UIC John Marshall Journal of Information Technology & Privacy Law

Volume 6
Issue 2 *Computer/Law Journal - Fall 1985*

Article 1

Fall 1985

Legal Protection for Microcode and Beyond: A Discussion of the Applicability of the Semiconductor Chip Protection Act and the Copyright Laws to Microcode, 6 Computer L.J. 187 (1985)

John R. Harris

LEGAL PROTECTION FOR MICROCODE AND BEYOND: A DISCUSSION OF THE APPLICABILITY OF THE SEMICONDUCTOR CHIP PROTECTION ACT AND THE COPYRIGHT LAWS TO MICROCODE†

by JOHN R. HARRIS*

The question of whether the present copyright laws[1] afford protection for microprogrammed circuits or "microcode," especially after *Apple Computer, Inc. v. Franklin Computer Corp.*,[2] has already been addressed.[3] With the passage of the Semiconductor Chip Protection Act

---

* John R. Harris specializes in electronics and computer-related intellectual property law with the patent, trademark, and copyright law firm of Jones & Askew in Atlanta, Georgia. After receiving a bachelor's degree in electrical engineering with honors from Georgia Institute of Technology in 1973, he worked as a special purpose computer systems design engineer for several years. He received his J.D. and M.B.A. degrees from Emory University in 1979. He has co-chaired the Southeastern Computer Law Institute, held in Atlanta, for two years. This article was first presented at the Second Southeastern Computer Law Institute in May 1985.

1. 17 U.S.C. §§ 101-118 (1982).

2. 714 F.2d 1240 (3d Cir. 1983).

3. *See* Harris, Apple Computer, Inc. v. Franklin Computer Corp.—*Does a ROM a Computer Program Make?*, 24 JURIMETRICS J. 248 (1984).

of 1984 (SCPA),[4] a new intellectual property protection scheme is available for manufacturers of computer-related and other electronic equipment providing more protection than the previous copyright and patent laws. Even this new law, however, fails to provide adequate protection for the microcode. This situation has recently been brought to the forefront of computer-related legal issues by NEC Corporation, with its effort to obtain a declaratory judgment that its new V20 series microcomputer chips do not infringe any copyrights in Intel Corporation's 8086/8088 microcomputers.[5]

This Article will examine several related questions. Is microcode, or a microprogram, protected under any existing intellectual property laws? Does the SCPA apply to microcode? If microcode is protected under the copyright laws, how should the question of "substantial similarity" be resolved in determining infringement?[6] Why do the patent laws not protect microcode, given that the function of microcode is to control the operation of a computer at the circuit level?

This Article presupposes a general familiarity with computers, copyright law, and patent law, but not with microcode. To appreciate the subleties regarding the protection of microcode, it is necessary to have some understanding of what microcode is and why it is crucial to the computer industry. Accordingly, this article will first address the technical subject matter of "microcode" and "microprogramming" for intellectual property protection purposes, and will then examine the applicability of the patent laws, copyright laws, and the SCPA for protecting microcode.

As will be shown, none of the existing intellectual property law protection schemes are adequate to protect microcode, and one important question is whether the instruction sets for computers should be afforded protection. One result of protecting computer instruction sets would be to provide computer manufacturers with exclusivity in the manufacture of computer hardware embodying the architectures they have originated. Whether this is desirable from a policy standpoint is beyond the scope of this Article, but is nonetheless an important consideration.

---

4. Pub. L. No. 98-620, sec. 302, 1984 U.S. CODE CONG. & AD. NEWS (98 Stat.) 3335, 3347 (codified at 17 U.S.C. §§ 901-914).

5. NEC Corporation and NEC Electronics, Inc. v. Intel Corp., No. C84-20799 WAI (N.D. Cal. filed Feb. 14, 1985) [hereinafter cited as the NEC Complaint].

6. A basic element of copyright infringement is that there must be substantial similarity between a plaintiff/copyright owner's work and a defendant/alleged infringer's work. *See* M. NIMMER, NIMMER ON COPYRIGHT § 13.03 (1985).

I. WHAT IS MICROCODE?

Many present day computers use microcode to coordinate the resources of the central processing unit (CPU) of a computer system.[7] A set of controlling "microinstructions" comprise a "microprogram," controlling the sequencing of the various CPU resources. Typically, each microinstruction supplies a parallel combination of 1's and 0's to control these resources and operates independently of the data flow of the CPU. Because terminology may be crucial in determining whether and how protection may be obtained, it is necessary to provide some definitions of computer terms and some background in computer architecture.

In this Article, the terms "microcomputer" and "microprocessor" will be used interchangeably; however, there are some who maintain that a microprocessor is a CPU on a single integrated circuit chip, while a microcomputer is any computer system which includes a microprocessor.[8] It is extremely important to understand that a microcomputer or a microprocessor does not necessarily contain any microcode or microprogramming. The presence of microprogramming is strictly a function of architecture (i.e., the design of the physical structure of the computer), and not of nomenclature or labels.

In contrast, minicomputers are generally considered to be small computer systems which have more processing power than a typical microprocessor chip. Many of today's minicomputers, however, employ microprocessors as their main CPU, in addition to various supporting peripheral circuitry, giving the system greater input/output power and flexibility. In further contrast, a mainframe computer is large and has few "mini-" or "micro-" aspects.

Confusion in the use of these and other computer terms is inevitable, since the terminology is continually evolving and is being created by the people who are creating the technology and not by grammatically-oriented linguists.[9] Definitions in any rapidly evolving subject

---

7. "Resources" are computer circuits that function separately in carrying out the functions of the CPU. The function of the CPU in most von Neumann-type computer systems is to execute programs stored in the main memory by fetching the instructions, examining or "decoding" the instructions to ascertain what to do, and then executing the instruction. Typically, the CPU comprises circuitry for performing a memory access or "fetch"; an instruction decoder; several small, local one-byte or one-word memories ("registers") for holding the temporary results of calculations and addresses in the main memory of the next instruction and the data to be operated on; and arithmetic logic units (ALUs) for performing mathematical operations on the data. These various components in the CPU are typically known as resources.

8. *See* A. TANENBAUM, STRUCTURED COMPUTER ORGANIZATION 25-26 (1984).

9. "If you ask n different computer scientists to define 'microprocessor' or 'microcomputer' you are likely to get n different answers." *Id.* at 25.

matter, such as computer technology, may change or become obsolete within a short period of time. Thus, it is not possible (or desirable) to place hard and fast definitions on "microcomputer," "microprocessor," "minicomputer," or "mainframe"; technical experts recognize the dangers of this and caution against it.[10]

When different computing entities first appeared, there were definite technical differences between them, facilitating the assignment of names and definitions. For example, early mainframes invariably had word lengths of thirty-two bits or more; when microcomputers first appeared, they usually had four bits. Mainframes could access extremely large memories (in the million-word or "megaword" range), while micro's could only access relatively small memories, usually less than a million bytes[11] or "megabytes." Today, microcomputers are employed in constructing minicomputers. Microcomputers can control vast amounts of memory and can execute the instruction sets of some mainframe computers. The IBM Personal Computer XT/370, for example, supposedly can execute the IBM System 370 mainframe's instruction set, therefore running what was once considered "mainframe only" programs.

A detailed explanation as to what is meant by "microcode" and "microprogramming" is necessary. To explain these concepts adequately, it must be noted that most modern computers are multi-level in structure, as illustrated in Fig. 1. In this illustration, Level 1 is the boundary between the digital logic level—the domain of the electronic circuitry of the computer—and Level 2, the "conventional machine level"—the beginnings of computer programs as most commonly understood. The levels above Level 2 are probably "computer programs" for purposes of section 101 of the Copyright Act.[12] Also, Level 0 is probably only entitled to patent protection, since only patent and not copyright laws protect physical functioning structures.[13] As is often the case when applying the law to a set of facts for the first time (or to facts

10. "[N]o conceptual difference exists between mainframes, minicomputers, and microcomputers. They are simply rough names for various overlapping parts of a continuous spectrum of processor power. Furthermore, future developments in technology are likely to make these names even less meaningful than they are already." *Id.* at 26.

11. The terms "word" and "byte" are also somewhat arbitrary. A "word" can be any number of bits, depending on what the manufacturer wants to call the basic number of bits which can be operated on with a single instruction. A word today typically means sixteen bits; a long word may be thirty-two bits. A "byte," on the other hand, is just what it sounds like: take a "bite" of a longer word such as sixteen bits and you get an eight-bit "byte."

12. A "computer program" is a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result. 17 U.S.C. § 101 (1982).

13. *See* Brown Instrument Co. v. Warner, 161 F.2d 910 (D.C. Cir.), *cert. denied*, 332 U.S. 801 (1947). In that case, the Court said:

which the drafters of the law did not consider at the time), it is difficult to draw a precise line at the microprogramming level and state unequivocably when a particular set of laws should apply instead of another.

