Hal T. Gordon
University of CA, Berkeley
Berkeley CA 94720

Instruction Sets Examined and Compared

The second part of this article starts with a look at on-chip and off-chip registers.

The "thinking power" of most CPUs is primarily in the on-chip registers, to which the logic elements have fast direct access, while memory locations (more slowly accessible via the address and data buses) serve primarily for storage. A large fraction of the instruction set (54 percent in the 8080) involves on-chip register manipulations. In one-chip LSI designs, there is a limit to the size and number of on-chip registers. The 2650 has 87 bits in its registers, the 8080 has 96, and the Z-80 "more-is-better" philosophy increased this to 208 bits.

The 6800 took the bold step of reducing the on-chip registers to 72 bits (and the 6502 carried the stripping-down to a record low of 56 bits!). It would be fascinating to know the arguments used by the 6800 designers to persuade Motorola executives to go along with this revolutionary departure from conventional design. One of them must have been that the loss of on-chip power could be compensated by adding (to the simple memory-increment and -decrement types present in the 8080 set) nine new instruction types that *directly* modify or test memory locations (CLR, COM, NEG, TST, ASR, ROR, ROL, ASL and LSR), using them as "off-chip registers." The 6502 designers backtracked, omitting the first five of the 6800 direct-memory instructions (presumably because they did not think they were valuable enough).

The big problem with doing "thinking" in external memory is that, although you now have a plethora of registers to work with, access to them is slow. If I had to guess, I would say that this is what led to the creation of *zero-page instructions* (where the op code *implies* page zero, so that only *one* address byte is needed). The 256 bytes of zero-page function as a bank of external registers. The concept is extended to its utmost by the 6502, in which 33 percent of the op codes involve zero-page.

The Z-80 designers, intent on maximizing the power of their set in every possible way, also saw the advantages of direct-memory ("off-chip register") operations. As with the 6502, they decided that not all the 6800 types were worth including. They did add all the 6800 shift and rotate instructions, plus the branch-carry rotates of the 8080 and the new RRD and RLD types, plus single-bit testing, setting and resetting instructions. Since the BCDEHL registers often serve as on-chip memory, most of these instructions were also implemented for them.

The overall gain in both on-chip and off-chip power is spectacular, although the speed (because of multibyte op codes) is not. The zero-page concept (which needs one-byte op codes) cannot be implemented, but with so many on-chip registers it is not needed. The Z-80 does its "fast thinking" on-chip.

Most CPUs have one *primary accumulator*, endowed with an exceptionally rich repertoire of operations, and a number of *secondary* ones with fewer and often different functions (sometimes, as in the stack pointer, quite specialized). The 6800 is unique in having two nearly equal primary accumulators. This means that the many op codes (56) that confer power on one are duplicated for the other.

There are also five op codes involving interaction between A and B, and three that are restricted to A, so that 61 percent of the whole set involves the accumulators. Although this uses up much of the supply of one-byte codes, it allows a kind of approximation to 16-bit operation. The existence of a 16-bit index register and 16-bit stack pointer also suggests that this idea was in the minds of the designers.

This concept was totally abandoned by the 6502, which went to the extreme of Spartan 8-bit simplicity (even its stack pointer was cut to eight bits by having stack instructions imply page one) and did everything possible to enhance the power of its one accumulator. Although its two auxiliary registers (X and Y) can do data transfers, they have almost no "thinking power" and serve largely as index registers for the many addressing modes of the accumulator. Half of the 6502 op codes involve the accumulator, mostly interactions with external memory (especially the zero-page "off-chip registers").

The 6502 is by far the most "extroverted" of all designs, since only 17 percent of its instructions command purely on-chip operations. You can recognize a 6502 machine-code listing by the high frequency of 2-byte instructions. The same operation coded for the 8080 will use mostly 1-byte instructions.

Although the Z-80 has a duplicate set of the 8080 registers on-chip, this is quite different from the 6800 concept. Only one set is "on-line" at a given moment. The last two precious new Z-80 one-byte op codes are used to *exchange* sets, one for the AF register-pair, the other for the BCDEHL registers. Although the exchange of registers is not novel (the 8080 has three instructions of this type), the new Z-80 exchanges are an advanced feature. In effect, you can quickly insert an alternate processor into the system in a state of readiness for a different or more complicated task.

