An Introduction to the Human Applications Standard Computer Interface

Part 1: Theory and Principles

To evolve into a consumer product the computer must have a standard, easy-to-use format.

Chris Rutkowski
Rising Star Industries
24050 Madison St., Suite 113
Torrance, CA 90505

Many people see the personal computer as merely a cheaper, smaller, and slower version of its larger data-processing relatives. However, it's becoming apparent that the personal computer is an entirely different type of machine, shaped by a technological evolution that should result in computers that work for people, rather than the other way around.

The proposed Human Applications Standard Computer Interface (HASCI) was designed as an important step in that evolutionary process. It is the result of approximately six years of effort, proceeding from the most general considerations to a very specific result. I will describe this process of development in two parts. First, I will explain the theory and principles behind the HASCI interface. We'll learn why the interface is needed and what it is generally intended to do. Next month, in part 2, I'll describe the actual implementation and design specifications of the interface.

Theoretical Background

I entered the microcomputer marketplace in 1975, during the very infancy of our industry. Then as now, those of us on the "inside" of the industry saw visions of microcomputers gracing every desk in the world someday, when the industry grew up.

Then as now, the consensus of opinion within the industry was that the microcomputer would be the bright star of the future. We knew it was so, but we couldn't prove it; therefore, financial backing was hard to come by. It is easy to forget that, in 1975, the microcomputer was not yet the darling of the venture-capital set; Wall Street had taken a bath on computer companies just a year or two earlier during a recession, and our claims to have found a magic formula for success fell upon jaundiced ears. The one precept on which everyone seemed to agree was that *no one could predict such a fast-changing market more than a year or so in advance.*

In the intervening years, I've heard that phrase a hundred times or more; I suspect you have too. It's one of those pieces of common wisdom that sounds good in a speech and makes for good press: the media repeat it, the bureaucrats who read the media repeat it, and the media repeat it again. This sort of publicity is discouraging. Nevertheless, with blinders firmly in place, enterprising companies continue the struggle to design their way into a murky future.

The Challenge Accepted

In 1976, during the first Atlantic City Computer Show (thank you, John Dilks, for your vision) the "can't predict" motto rang loudly in my ears. It was plain at the time that if our industry was to put a microcomputer in every home, the two essentials were money (lots of money) and manufacturing capability. It was also plain that prediction precedes production; no company executives in their right minds would put up the megabucks necessary to develop the microcomputer without knowing where that development would lead. Tooling for extreme mass production costs millions; for that kind of investment a one-year prediction lead was far from adequate.

Thus it was not until 1981 that IBM entered the personal computer market. It is to that company's credit that its machine avoids most if not all of the inanities perpetrated by IBM's peers. Witness the pitiful efforts of most minicomputer companies to introduce personal computers over the last few years; most if not all of these machines were obsolete before the first carton was shipped. The only prediction those companies could make was that their profitability would plummet within a few years if they couldn't penetrate the microcomputer field. And in fact, this has come to pass.

On the subject of market predictability, many heated discussions took place comparing various hardware and software components, but I realized that further arguments on the advantages of one processor over another, one operating system over another, or one language over another were wasted words unless you knew how those items related to the evolutionary path of the industry—the yardstick for measuring potential worth. And I took it upon myself to research the question of prediction.

Research Methodology

I chose a most unscholarly methodology, but one well suited to the task. Rather than dig through stacks of ponderous marketing tomes in dusty libraries and research what had already been done, I reasoned that any worthwhile work was probably buried so deep as to be invisible. After all, if viable principles of predictability (in terms of the computer market) were available, why weren't they in use? I therefore decided to conduct a broad survey of earlier technological industries, narrowed down to those that had reached the mass consumer markets.

I scanned the marketing history of twentieth-century Western civilization, seeking instances where highly technical products were converted into mass-market commodities over a relatively short period of time. If you think this through yourself, you'll find several examples, including radio, television, electric lightbulbs, and of course the automobile.