The boundary between hardware and software is not well defined, nor can it be, so artificial legal distinctions which rigidly state that "hardware is patentable only" or "software is copyrightable only" are doomed to failure. As the levels of computer hierarchy in Fig. 1 illustrate, there must be a bottom level which consists of a machine with circuits, power supplies, disk drives, and other "hard" objects.

Levels higher than the pure hardware level can be, but not necessarily are, "software." It is theoretically possible to build a computer which directly interprets higher level programming languages such as FORTRAN by designing the circuitry to interpret the statements of this language and execute them directly in hardware, without using a compiler such as at Level 5 in Fig. 1. Hardware and software are logically equivalent;[14] any operation which can be done in software can be done in hardware, and any operation which can be done in hardware can also be done in software. There are, however, certain tradeoffs such as speed, expense, reliability, and ease of modification.

Another important microprogramming concept to be explained is that of a computer's "instruction set." This is the set of all computer instructions available to a programmer at a particular level (of Fig. 1). This means that there will be different instruction sets for the application language level and for the machine language level. The machine language level is the lowest level at which a program can be entered into a computer from outside the computer. The documentation which accompanies a computer system will detail the machine language of the computer. This means that this is the ultimate form that a program of instructions to the computer takes to enable the computer to function. Thus, a computer's "instruction set" may be defined as a predetermined group of externally-originating commands, each having a predetermined format, which cause a known result. Any commands taking a form different from those in the instruction set will not have any effect on the computer, though wrong instructions have been known to produce disastrous effects.

> Articles intended for practical use in cooperation with a machine are not copyrightable [citation omitted]. Both law and policy forbid monopolizing a machine except within the comparatively narrow limits of the patent system. . . . Since the machines which cooperate with the charts in suit are useless without them, to copyright the charts would in effect continue appellant's monopoly of its machines beyond the time authorized by the patent law.

Id. at 911.

14. A. TANENBAUM, *supra* note 8, at 11.

The number of instructions in an instruction set varies from level to level and from machine to machine. Typically, instruction sets range from about twenty to about 300 instructions. The instruction set and the microprogramming level are closely related. The microprogram in fact determines the instruction set and can only operate with a given set of CPU resources, since the only purpose of a microprogram is to control a particular CPU's resources in sequences necessary to carry out the instructions in the instruction set. The instruction set, therefore, is determined by the microprogram and not by what particular resources may be present.

A microprogram, consisting of subinstructions or microinstructions, causes the interpretation and effectuation of instructions in the instruction set. The microprogram is usually invisible to a programmer interested only in the outwardly-appearing functions of the machine. Most programmers do not care what occurs within the machine; they simply want their programs to be carried out in the manner they intended. This is not to say, however, that programmers are not concerned with "fine tuning" the operation of a computer; indeed, many programmers are keenly interested in the internal handling of each instruction and attempt to optimize execution speed or storage space of a program. Rather, the key point is that most programmers write programs in a language level above and including Level 2, since it is extremely tedious and time consuming to do otherwise.

Microprogramming is a technique used by hardware designers to implement the control functions of a computer.[15] As originally conceived, microprogramming was meant to be a specific technique "to provide a systematic approach and an orderly approach to designing the control section of any computing system."[16] Control in a computer refers to control of the available resources.

It should also be noted that some computers, especially older ones, do not have a microprogramming level. In these machines, instructions are interpreted with sequential, hard-wired logic circuitry without anything which could reasonably be called a set of subinstructions to control the actual timing and use of the computer resources. The choice between direct interpretation by circuitry and indirect interpretation by microprogramming is determined by many considerations. For instance, with regard to speed of execution, direct circuitry is faster than microprogramming. With regard to economics, direct circuitry is more expensive than microprogramming. Finally, with regard to flexibility, microprogramming can be changed more easily.

---

15. Flynn, *Microprogramming Concepts, reprinted in* P. FREEMAN, SOFTWARE SYSTEMS PRINCIPLES—A SURVEY 70 (1975).

16. M. Wilkes, The Best Way to Design an Automatic Calculation Machine 16 (1951).

II. AN EXEMPLARY MICROPROGRAMMED COMPUTER

The next step in examining microcode is to discuss an exemplary, but hypothetical, microprogrammed computer architecture. For the sake of simplicity, assume that our hypothetical computer has only two instructions in its instruction set: ADD X, Y and MOV ACC, Z. The ADD instruction causes the computer to add two numbers—X and Y. The number X is contained in a memory address following the instruction, while the number Y is contained in a memory address following the number X (i.e., the two numbers to be added are embedded in the program at the two consecutive addresses following the instruction). The ADD X, Y instruction leaves the result of the addition in the accumulator (ACC), a temporary data storage register resource. The MOV instruction "moves" data from the accumulator to the address Z in the memory address immediately following the instruction.

A few preliminary terms must be defined before the exemplary CPU can be described. As shown above, computers include a number of resources that allow data to be fetched, manipulated, and so forth. These resources must be coordinated in order to execute the computer's instructions. Events such as fetching or adding data must occur in a proper sequence of steps, as illustrated in Fig. 3.

The resources of a computer CPU are connected by a group of parallel wires known as a "bus." Typically, there is an "address bus" (for carrying the main memory address information), a "data bus" (for carrying the data to be operated on), and a "control bus" (for carrying the signals which control the various resources). Examples of control lines include signals to READ and WRITE the main memory, that is, signals to fetch data from or store data in the memory at an address specified on the address bus.

Other resources found in this exemplary computer include combinational circuits known as "multiplexers" (also known as "data selectors") and "decoders." A combinational circuit has one or more inputs and provides outputs as a predetermined combination of the inputs on a nontemporal basis. A multiplexer, illustrated in Fig. 4, has inputs A and B (which can be multiple lines of a bus), a data output O the same width as the inputs, and a control line which selects either A or B to appear on the output. A decoder, illustrated in Fig. 5, on the other hand, has n input lines, and 2^n outputs. If the input is viewed as a binary number, the decoder circuit provides an output only on the one line which corresponds to the binary number input. As shown in Fig. 5, there are four input lines, and 2^4 equals sixteen, so each of the sixteen possible combinations of 1's and 0's in a four-bit binary number will cause only one of the outputs to be active. Decoders are especially important in microprogramming, because they allow the "packing" and

"unpacking" of data. These functions allow a microprogram to become more "vertical," and thus more "computer program-like," as opposed to "horizontal," which is more hardware-like. These concepts will be explained further as the exemplary architecture is examined.

The two additional resources that need to be introduced are the "memory address register" (MAR) and the "memory buffer register" (MBR). The MAR holds an address in main memory for the CPU to access. The MBR receives the data from the main memory and sends it to the CPU. The MBR also holds data from the CPU when data is to be transferred out to the main memory. Both the MAR and the MBR provide an external interface between the CPU and the outside world, particularly the outside world of the main data memory.

The CPU fetches data (or the next instruction) from memory by placing an address in the MAR and asserting the READ control signal. The memory responds by retrieving the data (or instruction) from the location specified by the address and sends it over the data bus where it is loaded into the MBR. Likewise, when the CPU must store data in memory, it places an address in the MAR, places the data in the MBR, and asserts the WRITE control signal. The memory responds by taking the data from the MBR and storing it at the specified address, which it obtains over the address bus from the MAR.

There is an implicit assumption that the CPU does not know the difference between data upon which it is to operate and instructions which it is to execute. Most computer architectures today are referred to as von Neumann-stored program architectures,[17] which means that both the programs to be executed and the data upon which the programs operate are stored in main memory (or provided from a source external to the CPU such as a keyboard or other input/output device). The CPU is able to distinguish between programs and data by being given an address in the main memory where the program starts. The CPU then begins to perform the operations of the program by fetching the data stored at this starting address as its first instruction. The address of the instruction to be executed is usually stored in a program counter (PC), which automatically increases the address by 1 at the conclusion of each instruction. In the exemplary CPU, the PC provides the address of the program to be executed to memory through the MAR.

In order to carry out the instructions in the instruction set, the CPU determines what the instruction is through "instruction decoding," and then coordinates the resources in a sequence to carry out the instruction. After an instruction in a program (fetched from the address stored in the PC) has been fetched, the instruction's type is determined by an instruction register, also known as an instruction decoder.

17. *See* H. GOLDSTINE, THE COMPUTER FROM PASCAL TO VON NEUMANN (1972).

The fact that some programs of the CPU are external may be helpful in drawing the line between copyrightable programs and noncopyrightable hardware. As described in more detail below, however, even a microprogram can originate outside of the CPU, making it unwise to assign undue weight to the term "external." Any distinctions premised strictly on externality would probably fail.

