7 May 1976, Volume 192, Number 4239 SCIENCE

Microprocessors?—An End User's View

Raymond E. Dessy

There is little doubt that microprocessors are going to change the way instruments in research laboratories and analytical service areas are designed and operate and how they will interact with their operators. Within 5 years most new equipment will be using microprocessors to acquire analytical data, perform small manipulations on the data base, and report the results. Thus, it is important that microprocessors be placed in a proper perspective, especially since there is an inclination to view them as the focal point of an entirely new capability.

The 1960's saw the evolution of the minicomputer from a questionable relative of the large computer system into a strong competitor. Minicomputer systems are now capable of supporting large numbers of high-speed peripherals. They have an architecture that allows them to run sophisticated software systems under the control of programs called Executives and permits the user access to a variety of high-level languages such as Fortran, Basic, and Algol. In these languages the programmer can elicit complex mathematical operations with single-line commands. The tedious and expensive operation of programming minicomputers at the machine code or assembly language level (where each machine operation must be coded into a number that is interpreted by the computer) has disappeared, except for cases requiring speed and flexibility.

The evolution of the minicomputer was a result of the interactions of a growing market, lowered prices, lowered manufacturing costs, and the ability to put more functional electronic components (such as transistors) into a fixed amount of space. The last factor is best illustrated by a chart showing the number of components per square inch for several years since 1960 and the technological developments that permitted these advances (see Table 1).

The impetus for some of this development was a consumer market oriented toward hand-held calculators. These devices provide a large amount of computing power at slow speeds. By allowing key-stroke programming they offer the user liberation from the strictures that higher-level languages impose. However, it must be realized that they are not suitable for applications requiring speed, massive memory, or input/output (I/O) to external devices.

The high-density large-scale integrated circuit (LSI) technology also attracted the interest of equipment designers, who realized that the point-of-sale market was in need of very simple computer-type equipment. Supermarkets, gas stations, and other consumer areas needed devices suited to an accounting, inventory, and data communications environment. Computing power and speed were not important, but I/O capability was a necessity. Beginning in 1970 this need led to the availability of integrated circuits containing some of the elements and capabilities of the central processor of a computer. These were termed microprocessors. Their intended markets led to a design for handling data words containing 4 or 8 bits (*b*inary dig*its*) of information, since data communications and decimal arithmetic, which use binary coded decimal (BCD) notation, are oriented to these word lengths (BCD is a numbering code in which each decimal digit is represented by 4 bits).

At that time design and construction costs for instrument and control equipment were soaring, and microprocessors offered an interesting alternative to the hardwired logic (logic implemented by fixed, single-purpose circuits) currently being used. Automobile makers, washing machine manufacturers, and process control designers saw the microprocessor as ideally suited to the sensing, control, and reporting operations in the units they manufactured in large quantities.

To the instrument maker, microprocessors were and are viewed as a means of reducing the design costs of instruments and making them easier to manufacture and simpler (but not necessarily cheaper) to repair. The result is that numerous instruments are now being sold in which data acquisition and reporting is done by microprocessors. In many units a microprocessor also controls instrument operation, either alone or with operator interaction. Typical examples include a sophisticated electrochemical instrument, x-ray fluorescence equipment, and a variety of chromatographic units. The promise of complete instrument self-calibration and optimization and the potential to change instrument function drastically by simple changes in the operating program has been voiced. These capabilities, however, have yet to be demonstrated.

The availability of microprocessors has excited much interest in the hobby and scientific realm because of the suggestion that they provide cheap computer power and will give minicomputer capabilities at hand-held calculator prices. But microprocessors vary in the functions they perform easily and effectively, depending on their architecture—much more so than do the spectrum of minicomputers available. There are serious pitfalls for the novice who mismatches his application and his microprocessor solution.

The author is professor of chemistry at Virginia Polytechnic Institute and State University, Blacksburg 24061.

Because of a lack of knowledge of computer operation, and particularly of software generation, the boundaries are confused between the two areas microprocessors will affect most drastically. On the one hand are applications in which control is of utmost importance and the limited vocabulary (instruction set) and computing power entailed by an 8-bit word is of no importance—the controller area. This area is in an extreme state of flux, and will remain so until production economies and competition can no longer lead to price reductions. On the other hand, during the next 2 years high-level minicomputer capabilities will be attained with sophisticated microprocessors. In the interim inexpensive microprocessor-based minicomputers will be attractive to those wanting computing power and speed for systems that will not be required to expand indefinitely in size and power—the stand-alone computer area. Both vendors and consumers have confused these two areas either by intent or by lack of understanding.

Table 1. Number of functional electronic components per square inch of space for five dates between 1960 and 1974.

Year	Components per square inch	Technological development
1960	4	Discrete elements (transistors, resistors)
1962	40	Integrated circuits
1965	400	Medium-scale integrated circuits
1969	40,000	Large-scale integrated circuits
1974	400,000	High-density large-scale integrated circuits

The root of the problem is that although new techniques are not being introduced with this technology—we are not doing anything new, we are doing it in a different way—the devices do require something most scientists do not fully understand or appreciate, software. To perform any task, microprocessors must be programmed. And the program must take cognizance of how data are collected, how rapidly they must be acquired, and in what ways they must be manipulated. All of this takes place in the digital environment and involves mathematical and logical operations which must currently be invoked by programmed instructions in assembly language or machine language. Program generation at this level is tedious, time consuming, and costly under the best of conditions, let alone in the environment surrounding most microprocessors. To utilize a microprocessor intended for the controller area in a task requiring one oriented to the computer realm is to invite software catastrophe. To use a microprocessor that is ideal as a stand-alone computer in the control environment is financially unsound.