I soon perceived a pattern in the emergence of these products that either had gone unnoticed before or had been erroneously classified as unimportant. To illustrate this, let's consider how one such product evolved.

Case Study: The Automobile

I ask you to turn your mental clock back to the year 1905 and consider the state of the automobile market at that time.

First, the automobile was nowhere near mass production yet. Most manufacturers were backyard experimenters (I suggest that the phrase "garage shop" must have originated somewhere around here). They were technology freaks working on the hottest gadget then conceivable.

Peruse some of the popular literature of the time; items about the coming wave of horseless carriages abounded. There were literally hundreds of fledgling manufacturers—every bicycle and carriage shop fancied itself to be the next Pullman Company (the coach manufacturer that became very successful making railroad cars). And what cars they made! Although most had four wheels, their similarity to the automobile of today stops there. Some of those contraptions were steered with tillers like a boat, while others had reins like a wagon. A few had three wheels. They had handbrakes on the right and foot brakes on the left; fixed throttles and throttles on the dash. Few if any were closed in with a roof. And not one was truly practical for the average person. (Does this sound familiar?)

It's easy to look back at these early machines and say, "How quaint." It's easy to overlook the fact that every single engineer and user had his or her own idea of perfection. Ideas abounded, and while each no doubt had some validity, no one could agree on what was valid and what wasn't. In modern terminology we would say that the engineers were coming up with possible *design elements* that were combined almost at random into *architectures* (a collection of design elements).

Now turn your mental clock forward to 1925, and consider again the state of the automobile. Things had definitely changed. The auto was in mass production. Hundreds of thousands per year were being added to a blossoming economy. And more important, we find that every car on the road had a steering wheel and a throttle, brake, and clutch on the floor. It had windshields and headlights. We find that, with the exception of a relatively few details, you would be able to climb into the typical automobile of 1925 and drive it away.

Architectural Stabilization

By 1925, the architecture of the automobile had become standardized. That architecture has not altered significantly in the ensuing 57 years. Today, the products that you see parked on the streets and recognize as automobiles are architecturally identical to each other. No architectural difference exists between a Subaru and a Rolls-Royce.

If you check other technical marketplaces (for example, that of television), you will see that this same phenomenon has occurred. First, independent engineers developed a wide variety of design elements. Then their ideas were assimilated and adapted until, now, the architecture has ceased to change. I call this phenomenon *architectural stabilization*.

In the period following architectural stabilization, the design effort and creativity that were previously engaged in the random creation of architectures is now geared toward the refinement of the design elements that comprise the stabilized architecture.

This point is crucial: a stabilized architecture ends the game of random invention and redirects the tremendous creative energy of engineers to a better-focused goal—the improvement of the design elements. The improvements realized may be quite substantial. For example, consider suspension systems. Before 1925 no automobile had a suspension system truly worthy of the name. In the ensuing years such comfort items evolved beyond all prediction.

Thus we see that while architectural stabilization may seem to limit certain aspects of design, it can and should precipitate a design revolution far more exciting than pure laissez-faire engineering.

The Mechanics of Stabilization

A detailed analysis of the marketing factors that affect each step of a product's life span is beyond the scope of this article. However, the results of my investigations revealed the following sequence of events leading to architectural stabilization:

- First, engineers (or technical specialists) conceive of a new product class and build it for its own sake.
- Engineers then use the product.
- If the product promises to fundamentally revise the quality of life for its users, the number of participating engineers will swell. (They sense the market potential and have visions of earning wealth and fame.)
- Eventually, this growing enthusiasm gains popular notice, and certain nonengineers purchase the product. These nonengineers find the architectures designed by engineers to be difficult to use; they recommend improvements *but are willing to undergo difficulty in using the product.* They are "enthusiasts."
- Increasing demand increases production, which lowers the product's price.
- People who are not willing to undergo substantial difficulty in using the product purchase it. These users are disappointed by the currently available products. They are consumers—they want the benefits without the difficulties.
- More communication about the product occurs in the popular media.
- If the product does not fill a truly fundamental need, its popularity subsides, leaving a core group of enthusiasts that will then grow at a slower rate. The product will show a gradual evolution of architecture across time.
- If the need for the product *is* truly fundamental, demand continues to grow, but actual market growth may slacken.
- This growth of demand (potential market) motivates engineers and enthusiasts to redesign the product to make it easy to operate. In other words, swelling demand precipitates the creation of a human interface that makes the device easy to use.
- An easy-to-use version of the product finds a ready and willing market.
- The first manufacturer to implement ease of use soon gains a market edge.
- Other manufacturers either follow suit or perish.

