This results in 384 possible combinations for this subset of the addressing modes alone. Evidently some more factorizing is required on these address calculations. This type of memory access is therefore broken down further, until we get:

```
load eax
memoryreset
load segment es
inc address ecx*4
inc address ebx
inc address imm 8
load [segment:address]
add
store eax
updateflags
```

To produce an emulated instruction set that performs acceptably, we have had to balance two sets of priorities:

- First, we must balance decode time against execution time in order to optimize overall execution speed. We must remember that in the interpreted processor, our main target is low-latency initial execution. Commonly executed code should get handed on to later optimization stages. Our initial aim here is to get the code out the door without blocking the whole emulation. Later optimizations can be done asynchronously, and time spent here holds up everything. So we are looking for a relatively simple set that can be decoded quickly.
- Second, we have to balance instruction set size against "compiled" code length. A small set will naturally produce verbose code, and a larger set should be more compact. We say "should" because a large set can still require longer sections if it is badly chosen. An interpreter for a smaller set will be smaller in code and footprint, and therefore will execute each of its operations much faster, but conversely it has a larger set to execute. So we are looking for a reasonable balance of set size against code length in order to get a near-optimal performance out of the interpreter.

In finding the optimal point for both of these trade-offs, it is important to keep Hoare's Dictum[‖] in the back of your mind:

Premature optimization is the root of all evil.

—C. A. R. Hoare

The precise sweet spot for lots of these optimizations will be system-dependent. In a Java environment, the system includes not only the physical hardware but also the JVM. With the added factor that the Java component of the environment is invariably just-in-time compiled, small scoped performance benchmarks are notoriously unreliable. To this end we refrained from overusing such benchmarks to guide our coding choices, and instead relied on first

‖ *http://en.wikiquote.org/wiki/C._A._R._Hoare*