It is my purpose in this article to present sufficient background philosophy, language, and examples that an interested user can make his own decision properly. The article is based on the conservative application principles that an instrument automation research group has developed through extended experience with both minicomputers and microprocessors.

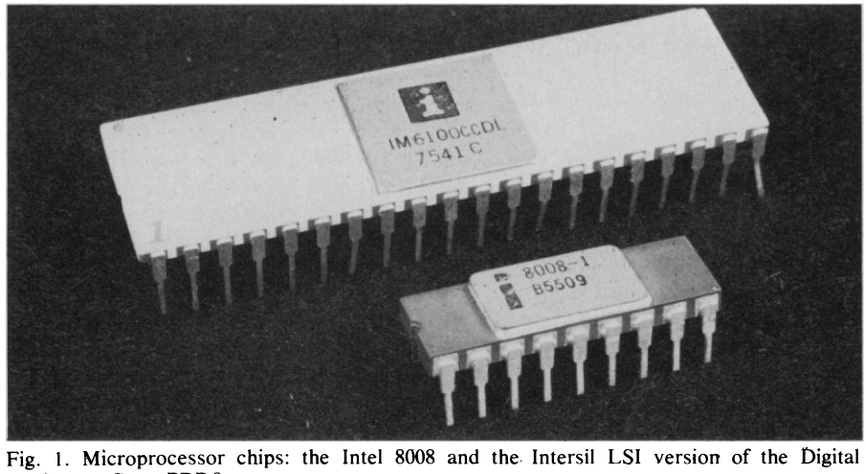

Fig. 1. Microprocessor chips: the Intel 8008 and the Intersil LSI version of the Digital Equipment Corp. PDP 8 computer.

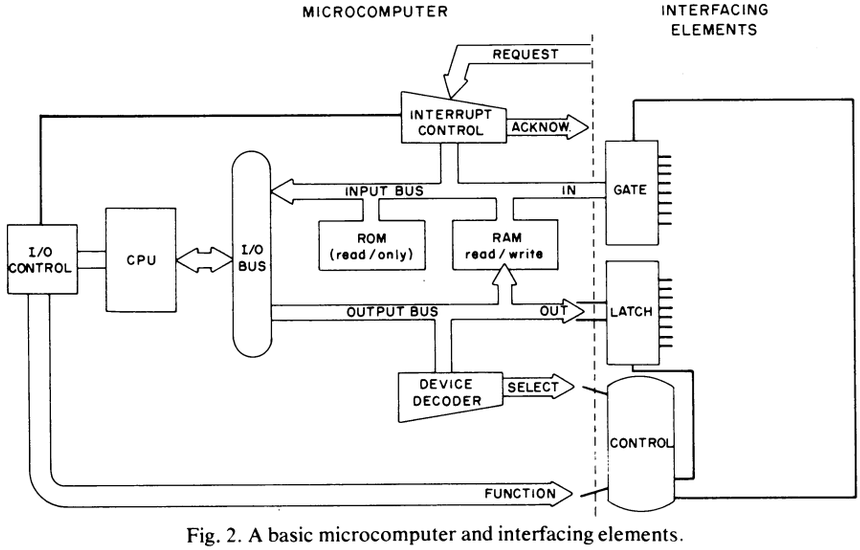

Fig. 2. A basic microcomputer and interfacing elements.

Current Microprocessor Design

Let us examine some typical microprocessors currently being vended. This will provide a real-world framework for introducing the jargon and philosophy of the area. Four separate processors will be discussed, each one more complex and powerful than its predecessor. Although many microprocessors are advertised, relatively few are easily available in unit quantities to the interested end user. The examples given here were chosen with such availability in mind.

A microprocessor is defined, for the purpose of this article, as a set of one to four integrated circuit chips, based on LSI technology, which will provide the functions characteristic of the central processing unit (CPU) of a computer (Fig. 1). To convert a CPU into a useful computer the first step is to surround it with memory in which programs and data can be stored. Memory may be of the type where information can be both stored (written) and retrieved (read) in a random manner; this is called random access memory (RAM) or read/write memory. Where only executing programs are involved, read only memory (ROM) may be used (Fig. 2). Also

needed are circuits to perform the tasks of (i) selecting a device for activation (device decoder), (ii) selecting a function the device is to perform (function control lines), and (iii) accepting and acknowledging interruptions (interrupt request and acknowledge lines). These aspects will be discussed later.

The CPU must have a way, or ways, of addressing memory to fetch the next instruction for execution, or fetch the piece of data for manipulation. The methods by which this can be accomplished are classed as memory addressing. As data are manipulated they will have to be stored in registers in the CPU. If arithmetic or logical operations are required, special registers to handle such tasks are needed, and these are called accumulators.

Finally, it is useful to have some simple means of establishing certain facts about the result of operations just performed in the accumulator—for example, the addition of two numbers. Such information is stored in 1-bit (yes, no) registers called flags, as indicated by the following examples.

Has a zero been left in the accumulator?
Zero flag: (1, 0)

Is the answer negative?
Sign flag: (1, 0)

Has a carry occurred?
Carry flag: (1, 0)

Has an overflow occurred?
Overflow flag: (1, 0)

(Overflow occurs when two positive numbers summed together give a result that is considered negative by the CPU.)

All that remains is to have some means of starting our computer up and controlling the flow of the program. This is done by setting up a register called the program counter, which contains the address (number indicating a specific location in CPU memory) of the first executable instruction, and then performing the fetch operation. At this point the program counter is set to point to the next executable instruction in memory; the program counter is an arrow that always points to the next step in the program.