The exemplary microprogrammed CPU illustrated in Fig. 6 shows that the CPU includes two registers as resources, REGISTER A and REGISTER B; an arithmetic logic unit (ALU), for performing mathmatical manipulations; an accumulator (ACC), for receiving the results of a manipulation performed by the ALU; an MAR; and an MBR. There is also an instruction register which receives data purporting to be an instruction and decodes such data by determining which of the two possible instructions has been provided in a program. An internal data bus routes data between the various resources. The internal data bus is "bidirectional," meaning that data can move in two directions along the bus (e.g., from the MBR to the ACC, or from the ACC to the MBR). The block labelled "control store" contains the microprogram.

In order for the CPU to execute the instructions ADD X, Y and MOV ACC, Z, the CPU must first fetch the instructions. It will be assumed that the address of the first instruction—ADD X, Y—is placed in the MAR by another mechanism.[18] Given our instruction set of two instructions, and given the list of resources at the disposal of the CPU, a sequence of operations can be constructed that must be carried out by the resources to execute the instructions. These two sequences are illustrated in Fig. 3.

A list of the possible commands to the available resources to be carried out by the microprogram can also be made, as shown in Fig. 7. Each of the possible resource commands has been assigned a separate number, so that the exemplary CPU has nine possible resource commands (which shall be called "microcommands"). This list is converted into a microinstruction in Fig. 8. In Fig. 8, a "0" means that the control indicated is not provided to the resource, while a "1" means that the control is to be provided.

Note that the microinstruction in Fig. 8 is labelled "Type A"; in Fig. 12, a different type of microinstruction will be introduced to illustrate how to make the microinstruction more vertical. What is being illus-

18. Most computers have a hard-wired, start-up address which is automatically placed in the MAR or its equivalent when the computer is turned on, reset, or otherwise initialized. It will be assumed that such a feature is present in the exemplary architecture. It will not, however, be described in detail, because it would divert discussion from the primary issues.

trated in Fig. 8 is that different microprograms, with different sequences of the same microinstructions, can execute different machine language instructions, thereby altering the instruction set. Likewise, a different sequence of different microinstructions can execute the same machine language instructions, giving the outward appearance to a higher level programmer that differently microprogrammed computers are functionally the same.

A set or series of microinstructions forms a "microprogram." It should be remembered that the execution of two machine level language instructions, ADD and MOV, are being examined. This execution occurs below the visible level of the programmer who writes a program in machine language, or who writes at a higher level and has the level of the program compiled or assembled downward to the machine language level. The distinction between a program and a microprogram, as have been defined here, must be borne in mind to avoid confusion. The program is provided to the CPU from the outside, while the microprogram is associated with the CPU resources, interprets the instructions, and coordinates selected CPU resources to carry out the instructions.

An issue that still needs to be discussed is timing. A clock is provided to allow computer designers to achieve their desired sequences or timing relations. A clock provides precisely timed electrical pulses which can be used to trigger events in the CPU sequentially, as illustrated in Fig. 2. Three other concepts that need to be introduced are machine cycles, clock cycles, and memory cycles. A machine cycle is the time it takes the CPU to execute one instruction. A clock cycle, on the other hand, is merely one complete pulse from the clock. A memory cycle is the time it takes for the memory to respond to a READ or WRITE command from the CPU. The machine cycle and memory cycle can be thought of as taking a particular number of clock cycles to complete. These relationships are illustrated in Fig. 2. The ADD instruction requires three memory cycles: one to fetch the instruction, one to fetch X, and one to fetch Y. The MOV instruction requires only two memory cycles: one to fetch the instruction and one to store Z in the memory.

A set of microinstructions (a microprogram) for fetching, interpreting, and carrying out the ADD instruction is shown in Fig. 9. The sequence in which the microinstructions are carried out is from the top on down, so that the first set of commands for the resources is at the top, the next one below it, and so on. Each microinstruction takes one clock cycle to complete, assuming that one clock cycle is a sufficient amount of time for the main memory to respond to an address from the MAR and place the requested data on the data bus to be loaded into the MBR.

Thus, the ADD X, Y instruction takes a total of eleven clock cycles to perform the machine cycle.

As can be seen in Fig. 9, the sequence of operations enables the various resources in turn to perform their basic tasks. For example, the first task is to load the MAR with the address of the first instruction and to assert the READ signal. On the next clock cycle, it is assumed that the memory has responded by placing the requested data on the data bus, so that the MBR can be commanded to load the data from the data bus. Once the data is in the MBR, it must be routed to the instruction register at Step 3. The instruction must be decoded, since at this point the CPU does not know which of the two possible instructions, ADD or MOV, it will encounter first. No resources are enabled, since it takes a finite amount of time to decode the instruction. After the decoding of the instruction, the CPU "knows" that an ADD instruction has been received; accordingly, the CPU instruction decoding circuitry causes the execution of the remaining eight steps of an ADD instruction.

If the instruction decoding circuit had found that another type of instruction was to be executed, a different set of microinstructions would be executed. In the exemplary CPU, the only other instruction possible is MOV ACC, Z. Accordingly, if the instruction decoding circuitry detected that the MOV instruction was to be executed, the microprogram that is illustrated in Fig. 9 would "jump" down to Step 16, which are the resource commands for the MOV instruction (note that Steps 12 through 15 are the same as Steps 1 through 4 and need not be repeated).

The portion of the exemplary CPU which contains the microprogram has various names, the principal one being "control store." The control store can take the form of a read-only memory (ROM) or of a read/write memory. The form of a control store is not important, except to illustrate that if a read/write memory is employed, the microprogram is relatively easy to change. By changing the microprogram, the functioning of the CPU will change by changing the response of the CPU to the machine language instructions of a higher level program. The control store of Fig. 9 takes a memory size of ten bits per word by seventeen words, for a total storage of 170 bits.

In order to use the individual words in the control store and to execute jumps between microinstructions as mentioned above, the concept of the "microinstruction register" (MIR) needs to be introduced. The MIR holds the next microinstruction to be executed. The MIR, however, may be considered as a sort of microresource. It needs its own controlling component to retrieve a microinstruction from the control store and to load the microinstruction into the MIR. A microprogram counter (MPC) is then used to point to the next microinstruction to be

executed in the control store, as illustrated in Fig. 10. Instruction decoding causes the MPC to jump to the appropriate address in the control store which was called for by the particular instruction decoded by the instruction register.

III. THE EFFECTS OF CHANGES TO A MICROPROGRAM

Any change to the microprogram of any computer alters the coordination of the various resources. Thus, the CPU may be microprogrammed to execute different machine level instructions or to execute the same instructions differently. In addition, the structure of the microprogram can be altered to accomplish different objectives. While the physical resource architecture may not change, a microprogrammed CPU can be made to appear completely different, with a different instruction set, merely with minor changes to the microcode. This should help illuminate the discussion of whether section 102(b) of the Copyright Act excludes microcode from copyright protection.[19]

Assume that the instruction ADD X, Y, instead of adding the numbers contained in the two addresses immediately following the instruction, adds the numbers contained in the addresses stored in the two addresses immediately following the instruction. In other words, the data which is added does not follow the instruction. Rather, the two addresses following the instruction contain the addresses of the data to be added. This means that in the exemplary microprogram, in order to execute such an instruction, the data following the instruction in memory should be loaded into the MAR as addresses for fetch operations.

A microprogram to accomplish this different type of ADD instruction is shown in Fig. 11. This type of ADD instruction may be referred to as an indirect instruction, because the instruction does not directly add the data immediately following the instruction in memory. Rather, the instruction causes the immediately following data to be used as addresses. The microprogram that accomplishes this type of "add" takes nineteen clock cycles, thus taking longer to execute. The bidirectional internal data bus allows the contents of the MAR to be temporarily stored in the accumulator so that the external program can continue executing in its program sequence. Thus, the data fetched at Step 5 eventually gets loaded into the MAR (at Step 8) to become the address for fetching X. Similarly, the data fetched at Step 12 gets loaded into the MAR (at Step 15) to become the address for fetching Y. Steps 7 and 14 include the process of enabling the contents of the MAR to appear on

19. In no case does copyright protection for an original work of authorship extend to any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is described, explained, illustrated, or embodied in such work. 17 U.S.C. § 102(b) (1982).

the internal data bus so that it can be temporarily saved in the accumulator.

The key point is that the physical resources of the CPU do not change; only the microprogram changes. The instruction ADD X, Y in Fig. 11 is a different instruction from that in Fig. 9. It should be understood, then, that a different microprogram can make a different computer, even though the hardware itself has not changed.

The structure of a microprogram can sometimes be altered to make it more vertical. A microinstruction having one control function per bit, such as "load the MAR," is considered to be horizontal. If the encoding of control bits is employed, the microprogram is said to have some vertical aspects. For example, note that in the Type A microinstruction of Fig. 8, there are seven resource registers which may be loaded with data. If each of these registers are assigned a number from 1 to 7, three bits can be used to select these registers, since $2^3 = 8$ (leaving an extra bit). Thus, one can eliminate a separate control line for each of the registers emanating directly from the control store, insert a three-bit decoder (such as in Fig. 5) to select a register, and thereby make the microinstruction narrower (six instead of ten bits). An illustration of this narrower, Type B microinstruction, with coded register select bits A, B, and C can be found in Fig. 12. A control circuit for this type of microinstruction is illustrated in Fig. 13.

