15;
end record;
```

Mapping to byte sequences

Representation on big endian machine

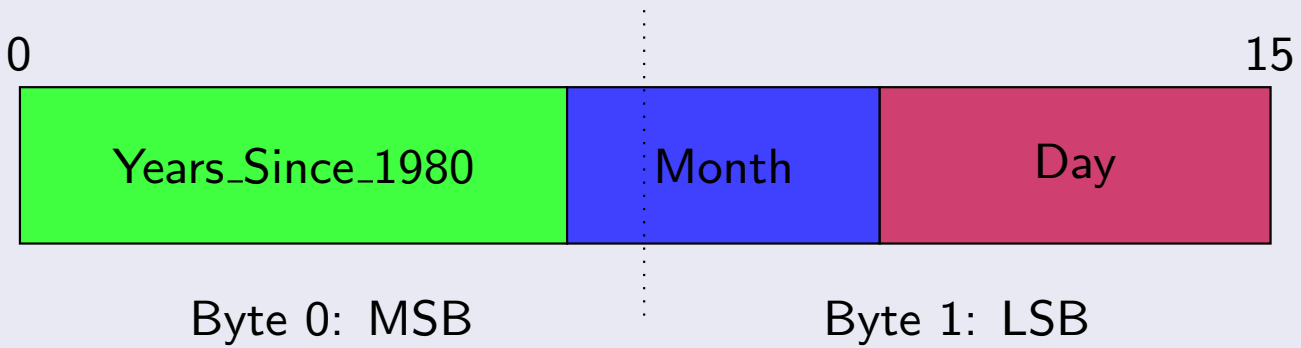


Same 16-bit region interpreted on little-endian machine

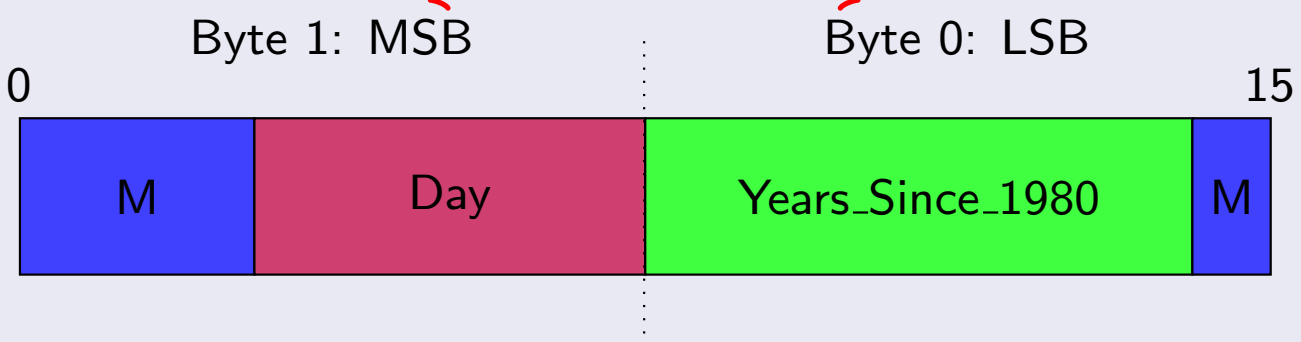


Mapping to byte sequences

Representation on big endian machine



Same 16-bit region interpreted on little-endian machine



Making it endianness neutral

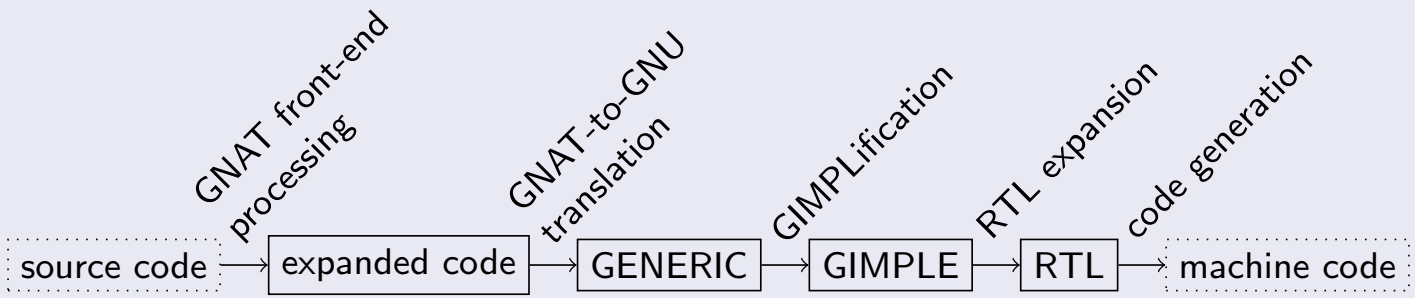
Revisited type declaration

```
type Date_And_Time is record
   ...
end record;

for Date_And_Time use record
   ...
end record;
for Date_And_Time'Bit_Order
  use System.High_Order_First;
for Date_And_Time'Scalar_Storage_Order
  use System.High_Order_First;
```

- The only needed change is the addition of an attribute definition clause
- Underlying machine scalar is byte swapped upon component load/store
- From a usage standpoint, component values seen as though in native endianness

Implementation

GNAT processing passes



- Scalar_Storage_Order is implemented in the RTL expansion pass: all powerful high-level optimizations on GIMPLE still apply
- SSO is a property of scalars that are *stored as part of a composite object*: scalar *values* and registers are always in native endianness;
- Byte swapping is part of *composite load/store*

Performance discussion

- Zero cost if specified endianness is native (no pessimization, $\neq$ endianness-independent arithmetic conversion)
- Unavoidable distributed cost when conversions are required
- Mitigation is possible by converting records as a whole to ancestor type with native endianness.

Derived type example

```
type Date_And_Time is record
   ...
end record;
--  Native, efficient representation

type External_Date_And_Time is new Date_And_Time;
for External_Date_And_Time use record
   ...
end record;
for External_Date_And_Time'Scalar_Storage_Order
  use External_Bit_Order;
for External_Date_And_Time'Bit_Storage_Order
  use External_Bit_Order;
```

Overview

Summary and perspectives

New representation attribute *Scalar_Storage_Order*

- Solves longstanding issue of expressing record layout consistently across target endianness boundaries
- Minimal code change required
- No error-prone or inefficient manual byte swapping
- Zero cost when confirming native byte order

Next steps:

- Modify Tranxgen to take advantage of new attribute
- Submit definition to ARG for standardization in Ada 202Z

Questions?

?