These features are shown in Fig. 3 for the Intel 8080 CPU. This CPU has a single accumulator in which arithmetic and logic operations can be performed. The word length for this processor is 8 bits. Data having a range of 0 to 255 in decimal notation can be accommodated. However, 8-bit-wide instructions do not permit a highly involved mathematics and control language. The situation is equivalent to limiting your own vocabulary to 8-letter words. It is also evident

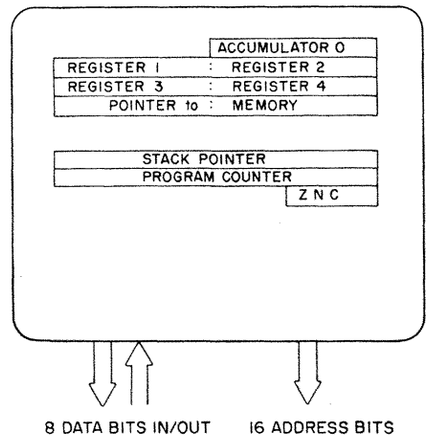

Fig. 3. Architecture of the Intel 8080 CPU.

that since each memory address must be a unique number, 8 bits of information could uniquely identify only 256 addresses. Therefore two 8-bit-wide registers in the CPU are used to form a 16-bit-wide pointer-to-memory, thus allowing the CPU to address up to 64 kilobytes of memory (a kilobyte is 1024 words, each 8 bits wide). The operation of accessing an 8-bit data word from memory requires the following operations.

1) Load pointer-to-memory with high 8 bits of address.

2) Load pointer-to-memory with low 8 bits of address. These numbers must be part of the executing program.

3) Perform fetch from memory at the location where pointer-to-memory is set and put data into accumulator.

The Intel 8080 CPU also has four 8-bit-wide registers that can temporarily store data. These can operate independently as R1, R2, R3, and R4. A limited set of operations treat these as 16-bit-wide register pairs, (R1, R2) and (R3, R4). Zero, negative, and carry flags are provided to ascertain the result of arithmetic and logic operations that occur in the accumulator. As in most microprocessors, these consist of addition, subtraction, increment, decrement, comparison, and the simpler logic operations "and," "or," and "exclusive or."

The last feature of the hardware shown, the stack pointer, is associated with the fact that most operations in the real world are discontinuous. In daily life, as you are interrupted by a visitor while filling out a form, the interrupted task will be put aside in a mental stack where it can be picked up at a later time. While you are taking care of the visitor the telephone rings, and the visitor's problem is also placed in the stack until you hang up the telephone. Then you can go back to the visitor's problem at the point where you left him, and when he leaves you can go back to the paper work.

A computer can be designed to operate in the same way, and the program counter is the key to the whole operation. A program can be interrupted in several ways. For example, if a software request is made to jump to an area of memory where you can calculate a commonly used function (such as $\log x$), and then return when done to the executing program, you jump to subroutine. Or, to service a request by an external device for information and then return when done to the executing program, you interrupt. Each time this occurs, if you can store the program counter and the flags, as well as the contents of the accumulator and all registers, in some sort of stack for retrieval at a later date, a satisfactory operation will be assured. After each such interruption the material stored on the stack will be popped off and the program counter, flags, accumulator, and registers restored to where they were at the time the interruption occurred.

If all of this material is stored in memory the length of the stack can be unlimited, and many nested interruptions can be serviced. The function of the stack pointer is to keep track of the last information placed on the stack. When the current interrupting task is serviced the stack pointer allows the program to restore the CPU for the next shelved task, and the stack pointer is moved to show the location of the information associated with any previously interrupted operations. It should be noted that the stack operates as a first-in, last-out device, in the sense that as many successive interruptions occur the first thing placed in the stack is the last thing removed.

Two other features of the Intel 8080 require comment in order to set the stage for the discussion of the other processors. Since the restrictions on memory access imposed by the use of a pointer-to-memory register are severe, a means of loading constants and other numbers into the registers quickly and conveniently is provided. Rather than having to load the pointer-to-memory register and then execute the fetch, the CPU has the ability to address memory relative to the current value in the program counter. The concept involves loading the register with the contents of memory pointed to by the program counter, which fetches the contents of the memory location immediately after the location in which the executing instruction is found. This particular form of program counter relative addressing is called immediate mode addressing.

The second feature involves the fact

that most computers manipulate numbers in purely binary fashion. The outside world often manipulates numbers in a decimal format, coded in binary, or BCD.

The Intel 8080 was originally designed for the point-of-sale market, and it does have the capability of taking two 8-bit numbers, each representing two BCD digits, adding them, and correcting the result to proper BCD notation. This is termed BCD-decimal adjust.

Application 1

A common automation task in the psychology laboratory is to provide a set of cue signals to a test animal, and reward correct responses with food. For example, a specific light, or a certain sequence of lights, being turned on should elicite a response that consists of depressing a particular microswitch. These functions are easily activated and sensed by an Intel 8080 processor. Stimulus and response data, at this level, require little mathematics. Equipped with an appropriate electromechanical interface, the microprocessor can produce the final data in a test on punched paper tape for submission to a larger computer for statistical analysis.

Very simple programs can calculate the digital numbers necessary to generate the cue signals, and these can be output by the processor. Most vendors now provide an interface board that will output such digital signals on command; for example: 1, light on; 0, light off (see latches in Fig. 2). Normally, all that needs to be added is driver circuitry to provide the voltages and currents necessary to activate the external devices. Input of information from microswitches involves digital information—Is it open or closed, 1 or 0?—and can be accomplished with vendor-supplied digital input interface boards (see gates in Fig. 2).