The Signetics 2650 — less well-known but no less remarkable than the other chips — is so different in its organization that it is almost in another dimension. I would guess that the basic decision was to have seven very versatile accumulators. As noted above for the 6800, conferring power heavily drains the supply of one-byte op codes. This was partly solved by organizing the six secondary accumulators into two banks of three, selected (not, as in the Z-80, exchanged) by setting or resetting one bit (RS) in a status register.

Once the concept of using status bits as auxiliary instruction bits (creating a nine-bit op code) had "broken the ice," it was probably easy to use two more status bits (WC and COM) to modify the operation of many instructions and to take the even more radical step of using the three high-order bits of the memory address to specify the addressing modes. Although the remaining 13 bits address only 8K, the address space is raised to 32K by having four selectable 8K banks. This is an anticipation of the "memory-bank-shifting" trick now coming into use to expand memory beyond the 65K limit in other systems.

Having burned so many bridges, the 2650 designers went on to an *on-chip stack* (eight 15-bit registers "pointed to" by three status-register bits) that automatically stores return addresses for subroutines and interrupts. Although Adam Osborne refers to this as "primitive," it allows extremely fast operation (a 2650 subroutine call takes only three cycles, compared to six for the 6502, nine for the 6800 and 17 for the 8080 and Z-80).

There is enough register power so that *nothing at all* is done exclusively on memory locations. Zero-page is not used as off-chip registers, but reserved for fast access to interrupt-servicing subroutines, or for short programs accessed by either branch- or jump-to-subroutine op codes that *imply* zero-page (only one address byte needed). The wealth of original, sophisticated ideas in the 2650 will profoundly influence future designs.

"The wealth of original, sophisticated ideas in the 2650 will profoundly influence future designs."

Addressing Modes

Although everyone agrees that well-designed addressing modes make programming more efficient, it seems to be hard to explain exactly *why* (and, with some of the trickier modes, even *how!*). Program memory locations are addressed consecutively by moving the program counter into the address register.

If the program byte is an op code, it is moved into the control register, and the program counter auto-increments to pick up the next program byte. Some op codes cause the next one or two program bytes to be "interpreted as data," i.e., to be moved into some *other* on-chip register(s) where (depending on what logic networks were set by the op code) they may be just stored or used in an operation. This is *immediate* addressing.

If the op code commands loading of the next two data bytes into the address register, this will provide *direct* addressing of a unique (usually non-program) memory location for the operation of that op code. All designs have this mode. It is good for occasional communication with single locations anywhere in memory, but having to specify an absolute address every time a memory operation is needed would be extremely inefficient.

In the 8080, the major addressing mode is one in which the op code *implies* that the correct memory address is stored in the HL register-pair (this may have been done by a previous load-immediate into HL), causing the content of HL to be moved into the address register. A sequence of 1-byte op codes can now access the memory location specified in HL by implying that the memory address is in HL. While the 8080 can also store 16-bit addresses in its BC and DE registers, these allow only *moves* between the primary accumulator and memory. Arithmetic and logical operations between the accumulator and memory locations are possible only with HL addressing (that's the reason for the exchange instructions between other registers and HL). The awkwardness of interaction with memory is one of the major weaknesses of the 8080, and the reason why other designs (always excepting the Z-80) have not adopted the implied mode.

In the 6800 there is a 16-bit "index register" (X) that resembles HL in that it can be loaded-immediate and incremented or decremented. However, op codes that imply X-addressing must be followed by a "displacement" byte, which is added to the "base-address" in X to generate the true address transferred to the address register. This "indexed" access to a memory location requires two program bytes, instead of only one in the 8080 mode. The gain is that any location in the address range from X to X+255 can be accessed. The difference is analogous to that between the limited moves of a pawn in chess and the freer moves of a rook.