Horizontal CPU designs usually have a relatively small number of wide microinstructions. Vertical designs, however, will usually have relatively many narrow microinstructions. In the exemplary CPU, the only real difference between the control circuit for Type A microinstructions and Type B microinstructions is that a slightly longer clock cycle may be required to allow sufficient time for the register select signals A, B, and C to pass through the decoder.

IV. NANOPROGRAMMING

A notion closely allied with that of microprogramming is "nanoprogramming," the generation of nanocode or nanoinstructions. The tendency towards miniaturization in electronics has fostered the need for labels to differentiate between levels of systems, which are arguably smaller than the preceding levels in some fashion, thus creating the need to introduce even more jargon.

Nanoprogramming is an extension of verticality in a microprogram. When certain microinstructions occur several times, it may be desirable, from a cost savings standpoint, to store the frequently occurring microinstruction in a control store only once, so as to enable it to be called forth when needed.

A close examination of the microprogram for ADD X, Y in Fig. 11

reveals that while there are nineteen different steps, there are only eight different microinstructions (actually, there are nine, but Step 4 will be dealt with separately). These eight different microinstructions can be placed in a nanostore (Fig. 14) to be accessed with a three-bit code. The microprogram would then consist of nineteen three-bit items for selecting the microinstructions in the nanostore, as illustrated in Fig. 15. The nanocoded microprogram of Fig. 15 corresponds to and implements the same instructions as the microprogram of Fig. 11.

The microprogram of Fig. 15 is executed, first, by fetching a three-bit code from the control store. The three-bit code is then used to select a nanostore word from Fig. 14, which is fetched and placed in the microinstruction register (MIR). The MIR is then used to control the various CPU resources as in the earlier examples. At the end of this step, the next three-bit code is fetched from the control store, and the process is repeated.

The microprogram of Fig. 11 requires a total storage of nineteen steps times ten bits per microinstruction, for a total control store of 190 bits. The required storage for the nanoprogrammed version of Figs. 14 and 15, however, is eight microinstructions times ten bits per microinstruction (eighty bits), plus nineteen steps times three bits select code per step (fifty-seven bits), for a total of 137 bits. Thus, fifty-three bits (190 - 137) of control storage have been saved. While these savings may not seem significant, it must be remembered that the exemplary CPU is extremely basic and simple. In addition, real world microprogrammed computers in use today have instruction sets running in the hundreds of instructions, so the savings can be extremely significant. The trade-off is that overall CPU operation will be slower, since time must be allotted for fetching data from the control store as well as from the nanostore.

V. BEYOND MICROCODE AND NANOCODE

There are two purposes for the foregoing exercise. One is to illustrate some concepts in computer science which have raised intellectual property issues, since there is a need and a desire for protection of the fruits of intellectual labor. The other purpose is to illustrate the difficulties encountered in applying intellectual property laws to rapidly changing technology. Technological progress does not appear to be slowing down. Computer science is not a stagnant field, and there are probably other concepts in computer architecture which are beginning to raise legal issues. If one is interested in other aspects of computer science, such as pipelining, array processing, bit-slice structuring of resources, and parallel processing, a number of excellent texts are

available.[20]

One important historical aspect of microprogramming is that it allowed the advent of emulation. Emulation is a method whereby the instruction sets of computers, other than a given one, are implemented with the resources of the given computer.[21] Emulation is an issue which has previously arisen in computer-related litigation[22] and continues to arise, such as in the NEC/Intel case.

Another issue is that the increasing availability of relatively inexpensive and fast read/write memories makes dynamic microprogramming attractive. Dynamic microprogramming is the use of fast read/write memories, rather than hardwired programmed logic arrays (PLAs) or read-only memories (ROM), for the storage of microprograms.[23] Having a separate read/write memory for a microprogram facilitates changes to the microprogram and gives the appearance of externality to the microprogram, as read/write memory makes the microprogram conveniently accessible to a programmer who wishes to program for the optimization of CPU resources. For this reason, it may be unwise to premise legal protection for programs strictly on the basis of the program being external to the computer.

In concluding the discussion on microcode and microprogramming, it has been observed that the term "microprogramming" is an inadequate description of the technique, and that what is actually occurring is "interpretive" programming.[24] Microinstructions might be viewed as a part of a computer programming language called a "directly executable language",[25] which merely serves to bring the programmer closer to the machine itself when writing a program. Accordingly, the question as to how close to the hardware can copyright protection for programs extend may depend on how strictly a court interprets the language of section 102(b).

The problem of differentiating hardware from software will not get any easier. The problem may be compounded by the increased usage of hardware description languages (HDLs)—computer program-like lan-

20. *See, e.g.*, A. TANENBAUM, *supra* note 8; P. FREEMAN, *supra* note 15.

21. *See* Flynn, *supra* note 15, at 79.

22. The case, *Data General Corp. v. Digital Computer Controls, Inc.*, 357 A.2d 105, (Del. Ch. 1975), was the first of a series of trade secret cases brought against manufacturers of Data General's Nova 1200 minicomputer emulators. The cases ultimately backfired and resulted in massive antitrust litigation. *See, e.g.*, Digidyne Corp. v. Data General Corp., 734 F.2d 1336 (9th Cir. 1984); *In re* Data General Corp. Antitrust Litig., 529 F. Supp. 801 (N.D. Cal. 1981); *In re* Data General Corp. Antitrust Litig., 490 F. Supp. 1089 (N.D. Cal. 1980).

23. *See* Cook & Flynn, *System Design of a Microprocessor, reprinted in* P. FREEMAN, *supra* note 15, at 85.

24. *See* Flynn, *supra* note 15, at 84.

25. *Id.*

guages for designing computer systems through the use of language-like techniques rather than through symbolic or diagrammatic techniques.[26] These HDLs allow computer hardware to be described in a language, which can then be converted by an existing computer into schematic diagrams, timing diagrams for studying resource timing relationships, and microprogram code for a proposed computer architecture. In the future, therefore, it may become virtually impossible to differentiate between a "computer program" for which one form of legal protection may be available, and an "electronic computer" for which another form of legal protection (such as the SCPA or patent laws) may be available.

VI. THE SCPA—DOES IT DO ANYTHING FOR MICROCODE?

The Semiconductor Chip Protection Act of 1984 (SCPA) creates a new form of intellectual property and constitutes a departure from prior U.S. law. Congress has previously sought to extend intellectual property protection to new technologies through the expansion of existing laws, for example, when it passed the 1980 amendments to the Copyright Act.[27] The SCPA is neither a patent law nor a copyright law. Though the SCPA contains aspects of both patent and copyright law, it more closely resembles copyright law.

The SCPA, being *sui generis*, attempts for the first time to meet the legal needs of new technology by creating a new, hybrid form of protection, rather than by attempting to adapt existing laws that were designed to protect the creations of inventors and authors. While credit could be given to the Founding Fathers for being farsighted enough to have contemplated that the Constitution should allow for such departures,[28] it is conceivable that a constitutional challenge might someday be made to the SCPA on the ground that a chip designer is neither an author nor an inventor. It is hoped that such an attack will not be seriously considered, since circuit designers must often possess skills of both authors and inventors to create new and useful semiconductor products.

The initial attempts in passing legislation for the protection of semiconductor products were not successful. This was due primarily to a split in the semiconductor industry over the goals of such legislation.[29] In 1983, the Senate version of the SCPA (also called the "chip bill")

---

26. *See* COMPUTER, Feb. 1985 (entire issue).

27. *See* 17 U.S.C. § 117 (1982).

28. U.S. CONST. art. I, § 8 states: "The Congress shall have Power . . . [t]o promote the Progress of Science and useful Arts, by securing for limited Times to *Authors* and *Inventors* the exclusive Right to their respective *Writings* and *Discoveries* (emphasis added)."