This sequence, or one closely analogous to it, occurs in the evolution of all product markets. For the microcomputer market, certain factors have become clear. First, the microcomputer market has not yet achieved architectural stabilization. Second, the microcomputer appears to have all the elements necessary to cause architectural stabilization to occur; that is, its impact on users is of sufficient importance to force stabilization to occur. Third, the microcomputer market has currently reached that step of increased popular demand that should precipitate the development of an easy-to-use version of the product.

It's no accident that human-factors engineering has risen to such prominence over the last year. It is a natural and necessary step in the evolution of the product classification from a *technical specialist's* market to an *enthusiast's* market and finally to a *consumer's* market.

Thus the development of a human interface coupled with mass-production technology should be the key to opening the consumer market for the computer.

Let me digress for a moment to observe that architectural stabilization occurs at many levels of observation, not only with products such as those discussed here but also with subproducts—raw materials and their elemental forms. All undergo microcosmic architectural stabilization. Likewise, stabilization tends to occur in structures far larger than products: nations, families, and businesses. All exhibit variations of this same phenomenon. It thus appears that architectural stabilization is a fundamental mechanism of systems evolution: the imposition of a mutually accommodative interface between two counter efforts, thoughts, forces, or intentions.

In Search of a Human Interface

The many clues that led to the de-

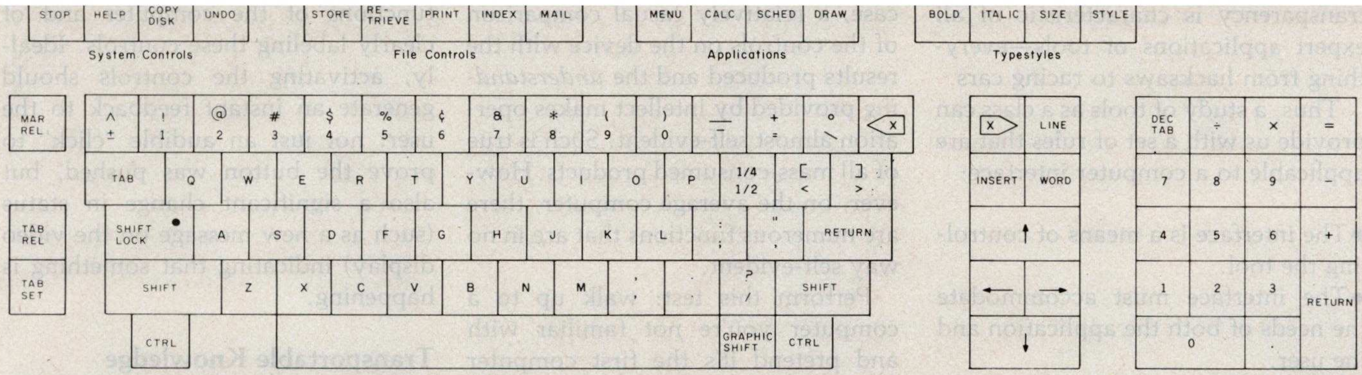

Figure 1: *The Human Applications Standard Computer Interface (HASCI) keyboard divides the computer system into a series of menus that link the user (as pattern recognizer) to the computer (as symbol manipulator). Virtually every application requires that certain fundamental actions be performable, and these fundamental actions are placed directly on the keyboard.*

velopment of our proposed human interface (HASCI) came primarily from fields far removed from the normal realm of computer science. The difficulty was this: before an interface could be designed, the actual relationship of man and computer had to be defined. I had concluded early on that the entire question of artificial intelligence could be ignored in the design of an interface, which was fortunate since no workable definition of intelligence exists. Rather, an interface involves questions of capability: What can people *do*, and what are they *good at*? This approach proved very profitable.