If the displacement byte were always zero, the 6800 X would work exactly like the 8080 HL. The 6800 X also shows a close analogy with its zero-page addressing; if X were set to 0000, X-addressing would be indistinguishable from zero-page addressing. In actual use, X-indexing allows *any* 256 consecutive locations in memory to be accessed by what is, in effect, a 1-byte direct address, with the added feature that the base address can be incremented or decremented by 1-byte instructions. It is a powerful mode, and the Z-80 designers liked it so well that they added *two* 16-bit index registers (IX and IY) to their chip. Unfortunately, their use (including increment and decrement) requires 2-byte op codes, and even 3-byte ones with the bit-manipulation codes, so operation is less code-efficient and slower than the 6800 X-addressing. Even so, it is a major enhancement relative to the 8080.

Zero-page addressing exists in the 6800 and 6502 (and in the 2650, in a different form). Since a "zero-page" op code *implies* that the high-order (page) address must be set to 00, only the low-order byte need be specified in the instruction; this is a combination of implied and direct addressing. Stack addressing in all designs implies that the address exists in the stack pointer, a specialized but efficient use of the implied mode.

Both the 6502 and the 2650 use only eight-bit index registers, which contain the equivalent of the displacement byte of the 6800; instructions that use them need a 16-bit base-address. The op code must be followed by two direct-address bytes (except in the 6502 zero-page-indexed mode, where only one is needed since page zero is implied), to which the index-register value is added to generate the true address.

In an indexed loop, the instruction can access consecutively (by index-register increment or decrement) no more than 256 locations because the index "wraps around." However, the index can do double duty as a loop-counter and can control the address of any number of instructions (accessing different memory areas) within the loop. Very complex operations are possible.

The 8-bit indexing eliminates 16-bit on-chip base-address registers (of which there can only be a limited number) by having base-addresses specified in the *program* locations (following each indexed instruction op code). Not only the address-range is restricted (to 256 consecutive locations), but so is the base-address; although you can write a RAM program that will modify its own addresses, this is a dangerous game, and not possible for a program in ROM.

This limitation of direct-indexed addressing is overcome in the 6502 and 2650 by *indirect*-indexed addressing (absent in the 8080, Z-80 and 6800, although these sets can emulate it). The true base-address of an indirect-indexed op code is stored in two contiguous RAM locations that function as a 16-bit "off-chip register." The op code must be followed by two direct-address bytes (cut to one in the 6502 because page zero is *always* implied) that specify the location of the "off-chip address register" in RAM. The operation picks up this "indirect" address (first the low, then the high) and adds the index value (in the on-chip index register) to generate the true memory address.

Although fully automatic, all this work takes time, so execution is slower (by two cycles in the 2650, but only one in the 6502) than direct indexing. But the addressing capability is now limitless, since the "indirect address register" is modifiable at will. In fact, this is a device that allows you to create in RAM as many 16-bit base-address-remembering registers as you may require, and is conceptually similar to the 6800/6502 transformation of zero-page locations into 8-bit "off-chip registers." Although access to them is slow, the supply is far greater than that available on-chip, even in the Z-80.

This may be why Adam Osborne believes that 8-bit indexing is "more powerful" than the 6800/Z-80 16-bit indexing mode. However, it seems to me that there are complex trade-offs such that 16-bit indexing will be faster and more efficient in some operations.

In every design, the program counter is a potential 16-bit base-address register, and all (except the 8080) use it as such in relative-branching. Only the 2650 extends the concept of PC-relative addressing to other instruction types, which can access memory locations *in the program area* by a one-byte address (added to the program counter).

There is some operational resemblance to the zero-page concept, but programs carry their "personalized off-chip registers" along with them wherever they may be located in memory. This extraordinary mode needs careful structuring; most of its power is lost if the program is "frozen" in ROM, and this may be an instance where the 2650 designers' imagination simply ran wild!

The Status Register

It is possible for a "thinking" instruction to be encoded so that it will cause a program skip, branch or jump if its operation yields a special state (such as all the bits in a register becoming zero). The DJNZ (Decrement B and jump relative if B≠0) of the Z-80 is of this type. However, if every possible "thought" were to be coencoded with every possible "action," the instruction set would become very complex. It is more practical to cause each "thinking operation" to set or reset one or more status flag bits to control the operation of *subsequent* jump instructions.

Although these bits are usually independent, it is convenient to have them in a *status register.* One reason is that whenever a running program is interrupted, its current status must be saved (usually in the stack) because the interrupt-servicing program may alter some status flags; before the interrupted program is reentered, the prior status must be *restored* for it to work properly. Also, it is often useful to check the content of the status register when you're tracking down bugs in a program.

