Subject: Re: Bug? yea or nay.
Posted by Dave Greenwood on Fri, 16 Feb 2001 16:25:16 GMT
View Forum Message <> Reply to Message

```
William Daffer <whdaffer@earthlink.net> wrote:
>
> by the way, I'm 'vapuser'
>
>
> Paul van Delst <pvandelst@ncep.noaa.gov> writes:
>
>> Vapuser wrote:
>>>
>>> I greatly simplified the structure in question to elucidate the point.
>>>
>>> IDL> print,!version
>>> { mipseb IRIX unix 5.3 Nov 11 1999}
>>>
>>> IDL> r={a:0.d, c:0L} & print,n_tags(r,/length),totsize(r)
>>> 16        12
>>>
>>
>> <snip>
>>
>>> What say you? Bug or not?
>>
>> From the IDL online manual (I know you already know this, Vapuser. It's here for
>> completeness):
>>
>> NTAGS
>>   KEYWORDS
>>     LENGTH
>>
>>     Set this keyword to return the length of the structure, in bytes.
>>
>>     Note - The length of a structure is machine dependent. The length
>>     of a given structure will vary depending upon the host machine.
>>     IDL pads and aligns structures in a manner consistent with the host
>>     machine's C compiler.
>>
>
>   I understand this, although, as I pointed out, I thought that if you
>   always went from largest to smallest you wouldn't get into trouble
>   since each indivudual quantity couldn't help but be on a border that
>   was a multiple of its type.
```

Data alignment is a performance issue, of course, but it's not quite as
simple as just having each quantity aligned on its "natural" boundary.

Hardware, especially modern RISC hardware, tends to be optimized for certain sized operations, for example 32 bits. That is, memory ops will be done as 32-bit load/stores, arithmetic ops as 32-bit adds/subtracts, etc. If you want to add 1 to a 16-bit number, that machine is going to actually read 32 bits out of memory, save the "other" 16 bits somewhere, possibly shift the desired 16 bits into the low order half of the register, possibly mask out the other 16 bits, add 1, possibly shift the result into the other half of the register, replace the "other" 16 bits and store the 32-bit result back in memory. On such hardware it's much more efficient to ensure that frequently used 16-bit quantities are stored as 32-bit values and just "waste" the other 16 bits. Many, if not most, modern C compilers do that for you (or *to* you, depending on your point of view).

Another way of saying the above is that it's more efficient to align data on the *hardware's* natural boundary than on the *data's* natural boundary. Compilers know about that and align data accordingly.

Dave

Dave Greenwood   Email: Greenwoodde@ORNL.GOV
Oak Ridge National Lab   %STD-W-DISCLAIMER, I only speak for myself