Clément Béra ~ Smalltalk, Tips 'n Tricks

64 bits Immediate Floats

09 - *Friday* - *Nov 2018*

Posted by Clement Bera in Cog, Spur

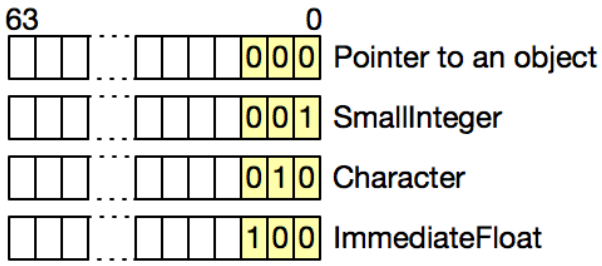
≈ Leave a comment

Hi all,

In this post I will try to discuss some inner details of OpenSmalltalk-VM immediate floats. Immediate floats are present only in 64 bits hence I won't talk about 32 bits VM in the whole blog post. In addition, OpenSmalltalk-VM supports only double precision IEEE floating pointer, hence I won't discuss single precision IEEE floating pointer.

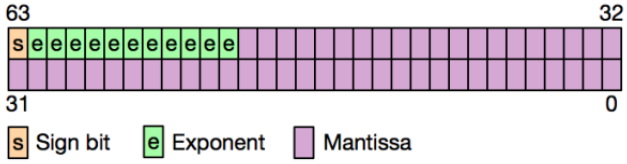
Immediate objects

OpenSmalltalk-VM uses an immediate object scheme to represent object oriented pointers (oop) in memory. Basically, due to 64 bits alignment, the last 3 bits of all pointers to objects are 000. This is abused to encode in the oop itself specific objects, in our context, SmallIntegers, Characters and ImmediateFloats. This optimization allows to save memory and to improve performance by avoiding boxing allocation for common arithmetic operations. The last 3 bits of an oop are called a tag. The Immediate Float tag is 100 (4 in decimal). Objects encoded directly in the oop are in our terminology called immediate objects.



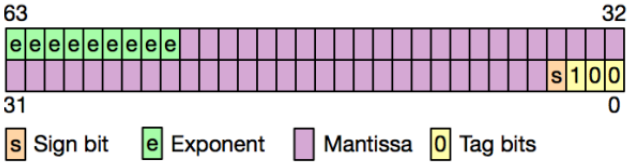
Immediate floats

OpenSmalltalk-VM and its clients use the double precision IEEE format to represent floating pointers, supported by most modern hardware.



The key idea to the immediate float design is to use an immediate representation of double precision floats to avoid boxing and save memory, while still being 100% compatible with the IEEE double precision format (Customers requirement).

Therefore, in 64 bits, OpenSmalltalk-VM use two implementations for floats. The most common floats are represented with immediate floats, where 3 bits of the exponents are abused to encode the tags. The rest of the floats are represented as boxed floats.



By design, immediate floats occupy just less than the middle 1/8th of the double range. They overlap the normal single-precision floats which also have 8 bit exponents, but exclude the single-precision denormals (exponent-127) and the single-precision NaNs (exponent +127). +/- zero is just a pair of values with both exponent and mantissa 0.

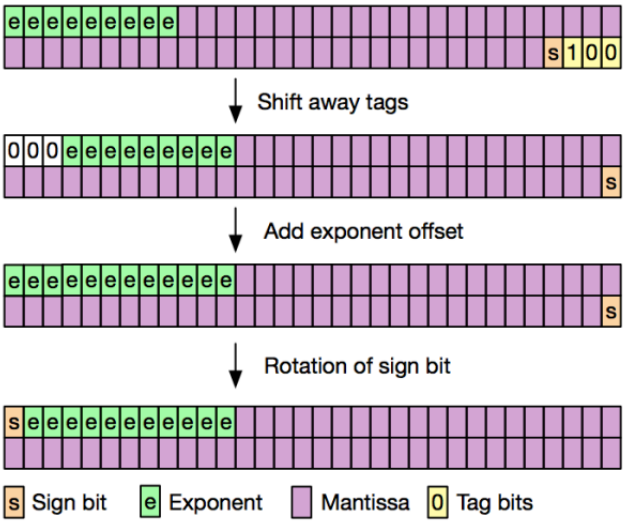
So the non-zero immediate doubles range from
+/- 0x3800,0000,0000,0001 / 5.8774717541114d-39
to +/- 0x47ff,ffff,ffff,ffff / 6.8056473384188d+38