The fact that the stimulus and response pattern can be altered by changes in the software (operating program) rather than by hardware changes is an attractive one. Most laboratory scientists will be more attracted to learning the rudiments of programming than to acquiring expert knowledge in electronics. However, with microprocessors the programming language will be machine or assembly code.

At this point it is necessary to bring up the concept of analog and digital data. In the example above the input and output data are in digital form—all conditions can be represented by an appropriate series of 1's and 0's. The digital world comes in discrete steps, like integral numbers. Yet the world we live in is composed largely of signals in the analog domain which have an infinite number of allowed states; witness the positions allowed on the meter in most voltmeters. A computer can consume only digital data. For this reason there must be a mechanism to convert analog information, which originates in most of our analytical equipment, into digital data. This involves analog-to-digital converters (ADC) (Fig. 4). On the other hand, after a computer has digested the input data and evaluated them on the basis of programs it contains, it is often necessary for it to communicate with the outside world. If this outside world is an analog one (like oscilloscopes and plotters) it is necessary to convert a string of digital bits into an analog signal. Digital-to-analog converters (DAC) perform this task. These more complex automation techniques are used in the application discussed next.

Application 2

In a biological monitoring system, fish are held in flow-through tubes and are subjected to input streams with varying amounts of contamination (Fig. 5). The respiration rate and "cough" rate can be ascertained by monitoring the inhalation-exhalation cycle with a pressure transducer. The output of the transducer is converted to digital data by an ADC. Peaks, valleys, and intensities are detected by means of software. The frequency of peaks and valleys provides respiration rate data, and intensities above a

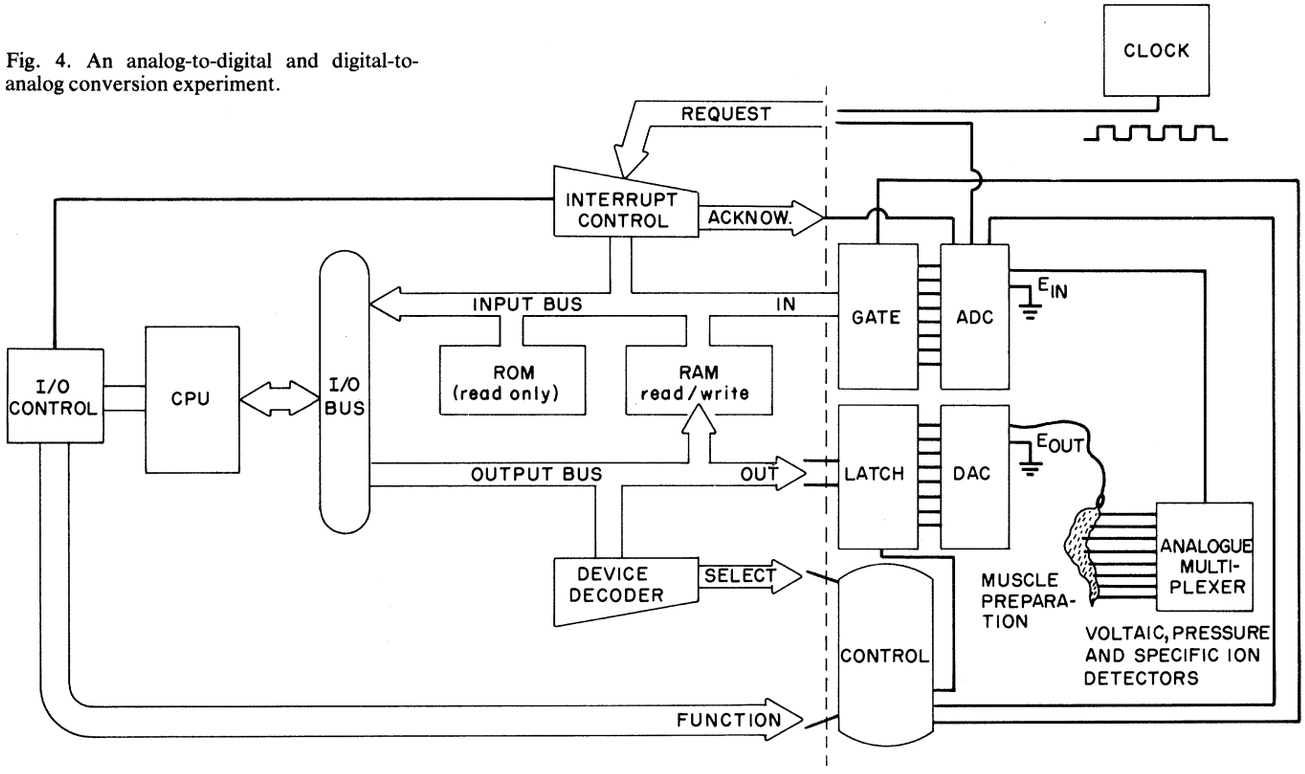

Fig. 4. An analog-to-digital and digital-to-analog conversion experiment.

certain value are associated with cough reflexes. These processes involve taking a derivative of incoming data to detect maximums and minimums. This is easily accomplished by taking the differences between two successive digital measurements and comparing the value with previous and future difference values. The air volumes involved are determined by integrating the incoming digital data, which requires summation of the data over the respiration cycle.

For the method to be economical and statistically valid, it is usually necessary to monitor many fish on a round-robin basis. For example, 15 fish might be involved in the study and it might be desired to sample each fish for 1 minute, every 15 minutes. The sampling rate during one measuring cycle has to be 120 points per second. A real-time clock (one ticking independently of the computer) can interrupt the computer at line frequencies (60 or 120 hertz) and the computer can use this information to continuously update multiple registers so that it can keep track of time (see Fig. 4).

