USE OF NOPCODES AS EXECUTABLE LABELS

Received: 78 April 7

Every microprocessor has one opcode defined as a "no-operation" (NOP), useful as a filler in program gaps or timing loops. In addition, some micros have opcodes (defined or undefined, the latter "illegal" and so not necessarily identical in chips from different mask sets) that are executed without causing any detectable change anywhere. These can be called *nopcodes*. E.g., the 8080 and Z80 have 7 instructions ($7F, $40, $49, $52, $5B, $64, and $6D) that move the content of a register into the *same* register, and so are *defined nopcodes*. Although they must now be the least-often used opcodes in the 8080 set, they *could* be used as *executable labels for the next instruction*. Such labels, although meaningless to the CPU and with no effect on operation (except timing), could be *assigned meanings* interpretable by debuggers, relocaters and other programs. The information content of the 7 "monops" (short for one-byte nopcodes) is limited, but could be increased by using sequences of 2 ("monop-duos") or 3 ("monop-trios"). For example, the 8080 could have 49 monop-duos and 343 trios. Since interpreting logic could easily recognize solos, duos, and trios, this would allow 399 different labels. If the 8080 NOP (00) were allowed as the second and/or third element, the total would become 511.

Although the 650X has no defined nopcodes, it now has a large supply of undefined ones. (This of course will change in the enhanced versions now in the design stage, but there are thousands of systems now using the original design.) It has 6 monops ($1A, $3A, $5A, $7A, $DA, and $FA). It also has 5 *two-byte* illegals ($80, $82, $89, $C2, and $E2) that require an "operand" byte but make no use of it. I think of these as "binopcodes" (nopcode plus one "noperand") or "binops". Since the noperand can be any one of the 256 possible 8-bit bytes, the information content of binops is very large (and they would be simpler to interpret than monop-duos or -trios). There are also "trinops" (e.g., the 8 illegals of type X_1C).

Before going on, I should reply to the letter by Paul Schick (*DDJ* #21), that is a strongly negative reaction to my article on 650X opcodes (*DDJ* #17). Neither in that article nor in my note in *BYTE* (*2* (12): 72, 1977) did I advocate the use of illegal opcodes *as instructions*. In fact, the whole point of my legality-testing subroutines OPLEGL (*DDJ* #17 and #19) and HYLEGL (*DDJ* #22) was to *screen out* illegals. I presume that the 650X designers know very well what their chip does with illegal opcodes (although they see no need to document it!). I respect their decision *not* to support even the illegal operations that might be somewhat useful. However, designers are not endowed with divine omniscience and infallibility. They can—knowingly—do something foolish, or—unkowingly—do something wise. The point I am trying to make in *this* note is that *some* nopcodes may be so useful as programming aids that their retention as defined elements in an instruction set merits serious consideration by designers. I am presently exploring the implementation of 650X binop-labels, but this will require time and thought, so I am publishing the *concept* in order to make it immediately available to other programmers (and to designers).

In thinking over possibilities for scanning-debugging (of which my program SIMBUG in *DDJ* #19 is a very primitive example), it became increasingly clear that the superiority of assembly langugae over machine language largely rests on its much higher information content, especially in labels. This makes assembly-language programs much longer, somewhat more intelligible to human beings, and quite unintelligible to the microprocessor. A fairly complex assembler program is needed to translate them into machine-intelligible form. The assembler also acts as a scanning-debugger and as a program locating and addressing controller. Labels play the major role in these useful activities with the crucial symbol-table at the core of everything.

In the KIM-1 program SWEETS by Dan Fylstra (*BYTE 3* (2): 62, 1978), there is an ingenious use of machine-language labels. However, a Fylstra-labeled program is not executable. Labels must be removed before execution, and if there are execution bugs they may need to be re-inserted for the program revision. This limitation could be removed by using binop-labels. In fact, some problems of machine-code relocation might be soluble by retaining at least some nop-labels in the taped or disked programs, which could then be address-corrected by a relatively simple program.

A special kind of labeling is also possible with *legal* opcodes. E.g., a STORE instruction addressing a location in ROM is in effect a NOP, so that both the ROM page and the location on that page can convey information. Also, *any* instruction that is *unreachable* in program operation could be used to store non-program information (a trick used in my subroutine BYTNUM, in *DDJ* #17, 1977. However, the detection of and information-retrieval from such labels would be less convenient than with monop- or binop-labels. A ROM-trinop ought to use a STORE opcode normally rarely used in working programs, to make detection easier (the *page* of every such opcode would have to be checked to distinguish between ROM and RAM), but would still involve more bytes and execution time.

The conservative view of a microcomputer system is that it should differ from large systems primarily in scale, in the way that a VW differs from a Mercedes, and that there ought not to be *qualitative* differences in hardware, software, or methods of operation. Innovations should arise in the giant systems, and then (to the extent that the micropocketbooks of micro users allow) be copied at the lower levels. I prefer another analogy. There are no elephants that burrow underground like moles or can fly like bats, and downsizing to the level of insects allows even greater diversification. Micro systems are going to be individualized and customized in ways that would horrify IBM, and some innovations at the micro level (most will of course be failures) may well migrate *upward*.

H.T. Gordon
College of Natural Resources
University of California
Berkeley, CA 94720