Encoding and decoding

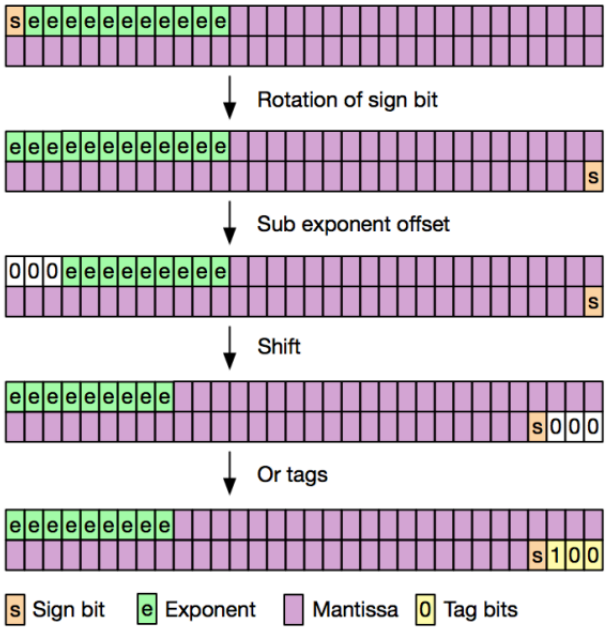
The encoded tagged form has the sign bit moved to the least significant bit, which allows for faster encode/decode because offsetting the exponent can't overflow into the sign bit and because testing for +/- 0 is an unsigned compare for <= 0xf.

So given the tag is 4, the tagged non-zero bit patterns are
0x0000,0000,0000,001[c(8+4)]
to 0xffff,ffff,ffff,fff[c(8+4)]
and +/- 0d is 0x0000,0000,0000,000[c(8+4)]

Decoding of non-zero values in machine code is performed as follow:



Encoding of non-zero values in machine code is performed as follow:



Reading floats in general is fairly easy, the VM checks the class index, if the class index of immediate float is present, then the float is decoded from the oop, if the boxed float class index is present, the float is read from the boxed object.

Each primitive operation (arithmetic, comparison, etc.) has now to be implemented twice, once in both classes, where the first operand is expected to be an instance of the class where it is installed. In Smalltalk float primitive operations succeed if the second operand is one of the 2 float classes or a SmallInteger. It fails for large integers and arbitrary objects, in which case the VM takes a slow path to perform correctly the operation.

At the end of arithmetic operations, the resulting float has to be converted back from the unboxed format to either an immediate float or a boxed float. To do so, the VM checks the exponent of the float against the smallFloatExponentOffset, 896. 896 is 1023 – 127, where 1023 is the mid-point of

the 11-bit double precision exponent range, and 127 is the mid-point of the 8-bit SmallDouble exponent range. If the exponent is in range, it can be converted to an immediate float. If not, one needs to check if the float is +/- 0, in which case it can still be converted to an immediate float, else it has to be converted to a boxed float. The code looks like that in Slang:

```
^exponent > self smallFloatExponentOffset
ifTrue: [exponent <= (255 + self smallFloatExponentOffset)]
ifFalse:
[(rawFloat bitAnd: (1 << self smallFloatMantissaBits - 1)) = 0
ifTrue: [exponent = 0]
ifFalse: [exponent = self smallFloatExponentOffset]]
```

x86_64 encoding/decoding

To conclude the post, here are the instructions generated in x86_64 to encode immediate floats. I put the instruction so that you can see how to encode efficiently using the theoretical design from the figures above and in addition quick checks for +/- 0.

```
000020da: rolq $1, %r9 : 49 D1 C1
000020dd: cmpq $0x1, %r9 : 49 83 F9 01
000020e1: jbe .+0xD (0x20f0=+@F0) : 76 0D
000020e3: movq $0x7000000000000000, %r8 : 4D B8 00 00 00 00 00 00 00 70
000020ed: subq %r8, %r9 : 4D 2B C8
000020f0: shlq $0x03, %r9 : 49 C1 E1 03
000020f4: addq $0x4, %r9 : 49 83 C1 04
```

Decoding is easier:

```
00002047: movq %rdx, %rax : 48 89 D0
0000204a: shrq $0x03, %rax : 48 C1 E8 03
0000204e: cmpq $0x1, %rax : 48 83 F8 01
00002052: jle .+0xD (0x2061=+@61) : 7E 0D
00002054: movq $0x7000000000000000, %r8 : 4D B8 00 00 00 00 00 00 00 70
0000205e: addq %r8, %rax : 49 03 C0
00002061: rorq $1, %rax : 48 D1 C8
00002064: movq %rax, %xmm0 : 66 48 0F 6E C0
```

Let me know if you have any question or you want me to expand this post with something else.

Note: Part of the blog post was extracted from the SpurMemoryManager class comment on immediate float, I thank Eliot Miranda and other OpenSmalltalk-VM contributors for writing it.