Analog-to-digital converters are available which can digitize the data in 10 to 100 μsec, and the data may then be input by means of digital input interface modules. Because of the mathematics and data storage required, it is desirable to have a machine with multiple accumulators and the ability to store data easily in arrays (one array per fish) giving respiration and cough data as a function of time. Reports would consist of typed forms produced every hour showing the time, fish number, respiration rate, and cough rate for the four quarter-hour periods, and the average.

The Motorola M6800 (Fig. 6) is a microprocessor that has two accumulators, a stack pointer, and the necessary program counter and flags. In addition to the standard zero, negative, and carry flags, it has an overflow flag. Most important, it has an index register, which makes array handling simple. In handling an array it is necessary to be able to use the base address (starting address of the array) and a displacement (how far down the array the desired element is) to calculate the absolute address (exact location in memory) of the element needed. An index register loaded with the base address is used for this purpose as follows (small capital letters indicate instructions given by the operator).

FIRST: CALCULATE RELATIVE LOCATION ELEMENT (DISPLACEMENT)

THEN: FETCH FROM MEMORY BY CALCULATING [(DISPLACEMENT) + (VALUE IN INDEX REGISTER)].

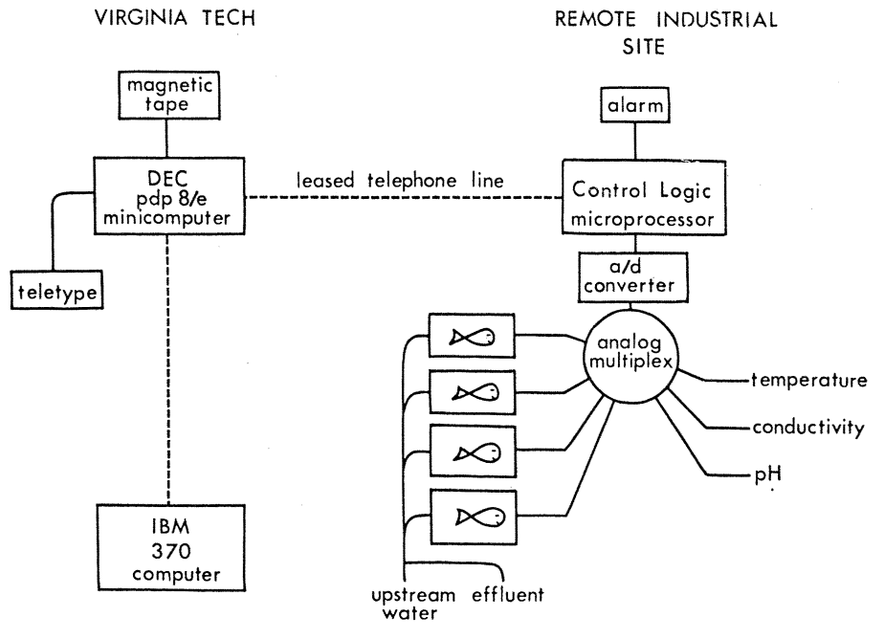

Fig. 5. A pollution monitoring system. [Courtesy of J. Cairns, Center for Environmental Studies, Virginia Polytechnic Institute and State University]

The value in the index register is not changed, and the CPU has another register, invisible or transparent to the user, in which the summed absolute address is entered to perform the correct fetch. It works just like the pointer-to-memory in the Intel 8080; indeed, the normal mode of addressing on the 8080 is index addressing with zero displacement. The Motorola M6800 can accept displacements of up to 128 places. It can also handle program counter relative addressing in a very sophisticated manner in comparison to the Intel 8080.

Mathematics

Most people employ a computer to perform calculations rapidly and accurately. Yet how the computer handles a mathematical operation determines the time required to perform it. This is not understood by the average user. The fact that an 8-bit-wide data word will only accommodate a range of integral values from 0 to 255 has been mentioned. It is possible in 8-bit processors to handle numbers in register pairs, and to think of data words 16 bits wide, accommodating a span from 0 to 65,536 (64 kilobytes). However, since only the accumulator register can perform mathematics in the CPU's mentioned, adding two 16-bit numbers (double precision in this machine) requires considerable juggling of the 8-bit bytes comprising each 16-bit word. If the numbers are in memory, as is usually the case, the number of operations required make this an extremely slow process.

An alternate notation, which will increase the range of numbers that can be handled, is like the scientific or floating notation commonly used. A number can be stored in 2 bytes (16 bits) in the following manner.

$$\underset{---}{\pm}\ \underset{---}{\text{exp}}\quad \underset{----------}{\text{fraction}} = \text{number} =$$

$$\pm\ .nnnnnnnnnn \times 2^{\pm nnnn}$$

The range of permissible numbers can extend from

$$.1777_8 \times 2^{17_8} (\sim 32{,}000)$$

to

$$.1000_8 \times 2^{-20_8} (\sim .00001)$$

a dynamic range of about 3×10^9.

The ability to handle larger ranges and fractional numbers has been achieved. The price has been paid in lower resolution—only 10 bits for the fractional representation in floating point (1/1,000) but 16 bits in the classical integer representation (1/64,000). To increase the precision, more bits are needed in the representation of the number. Typical compromises involve 8 bits of exponent and 16 bits of fraction, including a bit to represent sign (1 = −; 0 = +).