29. *See Hearings on H.R. 1007 Before the Subcomm. on Courts, Civil Liberties and the Administration of Justice of the House Judiciary Comm.*, 96th Cong., 1st Sess. (1979).

provided copyright protection for mask works,[30] but this provision was not in the House version.[31] The Senate was concerned that abandoning the copyright approach might result in uncertainty in litigation until the courts definitively interpreted the new legal concepts. Thus, the resulting law generally favors the copyright approach, explaining why the SCPA is administered by the Copyright Office in the Library of Congress.[32]

The SCPA applies to "semiconductor chip products" and "mask works."[33] A "semiconductor chip product" is a multi-layered product of metallic, insulating, or semiconductor material created in accordance with a predetermined pattern and intended to perform electronic circuitry functions. Although this definition probably was not meant to be extended to products other than semiconductors, the definition appears to be broad enough to cover products such as printed circuit boards and other types of circuit layouts.[34] A "mask work" is a series of related images, however fixed or encoded, that represents the three-dimensional pattern or topography of the chip's surface. The mask work is "fixed" in a chip when its embodiment is sufficiently permanent to enable it to be perceived or reproduced for more than a transitory period. The provision, *"however fixed or encoded,"* appears to be sufficiently flexible enough to include foreseeable advances in photolithography and other chip manufacturing technologies, as well as the use of hardware description languages which can be directly converted into chip layouts. Although there are several different types of "works" which could conceivably be protected by the SCPA,[35] only the chip configuration and layout were arguably intended to be the subjects of the legislation.[36]

---

30. *See* S. 1201, 98th Cong., 1st Sess., 129 CONG. REC. S5992 (daily ed. May 4, 1983).

31. *See* H.R. 5525, 98th Cong., 1st Sess. (1983).

32. *See* 17 U.S.C. § 903(c)(1) (Supp. II 1985). This law, originally H.R. 6163, substantially comports with H.R. 5525. *See also* 130 CONG. REC. S12,916 (daily ed. Oct. 3, 1984) (explanatory memorandum—Mathias-Leahy Amendment to S. 1201).

33. See 17 U.S.C. § 902 (Supp. II 1985).

34. *But see* Boorstyn, *The Doctrine of Fair Use*, 1 COPYRIGHT L.J. 1 (1985), who states, without citing any authority, that the definition does not include other products such as magnetic films or printed circuit boards.

35. These may include (1) schematic diagrams, layouts and the like represented in mylar, paper, photolithographic masks, and the like produced during the process of chip design and manufacture, (2) computer programs fixed on chips such as ROM or PLAs, (3) audiovisual works, program-related, generated or dependent, found "fixed" in chips for copyright purposes as in *Atari, Inc. v. North American Philips Consumer Electronics Corp.*, 672 F.2d 607 (7th Cir. 1982), and (4) the configuration, topology, architecture, or surface and subsurface appearance or pattern on the chips themselves. *See* Baumgarten & Patry, *Update on Software Publishers and Semiconductor Chip Legislation*, COMPUTER LAW., Feb. 1984, at 12.

36. *Id.*

These definitional provisions and the applicable legislative history are the only guidelines for determining what items are meant to be covered by the SCPA. There are, however, some provisions which exclude coverage in certain circumstances. Mask works which are not original are excluded from protection;[37] however, the term "original" is not defined. If traditional copyright concepts were to be used, then "original" would mean works that are independently created and not copied from another.[38] Also excluded are mask works, which consist of designs that are staple, commonplace, or familiar in the semiconductor industry, or variations or combinations of such designs, that when considered as a whole, are not original.[39] The issue of originality may potentially be a fertile ground for litigation, especially if an accused infringer is reproducing products that have been on the market for some time or embody only marginal differences over arguably familiar products. A defense of invalidity based on "commonplace" or "staple" grounds is most likely to be raised for gate arrays and memory devices (which generally have repetitive basic structures).

When litigating the issue of originality of designs, or of variations or combinations thereof, expert testimony may be required. One commentator has suggested that the determination of originality will depend on whether a mask work is merely an insubstantial variation of the prior art as it existed at the time of the registration of the mask work, or whether the effort, expense, and original contributions resulted in a new work when considered as a whole.[40] Under this view, a mask work must have resulted from substantial effort and investment, and it must contain more than insubstantial variations on the prior mask work.[41] This leads to the next type of exclusion from coverage under the SCPA, "reverse engineering."

The copying of a mask work solely for the purpose of teaching, analyzing, or evaluating the concepts or techniques embodied in the mask work, or the circuitry, logic flow, or organization of the components used in it, does not constitute an infringement.[42] It can be argued that once the mask work has been lawfully copied solely for these legitimate purposes, the reverse engineer may then incorporate the results of such analysis in creating his own "original" work, which would then be entitled to its own protection.[43] The requirement that a second mask work derived from the analysis of a protected mask work be created by re-

37. 17 U.S.C. § 902(b) (Supp. II 1985).
38. *See* M. NIMMER, *supra* note 6, at § 2.01[A] and cases cited.
39. 17 U.S.C. § 902(b) (Supp. II 1985).
40. *See* Boorstyn, *supra* note 34, at 2-3.
41. *Id.*
42. 17 U.S.C. § 906 (Supp. II 1985).
43. *See* Boorstyn, *supra* note 34, at 3-4.

verse engineering seems to be stating that the second work is not, under these circumstances, "copied" from the protected work. While this concept was not present in any earlier versions of the House or Senate bills, the version embodied in the SCPA seems to be a restatement of the idea/expression dichotomy found in copyright law.[44] Under this approach, the creator of a second mask work is, arguably, free to copy and use the ideas, concepts, or principles embodied in a lawfully-analyzed protected work.

Distinguishing between the idea embodied in a mask work and the form or manner of expression is not easy. Such differentiation, however, is necessary to determine whether or not works, as a whole, are old and staple or have been created with effort and original contribution. While the hybrid SCPA does not set forth tests such as that of "obviousness" found in the present patent laws,[45] patent law cases might help in making originality determinations. These cases, for example, caution against the dissection of old elements which may form a new combination because of the risk that the subject matter will not be analyzed as a whole.[46]

In applying these concepts to microcode, it is expected that mask work protection will be sought for microprogrammed microprocessors, since the unlicensed second-sourcing and pirating of such products provided much of the original impetus for the legislation.[47] The Intel/NEC litigation has already demonstrated that determinations along the lines of "reverse engineering" for microcode on the 8088/8086 microcomputers may be necessary; however, that litigation is premised on the copyright laws and not the SCPA or patent laws. Yet, if the case goes to trial, it may prove instructive in making these determinations.[48]

---

44. *See* Baker v. Selden, 101 U.S. 99 (1879), where the Supreme Court first ruled that copyright law did not provide protection against the appropriation of an idea or process utilized in an author's work, but rather protected only the particular "expression" adopted by the author to convey his idea or process.

45. "A patent may not be obtained . . . if the differences between the subject matter sought to be patented and the prior art are such that the subject matter *as a whole would have been obvious* at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains." 35 U.S.C. § 103 (Supp. II 1985) (emphasis added).

46. Many of these cases deal with situations wherein the Patent Office has rejected a patent for a claimed invention by creating a hindsight reconstruction of the invention through the combining of teachings from one or more prior art references. The courts are quick to state that such hindsight reconstructions are tantamount to saying that the claimed invention would have been "obvious to try" by combining the prior art references, which is impermissible in evaluating an invention "as a whole." *See, e.g., In re* Sernaker 702 F.2d 989 (F. Cir. 1983); *In re* Yates, 663 F.2d 1054 (C.C.P.A. 1981).

47. S. 1201, *supra* note 30, 129 CONG. REC. S5991-92 (introductory remarks of Sen. Mathias).

48. As of the final submission of this article, cross-motions for summary judgment

NEC alleges that it obtains certain functional advantages in its V20 design by altering the resources. For example, the V20 employs a dual internal data bus structure which allows two registers in the ALU to be loaded simultaneously.[49] The 8086/8088 microcomputers allegedly have only one internal data bus,[50] as does the exemplary CPU described above. Moreover, the microinstructions in the V20 are wider, since they must control more resources simultaneously[51] (such as selecting which one of the two internal data buses from which a selected register will receive data). In addition, the generation of addresses in the V20 are allegedly performed by a dedicated address-generating resource not present in the 8086/8088, thus enabling the V20 to generate memory addresses in two clock cycles as opposed to five to twelve clock cycles in the 8086 and 8088.[52] Other differences include resources in the V20/V30 such as a shift and loop counter (for certain arithmetical operations) and a prefetch pointer (for fetching the next instruction before completion of the present one to speed up branching operations).[53]

The presence of these additional resources—were the issue solely one of the infringement of a protected mask work—should probably require a finding that substantial reverse engineering resulted in the creation of an original work entitled to its own protection. The use of different, wider microinstructions arranged in different sequences to obtain different performance characteristics would make a determination of mask work infringement difficult.

Intel maintains, however, that the 8086/8088 microcode fixes certain starting addresses for microcode sequences at arbitrary locations. In the NEC chip, 257 out of 302 of these arbitrary starting addresses are the same as in the Intel chips.[54] The duplication of these starting addresses might very well suffice to establish a pattern of sufficient speci-

---

were pending in the Intel case directed to the issue of whether the copyright laws afford protection for microcode, and in particular, whether there were any material issues of fact for trial. The motions and briefs were not available for review due to a protective order. *See Intel Asks Court Judgment on Microcode Copyright Protection . . .*, ELECTRONICS WEEK, July 1, 1985, at 12.