"There are also 'non-testable' flags that control the operation of non-jump instructions."

There are variations in the status flags used by different designs. The common ones are the all-bits *zero* flag, the *sign* flag (set if bit 7 of a register is set to 1 by an operation) and the *carry* flag (a kind of "ninth" bit, set or reset by a variety of arithmetic or shift or rotate instructions). The 8080 also has a *parity* flag (set if the number of 1 bits following an operation on a register is even, and otherwise reset); the Z-80 retains this flag but assigns to it two quite different meanings (*parity* in non-arithmetic operations, but *overflow* in arithmetic ones) that in practice never conflict.

The 6800, 6502 and 2650 have no parity flag, but all have the *overflow* flag (set if there is a carry out of bit 6 in "signed binary arithmetic," where only bits 0 to 6 are numeric, bit 7 being the sign flag, + if 0 and − if 1). In all but the 2650 (as usual, somewhat eccentric) these flags (zero, sign, carry, parity and/or overflow) are individually "testable" by a pair of instructions (jump-if-flag-set and jump-if-flag-reset).

There are also "non-testable" flags that control the operation of non-jump instructions. In the 8080 only the *auxiliary carry* (out of bit 3 in arithmetic operations), which is useful for decimal arithmetic, is in the status register. Other flags (such as the *interrupt-inhibit*, which disables the interrupt pin of the processor) remain invisible on-chip. Unlike the testable flags, which are automatically set or reset by "thinking" operations, the non-testable ones are set or reset only by special instructions.

However, there are also special instructions to set or reset some of the testable flags. Thus you find (in all but the 2650, which has unusual instructions for program control of any or all flags) a specific set-the-carry instruction, highlighting the major role played by the carry in "thinking" operations. The 6800 and 6502 also have a clear-the-carry, but the 8080/Z-80 have a complement-the-carry (set it if clear and clear it if set).

The 8080 CMP A instruction (very little used, since it compares the accumulator to itself!) *could* be used to clear the carry (but also set the zero) flag; this is one of many possible examples of how "missing" instructions can be emulated.

In most designs, the testable flags are not affected by move instructions. In the 6800, most moves affect the zero, sign and overflow flags. The advantage is that a program "learns" something about a bit-pattern whenever it handles it. The disadvantage is that these flags become highly "volatile," so that a status from a previous "thinking" operation is lost even though it may be needed later. The 6502 design compromised: Only the zero and sign flags are affected, and only by *moves into* the on-chip registers.

Nevertheless, status-saving is very important. The 6502 has instructions that push or pull the status register directly into or out of the stack. The 6800 status-save is more awkward, since the status register content must go through the accumulator to get into or out of the stack. The 2650 status-save is similar (but since it lacks a conventional stack, the save is in some other memory location). The 8080/Z-80 cannot save *only* the status register; their PUSH and POP instructions must simultaneously (and slowly) save and restore the accumulator.

The many "thinking" instructions are, in effect, "set-flags-if" instructions. Much of the art of programming consists of using these in many different ways (ranging from the obvious to the fiendishly clever) so that each of an infinite variety of possible conditions gets translated into some unique status of from one to four testable flags (with no more than 16 distinct conditions able to exist at a given moment). For truly complex operations, this is very restrictive; this limitation is overcome by constructing in memory specific *decision tables* for each problem, using as many bits as may be necessary, and creating switching networks of far greater complexity than those within the CPU, and acting like "off-chip status registers."

One of the more promising innovations is the use of status bits to alter the interpretation of an op code. This is exemplified by the "decimal flag bit" in the 6502 status register. When this bit is set, all 16 add-and-subtract op codes do automatic decimal-adjusting. In effect, this bit doubles the number of arithmetic op codes. Other designs need a decimal-adjust instruction after each add or subtract instruction so that arithmetic loops run more slowly.

The Signetics 2650 uses its WC status bit to cause its add, subtract and rotate instructions to work either with the carry (WC set) or without it (WC clear). This kind of enrichment of an instruction set is likely to be more widely adopted than the Z-80 multibyte op codes. Its implementation requires a lot of thought, since program-setting of status bits becomes a nuisance if it has to be done frequently!