Even if you were offered a million dollars to manually multiply two times two a million times, you would have a very difficult time completing the task; most humans would be psychologically incapable of completing the job. Yet virtually any computer can do it easily and with remarkable speed. Conversely, such problems as "recognize a certain person's voice," solved by almost any infant (especially if the voice belongs to the child's mother), still represent a major challenge to even the finest computers and programmers.

An analysis of these problems suggests that people are much better than computers at recognizing patterns, while computers are much better than people at manipulating symbols.

Following this logic, the ideal relationship of computer and user should involve the computer as a symbolic manipulator and the user as a pattern recognizer.

This explains the overwhelming popularity of word processors and spreadsheet calculators. One manipulates words and letters, the primary symbols of man. The other manipulates numbers, man's second most important symbol set.

It follows that a complete computer for the typical user should provide the facilities for manipulating all the primary symbols of man (words and letters, numbers, general symbols or drawings, and the temporal relationships between these symbols—time).

We usually manipulate these symbols on pieces of paper, which if saved for later reference may be generically called documents. We require a means of storing, retrieving, and indexing these documents and of communicating their contents to some other person.

These considerations gave birth to a hardware-software synthesis. Rather than take the accepted path of generalization—designing the computer interface to accommodate *any* imaginable task—we conceived of an interface that would be specifically designed for symbolic manipulation tasks as described herein. The HASCI keyboard (figure 1) was the result.

Fundamental Principles

The described theoretical explorations led to the evolution of a number of principles that form the rationale of the HASCI standard. A detailed examination of these principles follows.

The Computer Is a Tool

The computer as symbol processor and the user as pattern recognizer complement each other well. In this arrangement, the weaknesses of each can be ignored; their strengths added together form a synergetic whole far more powerful than either, and such a blending of strengths is the functional property of any tool.

A hammer uses the advantages of a steel working face (hardness and mass) combined with the advantages of the human arm (motion and leverage) joined by an interface (the handle) to perform some task dictated by intellect. Similarly, the computer uses the advantages of electronics (rapid manipulation of symbols) combined with the capabilities of the human mind (pattern recognition), joined together by an interface (keyboard and screen) in order to perform tasks dictated by intellect.

In an ideal situation the relationship of user and tool approaches one of *transparency*. The user is able to apply intellect directly to the task; the tool itself seems to disappear. This transparency is characteristic of all expert applications of tools—everything from hacksaws to racing cars.

Thus, a study of tools as a class can provide us with a set of rules that are applicable to a computer interface:

- The interface is a means of controlling the tool.
- The interface must accommodate the needs of both the application and the user.
- The interface itself must present the information necessary for its use.
- Mastery of the interface may require practice.
- With mastery, the interface must become transparent to the user.

Clearly Label the Controls

Televisions are easy to operate. They have a limited number of controls. A stereo may have far more controls—complex models have dozens. But in each case, the controls either produce an immediately observable effect or are very clearly labeled as to their function. In each case, a relatively casual comparison of the controls on the device with the results produced and the *understanding* provided by intellect makes operation almost self-evident. Such is true of all mass-consumed products. However, on the average computer, there are numerous functions that are in no way self-evident.

Perform this test: walk up to a computer you're not familiar with and pretend it's the first computer you've ever looked at. Then guess how to save or load a file of information. Get it? No way! You've got to study the manual and learn the code. You're required to learn and memorize the information. A little memory requirement is a positive thing: it makes the skill more valuable. But when you must rely on memory, the interface is effectively in your head rather than on the machine. (Imagine the potential hazards if a power saw were designed this way.)