49. NEC Complaint, *supra* note 5, at ¶ 14.

50. *Id.*

51. *Id.* at ¶ 15 (the V20/V30 have twenty-nine-bit wide microinstructions, while the 8086/8088 have twenty-one-bit wide microinstructions).

52. *Id.* at ¶ 14.

53. *Id.*

54. Duffy, *Intel Court Bid to Bar NEC Chips Could Cost Users Strong Product*, PC WEEK, Mar. 5, 1985, at 8. In Fig. 9 of the exemplary CPU architecture described above, the corresponding "arbitrary" starting microcode sequences are Step 1 for the ADD instruction and Step 12 for the MOV instruction. It would have been just as easy to switch these two sequences of microinstructions in Fig. 9 so that the MOV microinstructions would begin at Step 1 and extend through Step 6, and the ADD microinstructions would begin at Step 7 and extend through Step 17.

ficity to enable a court to hold that there is substantial similarity under the copyright laws to establish infringement.[55]

The NEC/Intel case, however, is based on copyright and not mask work infringement (the 8086 was commercially distributed more than two years ago, thus not qualifying for mask work protection[56]). In any event, the SCPA should apply to proscribe the outright duplication of a protected mask work, which also happens to be a microprogrammed microcomputer. When the issue, however, is whether the SCPA can be applied to protect the instruction set of such a microprogrammed computer or to protect against modifying the microinstructions of a protected mask work to alter the performance characteristics of the chip, the conclusion that the SCPA applies is not as clear. Judicial application of the principles of originality, reverse engineering, and substantial similarity to allegedly infringing works must be sought in an appropriate case.

VII. COPYRIGHT PROTECTION AND MICROCODE

Although copyright issues have already been discussed with regard to the SCPA, copyright issues raised in the NEC/Intel litigation must now be examined. For example, does the existence of different types of microinstructions, arranged in different sequences to accomplish certain functions in a more efficient manner (as for increased speed and flexibility) as well as to execute the same machine language instruction set, require a finding of copyright infringement? That is, is Intel seeking to protect the instruction set of the 8086/8088? This appears to be the basic goal of Intel in the litigation, since there are clear economic advantages in being the sole lawful manufacturer of a computer having a popular and widely used instruction set. Protection of the instruction set for the 8088 (used in the IBM PC series of microcomputer systems) would allow Intel the enviable position of being the *only* manufacturer of microprocessors that could be used in such computer systems while still maintaining software compatibility. Any advantages obtained by one

55. In Nichols v. Universal Pictures Corp., 45 F.2d 119, 121 (2d Cir. 1930), *cert. denied*, 282 U.S. 902 (1931), Judge Learned Hand stated his famous and oft-quoted "abstractions" test for substantial similarity:

> Upon any work . . . a great number of patterns of increasing generality will fit equally well, as more and more of the incident is left out. The last may be no more than the most general statement of what the play is about, and at times consist of only its title; but there is a point in this series of abstractions where they are no longer protected, since otherwise the playwright could prevent the use of his 'ideas,' to which, apart from their expression, his property is never extended.

It is perhaps worthwhile to ask, is the machine language instruction not at least one abstraction level removed from the microinstructions which implement the instruction?

56. 17 U.S.C. § 908 (Supp. II 1985).

who expanded on or improved the execution of the instruction set would accrue to the originator of the instruction set. Products such as the V20 and V30 (or other emulators) would be impossible to produce legitimately without some form of license from the owner of the instruction set.[57]

The question can be posed in a slightly different manner. Is not the instruction set for any given computer, if such computer is microprogrammed, merely an expression of the microinstruction set, thus being entitled to copyright protection? Microinstructions can be viewed as a "set of statements or instructions . . . used directly or indirectly in a computer in order to bring about a certain result,"[58] namely, the result of interpretation and execution of the machine language instructions. The instruction ADD X, Y of Fig. 11 is a different instruction from that of Fig. 9. It should therefore be apparent that a different microprogram can make a different computer, even though the hardware itself (that is, the available resources) is no different. Thus, the question can be asked, is not a microprogram (with its microinstructions) a set of statements or instructions to be used directly or indirectly in a computer in order to bring about a certain result?[59]

On the other hand, is not the microprogram also a "procedure, process, system, [or] method of operation"[60] which is expressly denied copyright protection? The close proximity of microcode to the control of the physical resources can be expected to be problematic for the courts, since there is arguably no purpose for microcode, whatsoever, other than to control the *processing* of machine language instructions within the computer or to implement the particular *method of opera-*

57. One possible result of a finding that protection is available for an instruction set is the creation of legal relationships akin to "dominant/subservient" patents. *See, e.g.*, N.J. Zinc Co. v. Singmaster, 71 F.2d 277, 279 (2d Cir. 1934), where the court said "[t]here should be no confusion between he right of the owner of a dominating patent to prevent infringement thereof by improvement patents and the right of the improver to his improvements and his right to prevent infringement of the improvement patents by the owner of the dominating patent." The improvement or "subservient" patent owner cannot legally sell his improvement because it is covered by the dominant patent, nor can the dominant patent owner sell the improvement to his invention because it is covered by the subservient patent; cross-licensing arrangements are sometimes employed to solve this dilemma.

A similar situation might occur if the instruction set is protected under the copyright laws, but the improvement is protected by the SCPA. The owner of the copyrighted microcode for a given instruction set could not legally produce the improved version of the same computer protected as a mask work, nor could the owner of the mask work on the improved computer sell the improved computer because of the copyrighted instruction set.

58. 17 U.S.C. § 101 (1982) (defining "computer program").

59. *Id.*

60. *Id.* § 102(b).

tion of the computer in response to a particular machine language instruction. Microcode is therefore very close to the boundary of what is copyrightable.

One commentator has proposed that the critical question of copyrightability for microcode is whether the computer is "authored" or "built." This question is posed by asking whether the microprogrammer is manipulating symbols or setting circuits on or off.[61] Under this approach, the issue is determined solely by looking not at the microcode itself, but at how the microprogram is created. If it is done by setting circuits, then the result is not copyrightable. But if it is done by manipulations in accordance with a microprogramming "language," then it is copyrightable. Additionally, microprograms that use a previously-created microinstruction set should be copyrightable.[62] This appears to recognize that once the resources for a particular computer architecture have been selected (thereby determining what the microcode must physically manipulate), an arrangement of these microinstructions to carry out new instructions in the instruction set, if done symbolically, constitutes a copyrightable work.

This approach has the appeal of simplicity and is relatively easy to apply; all that needs to be done is to examine the work product of the engineers and see if symbols were manipulated in a "language" or if schematic drawings were scribbled upon. In other words, the issue of copyrightability would be determined by examining how the work is created—a method not unlike distinguishing between authorship and invention. As with authorship and invention, however, there is sometimes overlap. For example, in the landmark computer patent case of *Diamond v. Diehr*,[63] the program at issue would no doubt have been considered copyrightable under present copyright laws. This approach does not provide an answer as to whether an instruction set, however created, should be copyrightable. It seems unfair to offer copyright protection for any microcoded computer which is created by a person who manipulates symbols, but to deny the same protection for a computer which performs the same task in exactly the same manner created by an individual who prefers working with an oscilloscope and wires to "hacking code" (i.e., writing programs). Moreover, it is doubtful that very much microcode is produced without some combination of both symbol and circuit manipulation.

The mere examination of how a work is created should not be

61. Davidson, *Protecting Computer Software: A Comprehensive Analysis*, 23 JURIMETRICS J. 337, 390 (1983).

62. *Id.*

63. 450 U.S. 175 (1981) (claim for a process for curing rubber held to be patentable subject matter; a computer program controlled the process).

wholly determinative of the applicability of copyright protection (or any other protection, for that matter). Merely looking at how a microprogrammed computer was developed will penalize creative computer scientists who prefer working mainly in the hardware domain as opposed to the software domain, and may force companies who invest significantly in computer system development to shift towards symbol manipulation design techniques at the expense of more efficient designs.[64]

In the event that a court decides that microcode in a microprogrammed computer is copyrightable (as is possible in the Intel/NEC case), the following corollary may also be argued: there should be protection for the computer's instruction set, since it will not be possible to produce a set of microcode which does not contain substantial similarities and which still executes the same instruction set. But what if a given computer architecture is not microprogrammed, or what if someone constructs a nonmicroprogrammed version of a microprogrammed computer? As described above, there is no absolute requirement that microprogramming be employed to implement any particular computer CPU instruction set. If protection for an instruction set can be obtained by first creating the instruction set on a microprogrammed CPU, manufacturers would no doubt first implement new computers using symbol manipulation microprogramming techniques and leave advanced development for higher speed execution with discrete logic for later, after protection for the instruction set had been firmly established.