Things to Come

The 8080 and its rivals were made possible by LSI technology, although the Z-80 is an early product of the new era of VLSI (very large-scale integration). VLSI is now primarily being used for designs that combine a CPU, I/O, ROM and some RAM all on a single chip (Intel 8048, Mostek 3870, etc.). These are special-purpose controllers aimed at a mass market of millions of units.

Those who have read the articles on Microelectronics in *Scientific American* (September 1977) know what a fantastic *technical* achievement a VLSI chip represents. The *intellectual* achievement of creating a superior VLSI CPU chip will be far more formidable. The Z-80 strikes me as an awkward first try, a necessary first stage of a new learning curve. The 8080 (following an amazingly fast learning curve up from the 4004) was a work of creative genius in LSI. By comparison, the Z-80 is only an immense add-on, achieving much greater power but losing in elegance. This is hard to define: Many elements, all essential, all precisely right, fit into one perfect entity. The Taj Mahal is elegant; the Pentagon is not.

"Most of the creative ferment of the past decade was fostered by *small* companies."

Everyone knows that no existing design even comes close to being the ultimate one, and that far superior ones will become available in the near future. Before trying to guess (I have no insider knowledge!) what they will be like, we may wonder whether it is possible for giant corporations to harbor and nurture the highly creative and individualistic minds to whom we owe the revolutionary microprocessor designs. Most of the creative ferment of the past decade was fostered by *small* companies. Intel has become a giant, Zilog is backed by giants, MOS Technology was taken over by Commodore, Signetics by Philips Eindhoven.

Although large organizations can provide large resources, they are directed by an entirely different kind of mentality. It can be argued that giantism crushes creativity (in the automotive field, where do we find front-wheel drive with CVCC and rotary engines?) and replaces genuine value with shiny packaging and advertising.

Another crucial question is: Has VLSI made the 16-bit CPU the wave of the future? Here we have already seen the major minicomputer manufacturers (DEC and Data General) trying to counter the threat of major IC manufacturers (especially Texas Instruments) by downshifting to single-chip CPUs, such as the LSI-11 and microNova, in the hope that their highly developed operating systems and other software will give them a decisive competitive advantage.

Anyhow, it seems to me that Intel will probably not do any major redesign of the 8080 (beyond the enhancement at the electronics level in the 8085). The effective power will be enhanced by new LSI support chips (i.e., the 8275 CRT-controller and 8279 KB-I/O interface) that relieve the CPU of much drudgery, freeing it for thinking. The familiar stacks of PC boards, loaded with armies of chips, are on the verge of obsolescence. If Intel should decide to compete against the Commodore PET, etc., its entry could be based on a mere *handful* of chips, with a price/performance ratio that would be hard to beat.

As I stated at the start of this article, there are not many tasks that the 8080 instruction set cannot do very well, so resting on these laurels may well be the right strategy (and the Intel executives have so far proved to be skillful strategists). Although I have not carefully studied the 8048 design, which has many resemblances to the 8080, it is likely to be a strong contender at the lower level. This leaves the new 16-bit 8086, which I expect to be as revolutionary as the 8080 was, simply because it is sure to be very far up on the learning curve of microprocessor design.

In anticipation of the 8086, it is a certainty that Intel competitors are engaged in intensive design efforts, which may not be crystallized until the full power of the 8086 is known. The Z-80 designers (who were the 8080 designers), at last free of their desire to retain compatibility with the 8080, are sure to try to create something in VLSI that will not only be big but also wonderful. The dilemma is: Should you go whole-hog for 16-bit operation (minicomputer-style) or allow variable-bit-operation (as the TMS 9900 does)? Many operations don't really *need* 16 bits.

By now, most people know that the 6502 design arose from the discontent of some 6800 designers. They retained some 6800 elements, dropped others and added new ones, but left 105 op codes unused (consciously labeling them for "future expansion"). That this set — in an obviously unfinished state — can compete effectively with more complex ones proves the overriding importance of architecture and speed over mere size.