We therefore see the necessity of providing controls for the major functions of the computer and of clearly labeling these controls. Ideally, activating the controls should generate an instant feedback to the user: not just an audible "click" to prove the button was pushed, but also a significant change in status (such as a new message on the video display) indicating that something is happening.

Transportable Knowledge

The concept of *transportable operator knowledge* refers to the fact that users of consumer products expect and demand that the skills they acquire in learning to operate one machine be applicable on any machine of the same class.

For example, consider the typewriter. There are minor differences in the placement of certain controls, but a user who has learned on one typewriter can pretty well sit down at any typewriter in the world and type away. This is not because the task is overly simple: a typist must learn to manipulate a hundred or more keys, switches, and levers to operate the machine in differing circumstances. However, the typewriter as an architecture is fully stabilized to perform its appointed task. All typewriters have carriage returns, a means of setting tabs, a margin release, etc. These are sufficiently clear that an inspection of any machine rapidly reveals how to perform these functions.

Now consider the computer. Nearly every software writer and hardware designer has a unique way of telling a computer to save and load a file. Even though virtually every operator needs to perform these functions with great frequency, every time you change machines or programs you have to learn how to save and load all over again. (This is not to say that any one of these ways is wrong; rather, that on a consumer computer the basics should be done in one workable, learnable way.)

It is ironic that the data-processing and computer science industries have given so much attention to transportability of software. The benefits of this transportability appear to accrue primarily to programmers, and while it's understandable that people should create tools that they themselves need, transportable software eases only the programmer's burden. Transportable operator knowledge serves all users.

In a similar vein, it becomes clear that arguing the benefits of 16-bit versus 8-bit machines is analogous to arguing the merits of 8-cylinder versus 4-cylinder engines. Your choice should be based on how much payload you expect to haul, *not* whether you get a steering wheel with the vehicle. Performance from the consumer's standpoint is the ease with which desired tasks are accomplished: fast and difficult is still difficult.

When we approach the matter in this light, we realize that consumers will expect computers, both complex and simple, to have interfaces that are virtually identical. For all intents and purposes, anything that can be run on a 68000 microprocessor should be able to run on an 8080; the difference should be in how fast and how much, not *how*.

In terms of operating systems, while Unix may have certain advantages over CP/M (or vice versa), this is of no interest to the average user. Operating systems are tools for programmers. The symbol manipulator should function as an intelligent interpreter between the user and the operating system, and that interpreter should function almost identically on *any* operating system. (Most applications programs are considered as running *under* an operating system. The interface, however, should be considered as running *over* the operating system. It actually mediates between the operating system and the user just as would a programmer. In this case the interface is the expert who makes the difficult seem easy.)

Design Out Technical Choices

Early in the days of the S-100 bus, I put together a kit for a serial interface board (the 3P+S). It was quite marvelous and went together easily, that is, until I got to the "jumper options." There were dozens of options. You could configure the system just about any way you might imagine: number of data bits, parity, stop bits, and so on. All fine except for one small problem: I was a novice computer user and had no possible way of knowing which of these options served my purposes. After a few days of messing about and getting nowhere I asked a computer expert for help. He had the board configured for my system in a matter of minutes.

This highlights a typical problem. Because a computer can be configured in many ways, experts often want to build in every conceivable option because "you never know what the user may want to do with the system." However, we have already accepted the concept that the consumer computer is a tool for manipulating symbols. So we do have an idea of what the user will want to do.

Even a so-called user-friendly system may have an incredible array of choices. I recently bought what was billed as a user-friendly electronic mail system. It offers me options of stop bits and parity and data rate—just like the old 3P+S. It also presents a vast array of choices of how to send the data: compacted format, binary code, straight ASCII (American Standard Code for Information Interchange), and more. The designer of this code apparently confused "user-friendly" with "all possible options accessible." (The term "user friendly" must surely rate as the inanity of the decade. When was the last time you thought of a tool as "friendly"? "Usable" and "useful" are the appropriate operative terms.)