With respect to nanoprogramming, if a manufacturer of a CPU having Type A microinstructions claims copyright infringement against a manufacturer of a CPU having Type B microinstructions (e.g., see Figs. 8 and 12), will a court find that there is substantial similarity between a microprogram of Type A microinstructions and a microprogram of Type B microinstructions? Unquestionably, the end results of both microprograms would be the same, yet in this case, there is extra hardware (the decoder to select a register) required to carry out the microinstruction. This question may be faced by the court in the Intel/NEC litigation, since it is alleged that the respective microinstructions of the two computers at issue have different microinstruction widths.[65]

CONCLUSION—WHY NOT ANOTHER *SUI GENERIS* LAW?

It is possible that the question of protection for microcode may not

64. In this author's opinion, the increased usage of hardware description languages (HDLs) may not make this such an undesirable consequence, provided that the efficiencies of operation traditionally associated with nonmicroprogrammed architectures can be preserved.

65. *See supra* note 54.

be finally disposed of in the Intel/NEC litigation. Only copyright issues have been raised (or could be raised), but copyright law seems to be ill-equipped for determining whether instruction sets should be protected because of the differences between the 8086/8088 and the V20/V30 resources and microinstructions. Proposed approaches for adapting existing copyright law to microcode, such as examining how the microcode was created, are not desirable. This is another example of straining the existing law to cover something for which the law was not intended. Technology continuously changes, yet there is still a propensity in trying to apply old law to new problems. The inevitable result is that there will be legal and economic uncertainty until there is a definitive ruling or legislative enactment.[66]

What, then, can be concluded about copyrightability for microcode? For that matter, can any conclusion be drawn that microcode is legally protectable? As discussed, a finding of *de jure* copyrightability for microcode, as in the Intel litigation, may result in the *de facto* protection for the instruction set, especially if one focuses on the substantial similarities which necessarily result when one computer manufacturer attempts to create an emulation or improved version of another manufacturer's successful product. There is no question that a manufacturer of a highly successful product such as the Intel 8086/8088 desires to maintain this success. Maintaining such success, however, may result in higher prices for consumers and slower technological progress, since competition will be chilled.

The SCPA does not really resolve any of the problems, since different resources and microinstructions in the emulator product strongly suggest reverse engineering. The SCPA was not intended to address any microcode related issues directly, although it has the effect of forcing careful documentation by those who employ reverse engineering.

Although this Article does not discuss in detail the applicability of the patent laws to microcode, the author, whose primary experience lies in the applicability of the patent laws to electronics and computer-related technologies, believes that in the proper case, the patent laws will cover microcoded computer architectures, but not instruction sets. This conclusion is based on the author's experience that the functions of the computer, if sufficiently novel, can be patented, as both a process and as a structure for carrying out the function. Patent protection, however, is time consuming and costly to obtain, and even more time consuming and costly to enforce. In many cases, and especially in computer archi-

66. While this author recognizes the desirability of slow, deliberate legal change in situations when significant and possibly undesirable social change would be engendered by hasty, ill-considered laws, the author questions whether such slow change is desirable when economic change or economic detriment is at issue.

tecture, the technology and the market have advanced before a patent can issue. While patent protection is superior in the proper case, as when truly significant innovations are involved and not just a slight evolutionary improvement, it is not acceptable for many types of technological innovation. Even improvements that are slight or evolutionary require capital investment, thus deserving protection.

Congress should, first of all, consider whether protection for a computer's instruction set is desirable as a matter of public policy. This would resolve many questions of whether different computer architectures (i.e., particular resource arrangements and microprograms for carrying out the instruction sets) deserve protection. This would prevent the forcing of reliance on patent laws where protection may be slow in coming or simply may not be available if the technical differences do not amount to "invention" but are nonetheless economically important.

What may then be needed is a new *sui generis* law for microcode, especially if Congress deems it important for those who created the computer and invested in making it a commercial success to retain exclusive domain over the instruction sets. The SCPA, hopefully, signifies a recognition by Congress that *sui generis* laws are an appropriate and convenient approach for handling rapidly changing technology when important economic consequences are at stake.

At the very least, Congress should amend the SCPA or the Copyright Act so that it clearly states that microcode, however created, is considered a protected work. There is already ample precedent in the Computer Software Act of 1980[67] for amending intellectual property laws to accommodate changing technology. If instruction set or microcode protection is not desirable, then it is a simple matter to provide an amendment to this effect as well, so that computer makers know where they stand. The matter of protection deserves consideration by the body best suited for the task—Congress—instead of, as all too often happens, forcing the issue on the overburdened courts.

---

67. *See supra* note 27.

FIG. 1

Modern Multi-Level Computer Organization

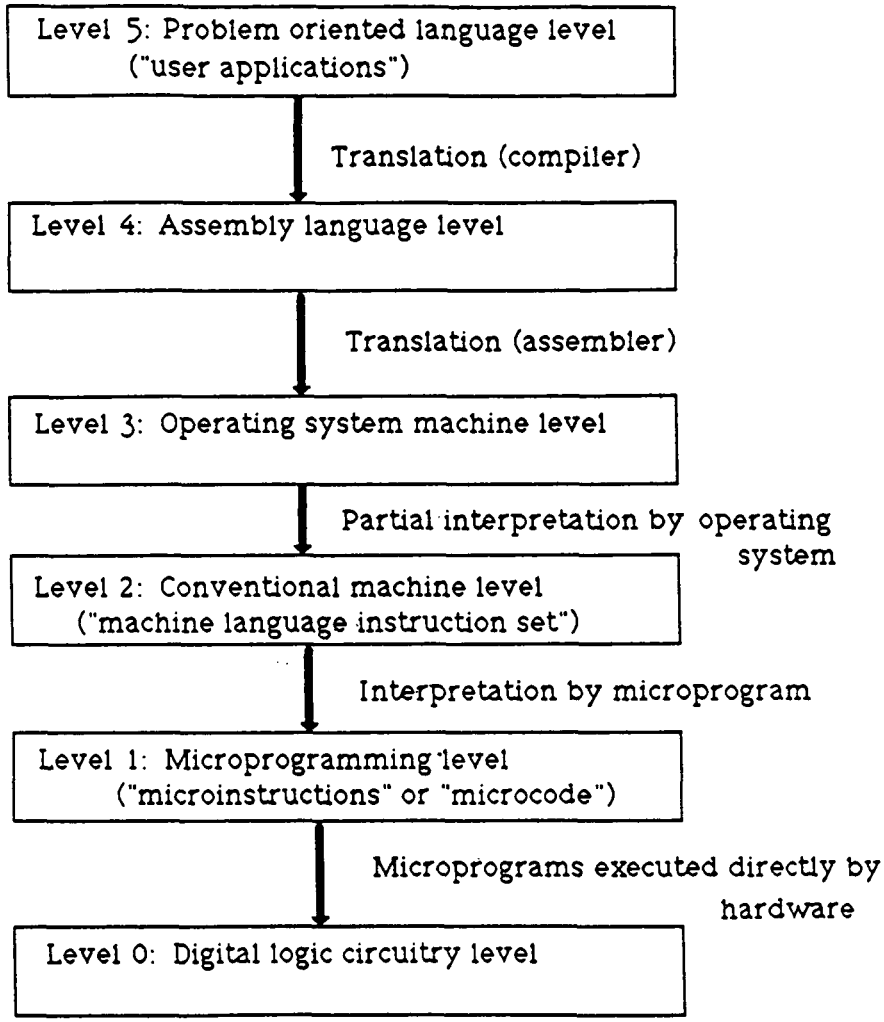

FIG. 2

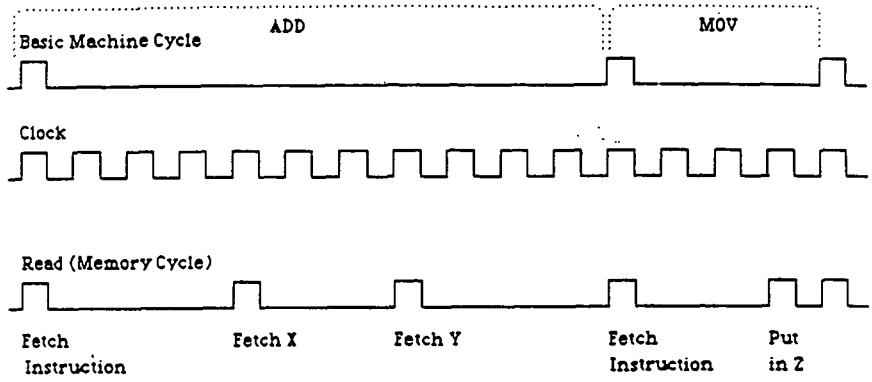

FIG. 3

Instructions in instruction set: ADD X,Y (add the number contained in the address following the instruction (X) to the number following in the next address (Y); leave the results in the accumulator (ACC))