It is hard to guess what direction the VLSI expansion of the 6502 will take, since so many options are wide open. Nevertheless, the "feel" of the 6502 suggests that it will not add as many new on-chip registers as the Z-80. Its extreme one-accumulator orientation suggests that this will be expanded to allow many 16-bit operations. If the original designers are still involved, its new instructions are likely to add quite generalized power, not like the specific subroutine type of the Z-80 block move or search.

Motorola has already done some enhancing of the 6800, though neither the 6801 nor the 6802 is a major upgrading of the kind that competitive pressures will eventually require. If I were a Motorola executive (luckily for them I'm not!), I would be wondering whether the twin-accumulator concept, which would have to be retained in a compatible upgrading, is viable. As for the 2650, that I (and surely many others) admire its ingenuity cuts very little ice since it has not sold well, and Philips executives must be pondering its fate.

In the next few years, imponderables such as the economic climate and the intuition of executives will play a more important role than the brilliance of designers (in whose honor I wrote this article). Peering somewhat farther into the future, I think it likely that the creative genius of the Japanese, who have been making giant strides in computer technology, will be entering the picture.

Existing designs have enough momentum so that they will persist for quite a while. Some differences between them were revealed in the BASIC timing comparisons by Rugg and Feldman (June and October 1977 *Kilobaud*). Such tests may reveal more of the moronic nature of BASIC than of the ultimate power of a microprocessor. The efficiency of even the finest instruction set gets degraded by older, human-oriented high-level languages like BASIC or FORTRAN. Only a stupendously large and costly optimizing compiler program can translate them into efficient machine language.

In a talk at the 1977 WESCON, Carol Anne Ogdin referred to these older languages as "dinosaurs" and stressed the superiority of newer ones such as PASCAL and FORTH. The microcomputer revolution will not be completed until the gap between computer language and human language is bridged, even though the gap between personal computers and megacomputers is narrowing rapidly.

Professional programmers (a very special breed) now function as the language bridge, and have not yet succeeded (though IBM tried) in writing computer programs to replace themselves! Unless human language evolves into something more logical and less ambiguous, perhaps they never will; but it is the next great challenge to creativity. What's the good of having a genie at your beck and call, if you can't tell it what to do? ■

An early view of the next generation. The above was completed before any of the new designs had materialized. Several are now available, but it is not possible here to do any in-depth evaluation. With *thousands* of possible op codes, the 8086 looks impossibly complex to me. It will encounter competition from 16-bit rivals such as the Zilog Z-8000 and Motorola M-68000, and other manufacturers may risk entry into a small and overcrowded market. All these hardware marvels lack the tested software that can make them useful.

One hint that the day of the 8-bit machine is not over is the forthcoming Intel 8088, a dual 8-bit processor on one chip that will include much of the advanced thinking of the 8086. Intel executives apparently decided they needed something better than the 8080/8085 to stay competitive in the low-cost microcomputer market.

A formidable competitor is the Motorola 6809, with enhancements so great that I feel its performance will, in most areas, excel that of all older designs. It retains the twin accumulators of the 6800, but some instructions use A and B as if they were one 16-bit accumulator. It adds a second index register (Y, exactly like the 6800 X) and a second stack pointer (U, similar to the 6800 S but never used for automatic storage of return addresses or registers). Both stack pointers can also serve as index registers.

This is part of an enrichment of addressing modes. However, as in the Z-80 the modes must be specified by a second op-code byte, sacrificing speed for power. Probably to compensate for this loss, the fast "zero-page" addressing mode has been made more flexible: the "page" address is stored in a new 8-bit programmable "direct-page register," so that *any* page in memory can be addressed as if it were the 6800 zero-page.

Also, direct-page addressing, needing only one op-code byte and one address byte, has been made indexable as in the 6502. PC-relative addressing, as in the 2650, is implemented, together with "long" (16-bit) PC-relative branching, so that programs can be written to run *anywhere* in memory. Many of the valuable ideas of 6800 competitors have been adopted and extended.

I have not seen the full 6516 instruction set, which is said to use only one-byte op codes. This will often mean higher speed, but 105 new op codes cannot yield as much capability as the much greater increase in the 6809. In the coming generation, we may see several chips with very different instruction sets, none so superior that it can crush all others. The older designs, however, have become obsolete.