Burdening the user with decisions concerning technical choices in no way addresses the task to which the tool will be applied, i.e., the manipulation of symbols. The system should *automatically* test the lines and choose settings appropriate for the circumstances. The user is then free to concentrate on the act of manipulating symbols rather than on the hardware. (This is how transparency is achieved).

Thus a rule of thumb evolved: technical choices irrelevant to the symbol-manipulation task at hand should be eliminated from the user interface.

Predictability

In order to ease the chore of learning the HASCI system, we have attempted to keep the system as straightforward and predictable as possible. We try to allow different operations to be performed in a similar fashion whenever possible or appropriate. This does *not* require that there be only one way of doing each function, however.

For example, you can move the video cursor by pressing cursor keys on the HASCI keyboard. These arrow keys, when pressed in combination with the Shift key, or in combination with arguments such as WORD, move the cursor by different units. Even complete novices experience little difficulty with this scheme. Learning is accomplished by inspection and some experimentation.

Burdening the user with decisions concerning technical choices in no way addresses the task to which the tool will be applied.

However, experienced users may find this method cumbersome; moving their fingers from the main keyboard to type on a different group of keys slows them down. For the more-than-casual user, Control-letter functions (where you press a control key and a letter key simultaneously instead of a separate cursor key) are much quicker. Therefore, the HASCI processor also recognizes control key combinations for these same functions.

In this fashion both the novice or occasional user as well as the professional are well accommodated.

Simplicity

In designing a user interface it's important to keep simple things simple. More complex functions may be handled in a more complex manner because these will typically be used by more experienced users.

It's easy for experienced users to forget just how overwhelming a microcomputer can be. We attempt to judge the value of any product solely by the number of features offered for a given price. But what of the neophyte? Novices can assimilate only so much in one gulp, and that gulp is apt to be a small one.

A year and a half ago I tested the concept of a seven-function word processor, analogous to a four-function calculator. My premise was that seven functions are absolutely necessary for a useful screen editor: text entry, moving the cursor, insert character, delete character, save file, load file, and print file. With these functions, you can handle almost any word-processing task. More advanced functions can expand these capabilities and increase ease of use.

I tested the validity of this screen editor on a number of nontechnical users and found that they could be taught these basic functions in a few minutes of verbal instruction. And with only these functions, the system was truly useful. In fact, some of the users never asked if there were more functions. Even such a bare-bones editor proved to be a very useful tool, about as far ahead of a typewriter as the typewriter is ahead of clay tablets and sharp sticks.

I am not recommending that a screen editor be limited to these functions. On the contrary, I believe that constantly increasing the power of the system to manipulate symbols is mandatory and very desirable. However, the basics must not be obscured by the complexities of more advanced functions.

The HASCI standard calls for a selection of the most desirable functions to be placed directly on the keyboard with dedicated function keys. Many users will never venture beyond this—they will never feel the need to do so. More complex functions can be accessed via the use of Control-letter functions for access to specialized menus.

Defang the Computer

Over the years I've seen dozens of ways to get bitten by a computer. For example, one popular computer uses 8-inch drives for increased storage. There's a catch, however: the disks absolutely must be removed from the machine before it is turned off; failure to do so results in absolute and complete loss of all data on every disk in the system. Now it's easy to say, "Always remember to take out the disks," but in fact even experienced users occasionally fail to remember. They get so wrapped up in the job they're doing (as they should) that they forget that the hardware itself needs this critical piece of attention.

Another computer hazard shows up in the use of editors. Have you ever deleted something and then wished you hadn't? I'd be surprised if you said no. I know of no more awful feeling than to have just erroneously deleted a document that I put a week's work into. The system should be smart enough to alleviate or entirely eliminate these dangers.

One answer to this problem is to deliberately place a slower menu structure in the way of any potentially destructive action. This often takes the form of a query, such as: "Your action will cause (*a certain consequence*) to occur. Please confirm this before I continue."