In an 8-bit machine the handling of floating numbers is time- and space-consuming. The program (software) necessary to support conversions from integer format to floating point format, to support the mathematical functions of addition, subtraction, multiplication, and division, or to support any extended func-

tions (logarithm, sine, and so forth) is long. Typically, 1500 bytes for four-function arithmetic and 3000 bytes for simple extended functions are required.

The scientist should be aware that although he can integrate, take logarithms, and so forth directly, the computer must perform all transcendental functions by approximation methods. In calculators this is done in hardware, and the times involved are often hundreds of milliseconds. In computers with minimal configuration this must be done by software. Typical times for minicomputer systems are 1 to 10 msec. For microprocessors this time must be multiplied by a factor of 2 to 10. Both the execution time of a code and the cost of writing that code must be carefully evaluated.

It should be noted that the four processors discussed in this article, if purchased as operating or near-operating systems, would each cost \$1600 ± 20 percent in a configuration with 8 kilobytes of memory, terminal interface (serial I/O), and parallel interface (digital I/O). Price considerations might encourage a user intending to build 500 units to employ a less expensive CPU, despite the extra software costs, since he can write off program development costs over the entire production run. A different set of financial factors is involved when only one or two automated units are being considered. Anything that will reduce software preparation costs is vital. It is for this reason that a careful analysis of your problem is important in matching CPU specifications to your needs. With this in mind, a more complex problem is examined next.

Application 3

The typical amino acid analzyer is a system ideally suited for automation by microprocessors. In operation a number of valves must be controlled by the processor to change the eluting buffer in both concentration and composition. Usually absorbance at several wavelengths is used to detect the eluting peaks. As the data are received by the computer they must be processed to reject information below a selected threshold value. When a peak is detected, rather sophisticated mathematical operations must be applied in real time. A running weighted digital filter is applied to the data to reduce noise. First-derivative techniques are employed to detect maximums and widths at half-height. Second-derivative techniques are often used to detect and help characterize shoulders.

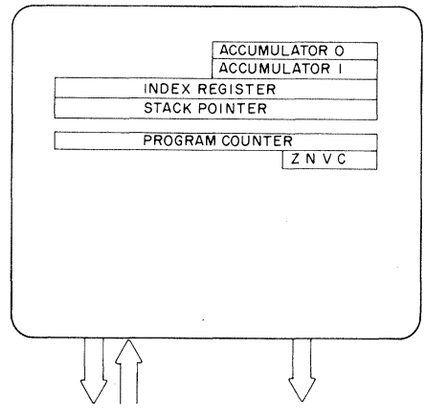

Fig. 6. Architecture of the Motorola M6800 CPU.

Because of ammonia bleed it is often necessary to accommodate baselines that drift and to subtract out plateaus.

In addition, there is no point in automating unless the unit is programmed to identify materials by the time window in which they elute, and calculate from calibration factors the concentration of each amino acid. This requires the system to be capable of accepting a command to calibrate, followed by a list of amino acids in the standard and their known concentrations. The amino acids are entered in the order in which they will be eluted from the column. As the sample is analyzed, time windows are calculated and response factors determined from the known concentrations and observed absorptivities. These constants are stored as arrays for use when an unknown is processed.

This requires a processor with good memory addressing capabilities, multiple accumulators, and a 16-bit data word to accommodate the mathematical operations required. Arithmetic capabilities

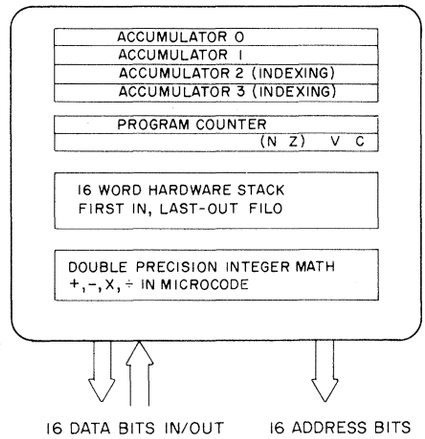

Fig. 7. Architecture of the National Semiconductor IMP-16 CPU.

available in hardware will be helpful.

The IMP-16 series (Fig. 7) of the National Semiconductor Corp. has four accumulators. Two of these (AC2 and AC3) may be used as index registers, allowing "simultaneous" access to two arrays. The remaining accumulators can be used for the typical mathematical operations.

The IMP-16 is a microcoded CPU. This means that a special decoding element capable of handling very long "command" words is present. Being longer, these commands can elicit quite complex operations with a single instruction. It is possible with microcode to provide the IMP-16 with the capability of implementing special instructions to perform double precision (32 bits on this machine) integer arithmetic (addition, subtraction, multiplication, and division). The two 16-bit numbers involved are placed in the AC0 and AC1 registers, and the microcode executed. The results are left in AC0 and AC1. This speeds up the mathematical operations considerably in comparison to software integral arithmetic. Numbers to 4.3×10^9 are accommodated.

The IMP-16 has the ability to address memory relative to the program counter. This can be viewed simply as another operation in which a displacement with reference to a base address is used to calculate memory address, but where the program counter provides the base address. This is a logical extension of the immediate addressing scheme described for the Intel 8080, where only zero displacements were allowed. However, in the IMP-16 addresses ±128 bytes from the program counter can be accessed. All of these schemes are attempts to provide methods by which fewer bits and less time are needed to access memory in a program.

The IMP-16 also has indirect addressing. Instead of a direct, relative, or indexed operation giving the address in memory required (pointer-to-memory), the indirect operation gives the address of the address in memory required (pointer-to-a-pointer-to-memory). This feature is valuable because the stack in the IMP-16 is a hardware stack limited to 16 words. This makes access to materials on the stack rapid, compared to memory fetches, but one cannot build long arrays in the stack without data "falling through" (more than 16 words put in the stack) and being lost.

