

Branch displacement Optimization

Source: <http://coding.derkeiler.com/Archive/Assembler/alt.lang.asm/2006-11/msg00216.html>

- *From:* "randyhyde@xxxxxxxxxxxxxx" <randyhyde@xxxxxxxxxxxxxx>
 - *Date:* 2 Nov 2006 16:00:04 -0800
-

Recently, Rene "Betov" Tournois has seen the light and thought that "small-first, grow large" is a good idea. Somehow, he thinks that this is a new and nifty idea. Of course, the explanation for it has been in this very newsgroup since Jan, 2004. See the following link and follow-ups:

http://groups.google.com/group/alt.lang.asm/browse_frm/thread/528907cad91150c4/a49af5f25dd15296?lnk=gst&q=b

However, because this seems to be of current interest, I'll repost that essay here:

=====

This is a short essay that attempts to explain the basics of "displacement optimization" in machine code such as on the x86. In posts I've made here and on various newsgroups (comp.lang.asm.x86 and alt.lang.asm) it is quite clear that many people don't understand what this is all about, and even those who do understand the basic issues can't believe the problem is as difficult to solve (optimally) as I claim that it is. In this essay I'm hoping to explain the process and the problems in such a way so to make all this clear.

What is This All About?

The first place to start is with the question "what is branch displacement optimization?" On certain CPUs (e.g., the x86), certain instructions have a limited branch range because of the way the CPU encodes the target address of the branch. For example, a typical "JE" instruction on the x86 is two bytes long – one byte for the opcode and one byte for a relative displacement. This relative displacement allows the instruction to transfer control to some instruction within +/- 128 bytes (roughly) of the JE instruction.

Branch displacement Optimization

What happens if the target address is **not** within range? Well, if the target address is within +/- 4Gbytes, the 386 and later devices have a special six-byte version of JE (two-byte opcode and two-byte displacement) that can transfer control to the label. If you're operating on a chip earlier than the 386, you have to emit a five-byte sequence like the following:

```
jne skipJmp ;two bytes  
jmp target ;three bytes  
skipJmp:
```

In early x86 assemblers (and in a few assemblers that have been created lately), it was the programmer's r