Another solution would allow you to change any decision even after the computer has acted on it. This is expressed as an Undo function key. Literally, this key allows you to undo or reverse your decision. For example, pressing the Undo key within a menu would take you to the prior menu. Pressing Undo within an editor after you had made a deletion would bring back the deletion. However, in order to fully defang the system, you should not allow the operator to undo *everything*. For example, suppose you just typed in three pages of text and pressed the Undo key: would you want the system to Undo your three pages of text? Hardly.

The HASCI concept requires that designers allow people to be people, not machines. Even the best of us occasionally forgets the right sequence or fails to do some required part of a protocol. It is the responsibility of the systems designers to defend the right of users to be human beings.

One shortcoming of many computer systems involves the use of modes. I don't see modes as inherently bad; certainly a human being does only one function at a time—you can't do order entry and write a letter at the same time. However, the problem in most system designs is that it is very difficult to change between functions.

Suppose you are merrily typing away and you need to calculate a few numbers for the document. Should you have to save the file, load the calculator, perform the computation, print the results, and reload the editor, all just to enter the result of your calculation? That's the trouble with modes. They make it difficult to change between functions and trap the user in the complexities of system integration. Common symbol-manipulation tasks and document-manipulation tasks should be accessible with push-button ease. HASCI allows you to change functions at will by pushing the appropriate control. Furthermore, when appropriate, if a prior function is recalled, you should find that function configured as you left it.

In an ideal implementation of HASCI, you should be able to turn the machine off, then power it back up and find it just as you left it, even if it was running a program at the time.

What You See . . .

The phrase "What you see is what you get" summarizes a concept of text display on word processors whereby formatting commands no longer appear as obscure codes imbedded in the on-screen text. Instead, the commands appropriately modify the displayed text so that you can see your specified formats on the screen before you print out hard copy. For example, if you indicate that a line is to be centered, it will appear centered in the displayed text. In addition, if you specify a change in type style, the altered text will appear in a graphic approximation of that style, enabling you to visually distinguish it from the surrounding text.

When we got the first sample of the Epson MX-80 dot-matrix printer way back when, it already had a terrific selection of type styles available: emphasized, double-emphasized, compressed, etc. This opened up a whole new era of correspondence-quality printing, where the perfection of a fully formed character is gladly traded off for vastly increased versatility coupled with adequate legibility. The MX-80 was, of course, only the start. The newest printers now offer as many as 60 or 70 different type styles, and they also offer programmable character fonts. We may certainly expect to see the matrix densities of these machines increase very substantially over the next year or two, widening still further their performance gap over the fully formed character printers.

But then as now, the problem was that the editors and personal computers available were designed to display on their screens only one or at best two or three different type styles—far fewer than even the first MX-80 was capable of printing.

This meant that although the printers had the capability, the computers were far behind in making this capability available in anything resembling an easy-to-use fashion. Most of us have had to settle for inserting control codes using one language-like protocol or another. This is clearly unacceptable because it violates the "easy to learn" maxim.

Here is a case where very useful symbolic manipulation features are very difficult to access. The answer is to design the system with this capability in mind, make these functions easy to access, and at least where desktop units are concerned, place these changes *right on the screen*. This establishes a feedback loop which makes the system easy to operate.

"What you see is what you get" is more than a maxim. It is a crucial consideration in the effort to make the symbol manipulators—computers—easy to use.

Consumer Quality

All the above principles and guidelines add up to make the computer a consumable product. With the computer, as with any good stereo, television, or automobile, we expect to be able to gain access to substantial capabilities with little if any specialized knowledge. Manuals are for reference; you shouldn't need an advanced degree just to open the box. You should be able to set up the computer, hook up the cables in the obvious places, turn it on, and have it work right the first time and every time. Using computers to advantage should be a game that everyone can win.

Beyond Theory

Now that the theory and principles behind the HASCI system have been explained, some obvious questions arise: "How can this idea actually be implemented on a personal computer? What specific keys do we need? What should they do? And what should be displayed on the monitor screen?"

Next month, I will address these questions. I'll explain how an easy-to-use, consumer-quality computer should be designed, and I'll discuss a new computer, based on this concept, that should appear on the market very shortly.■