The indirect method allows a pointer to be set up in a memory location and then used to create an array in another part of memory. Unfortunately it is the

program's task to continuously increment or decrement this pointer location as the file is created. This resembles the operations involved in the Intel 8080, where it was the user's job to update the pointer-to-memory register in creating a list.

Application 4

Since the lifetime of perfused biological samples is short, it is usually necessary to abstract as much information as possible from the experiment in a short time. Typical experiments involve nerve or muscle preparations in which a variety of transducers are placed in a large number of places (Fig. 4). Galvanic activation of the preparation, controlled in intensity by the computer, can lead to a response studied by simultaneous measurement of the rate of propagation of the electrical impulse in the nerve and the response at the activated tissue.

Electrical, specific chemical ion, and pressure transducers will be needed to obtain data. These data can measure (i) rates of electrical propagation of the stimuli, (ii) propagation delay areas, (iii) ion recovery rates, and (iv) response time in the tissue. Predictions of fatiguing rate must be made, and closed-loop control of the experiment may be desirable.

Sampling of data in such a system requires a processor with extremely well developed memory addressing, since multiple arrays will be created and used. Mathematical operations are quite sophisticated and require real-time execution with a fair degree of resolution. It would probably be desirable to be able to replay an experiment in slow motion on a video terminal. Heretofore these operations have been the perogative of a large minicomputer, but microprocessors are now available with the capabilities of handling the problem.

The LSI-11 (Fig. 8), manufactured by Digital Equipment Corp., is a microprocessor chip set that emulates the operations (instruction set) of Digital Equipment's PDP 11/40 computer. It is thus a CPU that falls in the gray area between microcomputers and minicomputers. With six accumulators, any and all of which can be used as index registers or stack pointers, the LSI-11 has remarkable power. It has all the forms of addressing previously described: direct, indirect, index, and relative. In addition, it has autoincrement and autodecrement features when indirect addressing is used. In this mode the CPU automatically steps the pointer forward (increment) or backward (decrement) through a sequential operation. This removes the need for the program to update pointer-to-memory information as contiguous files are being created. It also has microcoded integer multiplication and division as well as two-word floating arithmetic addition, subtractions, multiplication, and division. The latter mode will accommodate a range of $\pm 10^{\pm 38}$. Typical time for floating division is 150 microseconds.

Table 2. Memory and time needed to add two 16-bit numbers.

Microprocessor	Memory (bits)	Time (μsec)
Intel 8080	120	46
Motorola M6800	104	35
National Semiconductor IMP-16	80	28
Digital Equipment Corp. LSI-11	48	11.7

Discussion

It is appropriate at this point to sum up the capabilities of the various processors by comparing the relative amount of memory storage (for program) and time that would be needed to execute the addition of two 16-bit words (see Table 2).

For 32-bit integer multiplication or division the IMP-16 will take 2.7 times more time than the LSI-11. Although equivalent benchmarks are hard to find, a 3-byte software floating addition takes 2.5 msec on the 8080, compared to 50 μsec for a 4-byte firmware floating addition on the LSI-11—a factor of 50. Before attempting to use this information alone to focus on the processor of choice, let us look at software.

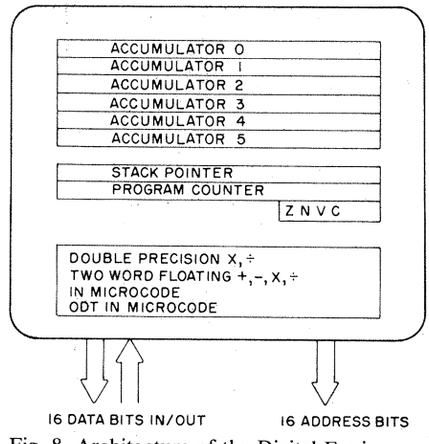

Fig. 8. Architecture of the Digital Equipment Corp. LSI-11 CPU.

The generation of software for microprocessors is, and will continue to be, the major problem. The CPU can accept code only in a language made up of 1's and 0's—machine code. Humans "speak" arithmetic, algebra, and words, but not 1's and 0's. It is possible to "hand code" from an instruction card in machine code. In the simpler processors like the Intel 8080, it is possible to code programs of less than 500 bytes in this way—if the user has had considerable experience. As the programs become longer, mistakes in address calculation and the need to insert forgotten code become insufferable problems. In more complex machines, particularly where displacement values must be calculated, the frustration point comes even earlier.

It is possible to operate in a mnemonic language, with abbreviated symbols to assist humans to remember instruction codes. In this method the user writes source code (a language from which another language is to be derived), using an editor which allows for corrections, insertions, and deletions. This source code can be stored temporarily on paper tape, cassette, or high-speed rotating memory such as floppy disks. The latter is the only really satisfactory solution because of problems of handling, speed, and error rate in the other devices. The source code is then submitted to another program called an assembler, which does all the address calculations and converts the words into instructions that can be handled by the machine. Unfortunately, adding the software to support the editor, assembler, and disk storage and retrieval to the bulk storage hardware increases the cost of a microprocessor to a point where it is equal to or greater than that of some minicomputers. Where large numbers of units will be manufactured in-house or for resale this can still be economical, but for the one-of-a-kind developer it is a disaster. It would be better to start off with a well-known minicomputer which has a large amount of available software.

Most laboratories are using cross-assemblers (assemblers running on one machine that produce machine code for another) which run on larger minicomputers or through time-sharing facilities. These are available for most of the large-volume microprocessors; however, a careful study should be made before assuming that a cross-assembler is indeed available for your processor, and that it is "bug-free."

Finally, several companies are beginning to offer higher-level languages such as Basic or a subset of PL-1. The ability

to handle real-time functions is lacking in the former (just as real-time Basic lacks many I/O features). Both produce code that is inefficient in its use of memory and therefore costs time and money to execute. This will not be a serious factor for the one-of-a-kind developer. He should not be stressing the CPU, and a few hundred dollars in memory will easily offset the price of coding, which costs about $10 per line. At this point in microprocessor development high-level languages are not universally available or properly developed, and they may not be available for the CPU that meets the hardware and software characteristics needed for your application. At least three vendors are making microprocessors that use the same machine code as an existing minicomputer. Thus, software at any level can be developed on a host minicomputer and "downloaded" without code conversion. This is one elegant feature of the LSI-11 which is a mini-based microprocessor.

It is vital that the prospective microprocessor user be aware that hardware and software documentation will be far less extensive than similar material provided with minicomputers. Microprocessor vendors assume a certain level of sophistication in their end users and provide accordingly. Microprocessor hardware may be purchased in the form of CPU chip sets, but this is not recommended for the average scientist. Nor is purchasing the CPU card that contains most of the decoding and I/O control elements. A complete system with some software support is most practical. For the novice, or even those with considerable experience with high-level languages, it is essential that the machine have a bootstrap (the program that loads subsequent programs) and a rather sophisticated debug program. The latter permits the user to examine and alter the content of memory addresses in machine code. Programs can be initiated and in some cases run in single-step mode or with set breakpoints. These are the only feasible ways to find errors. Diagnostic software that detects and reports hardware malfunctions is essential, since few microprocessors sold have field or factory repair service.

Although high-level executives and operating systems (a collection of programs that aid in programming) are advertised, the state of the art is still primitive. Since much of the information published in this area for minicomputers is often misleading, it is doubly important to establish the reliability of software claims in the microprocessor area.

The best way to judge your chances for success in the plunge into microprocessors is to see if the prospective vendor or you can understand and answer the following questions about the system of interest to you.

1) Can the firmware bootstrap and debugging program accept terminals with mark, space, odd, or even parity?

2) Does the debugging aid permit the saving of all registers and flags during execution of breakpoints?

3) Are the binary-formatted tapes provided in absolute format, or is a linking loader needed? If one is provided, how much memory is required to run it?

4) Is there a software interrupt disable command? If the interrupt priority scheduling is in hardware, does it arbitrate correctly? If it is in software, does masking-off capability exist in the interface hardware?

5) What is the resolution of the floating point package? (Some 24-bit floating point routines only give five-digit resolution, compared to eight digits on handheld calculators.)

6) If an operating system is supported, are the device handlers written so that they may be added and deleted easily to reconfigure the system—or are they written in in-line code, making alteration almost impossible?

7) If high-level languages are supported, how long a user program can be run, and how many user-assigned symbols are permitted in the normal memory configuration?

8) What is the bit drop-rate of the peripherals provided?

9) Does the vendor provide ROM "burning" facilities so that programs may be loaded into nonvolatile memory for long-term use?

10) Who is responsible for issuing refresh commands in memories that are dynamic? And does this process interfere with the operation of other peripherals?

All of these above questions have been constructed from actual case histories in microprocessor applications. The problems involved were overcome, but at considerable cost in time and money. If the vendor or you cannot respond to these questions, you are not ready to use microprocessors. Microprocessors offer an inexpensive solution to the hardware problems encountered in laboratories, but they may engender a manpower expenditure in programming and hardware setup that is prohibitive.

Developments in this area have been breathtaking in the last 3 years, but a technological plateau has been reached that I believe will be maintained for several years. The main thrust during this period will be toward lower cost rather than an increase in power that is evident to the end user. One can then expect the introduction of new capabilities in hardware, mathematics and statistics packages, memory methodology, and substitutes for rotating bulk-storage devices. Large-scale integrated circuit chips will ease the interfacing task by providing flexible, inexpensive I/O control, but they will usually be specific to particular CPU's and peripherals. A supermarket or erector set philosophy in microcomputer system design is still a long way off. Some think it will never come.

By this point, it is hoped that the reader will have enough understanding to analyze the potential application of microprocessors to his tasks. In processes where mathematics and memory accessing are minimal and the programs are of a length suitable for hand-coding, the Intel 8080, with its simple-to-understand instruction set, is ideal. Where speed and language sophistication are important, CPU's like the Motorola M6800 are applicable. Where speed, mathematical power, and instruction set versatility are important and expansion is envisaged, look to mini-based microcomputers like the LSI-11.

It is strongly urged that your first introduction to computers not be microprocessors. The learning curve is a steep one that is best approached by some experience on gentler slopes and some previous exposure to such factors as the cost of program development, problems with the mathematics package, and running out of real time is desirable.

Notes

1. Suggested further reading: A. Osborne, *An Introduction to Microcomputers* (Adam Osborne and Assoc., Inc., Berkeley, Calif., 1975); R. H. Cushman "Second annual microprocessor directory," *EDN* **20**, 42 (20 November 1975); R. E. Dessy, P. J. Van Vuuren, J. A. Titus, *Anal. Chem.* **46**, 917A (1974); *ibid.*, p. 1055A.
2. This article has been developed with the aid of E. Fiorino, J. Fiorino, P. Hallman, J. Leone, I. Starling, D. Binkley, D. Hooley, C. Knipe, W. Nunn, D. Shearer, M. Starling, and H. Wohltjen. These people compose the Chemistry Department's Instrument Design and Automation Research Group.