MOV ACC,Z (move the contents of the accumulator to the address following the instruction)

ADD Steps

1. Fetch the instruction from memory
2. Put it in the instruction register/decoder
3. Decode the instruction to see what it is
4. Fetch the number X
5. Place in register A
6. Fetch the number Y
7. Place in register B
8. Store the results of addition in the accumulator

MOV Steps

1. Fetch the instruction
2. Place in instruction register
3. Decode instruction
4. Transfer the contents of the accumulator to the address following the instruction

FIG. 4

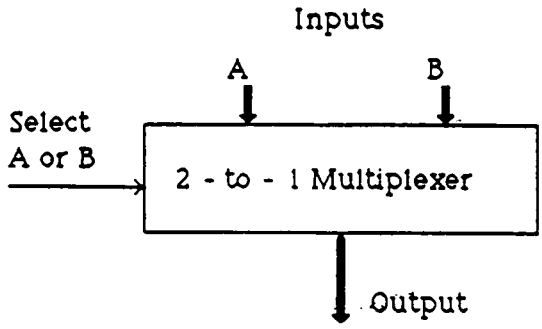
Inputs
A
B
Select
A or B
2 - to - 1 Multiplexer
Output

FIG. 5

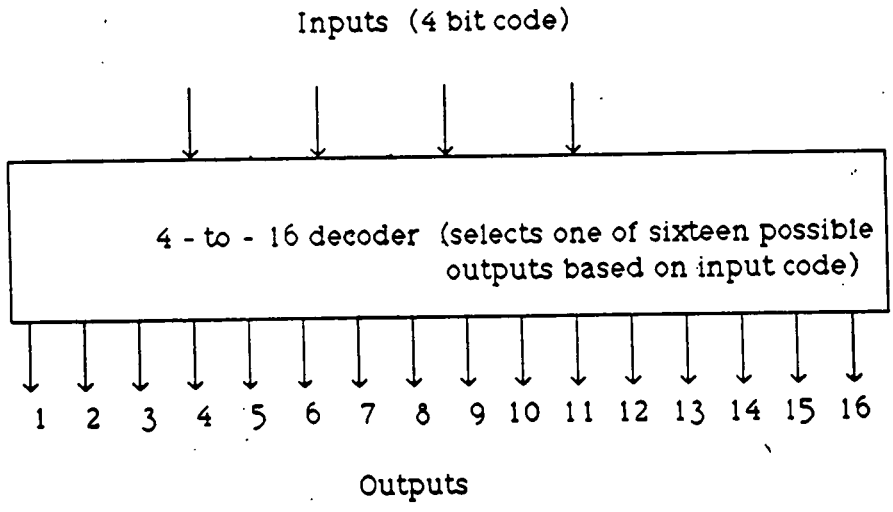
Inputs (4 bit code)
4 - to - 16 decoder (selects one of sixteen possible outputs based on input code)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Outputs

FIG. 6

Exemplary CPU Architecture

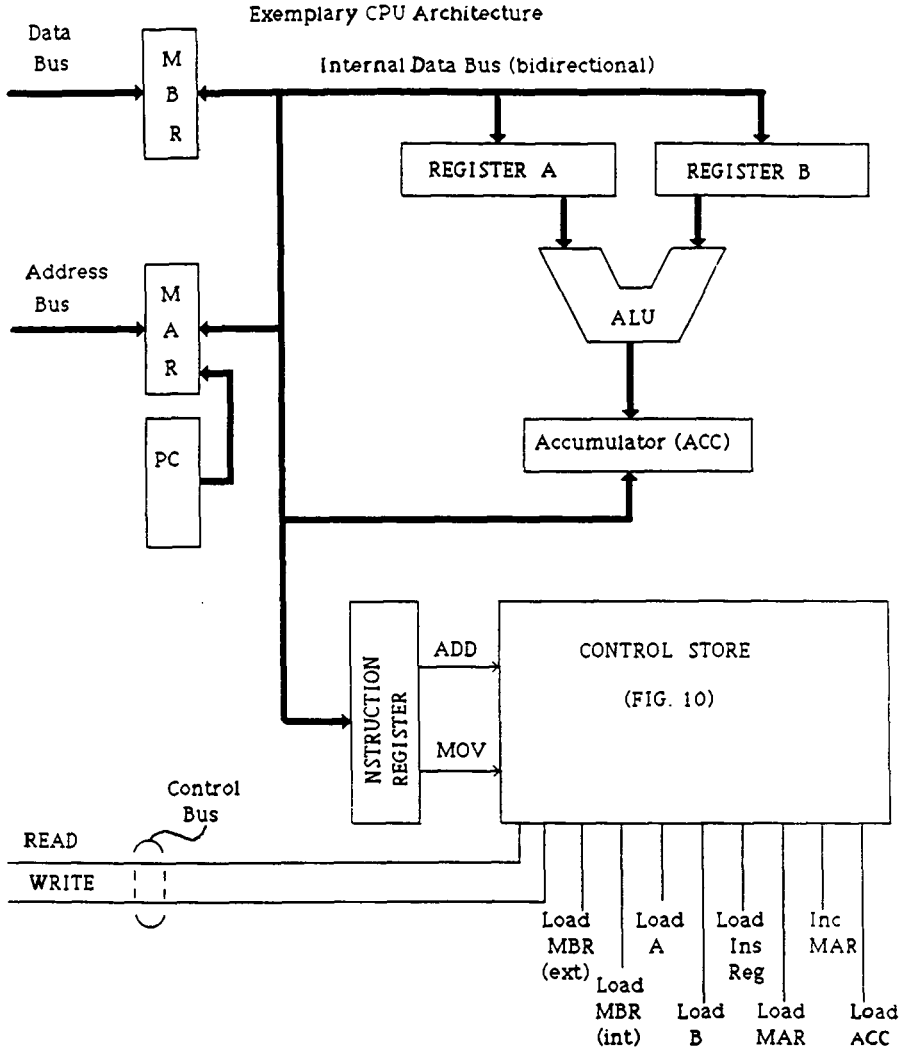

FIG. 7

Resource	Possible Commands
1. Memory Buffer Register (MBR)	1. Load MBR from external data bus 2. Load MBR from internal data bus
2. Register A	3. Load from internal data bus
3. Register B	4. Load from internal data bus
4. Instruction Register (INS REG)	5. Load from internal data bus (decodes instructions)
5. Memory address register (MAR)	6. Load from internal data bus 7 Increment (add 1 to contents)
6. Memory control READ, WRITE	8. Tell memory to READ from MAR address 9. Tell memory to WRITE to MAR address

FIG. 8

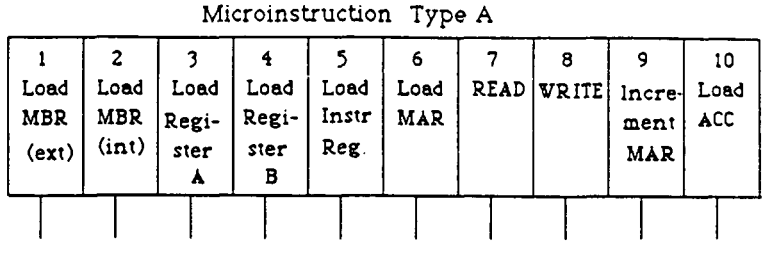

FIG. 9

STEP	Microinstruction									
ADD X,Y	1 Load MBR (ext)	2 Load MBR (int)	3 Load A	4 Load B	5 Load Ins. Reg.	6 Load MAR	7 READ	8 WRITE	9 Incr MAR	10 Load ACC
1 Fetch instruction	0	0	0	0	0	1	1	0	0	0
2. Load data into MBR	1	0	0	0	0	0	0	0	0	0
3 Place in ins. reg.	0	0	0	0	1	0	0	0	0	0
4 Decode	0	0	0	0	0	0	0	0	0	0
5 Fetch X	0	0	0	0	0	0	1	0	1	0
6. Load data into MBR	1	0	0	0	0	0	0	0	0	0
7. Place in A	0	0	1	0	0	0	0	0	0	0
8. Fetch Y	0	0	0	0	0	0	1	0	1	0
9. Load data into MBR	1	0	0	0	0	0	0	0	0	0
10. Place in B	0	0	0	1	0	0	0	0	0	0
11. Load to ACC	0	0	0	0	0	0	0	0	0	1
MOV ACC, Z										
12. Fetch instruction	0	0	0	0	0	1	1	0	0	0
13. Load data in MBR	1	0	0	0	0	0	0	0	0	0
14. Load ins. reg.	0	0	0	0	1	0	0	0	0	0
15. Decode	0	0	0	0	0	0	0	0	0	0
16. Load ACC to MBR	0	1	0	0	0	0	0	0	0	0
17. Increment MAR and write to memory	0	0	0	0	0	0	0	1	1	0

FIG. 10

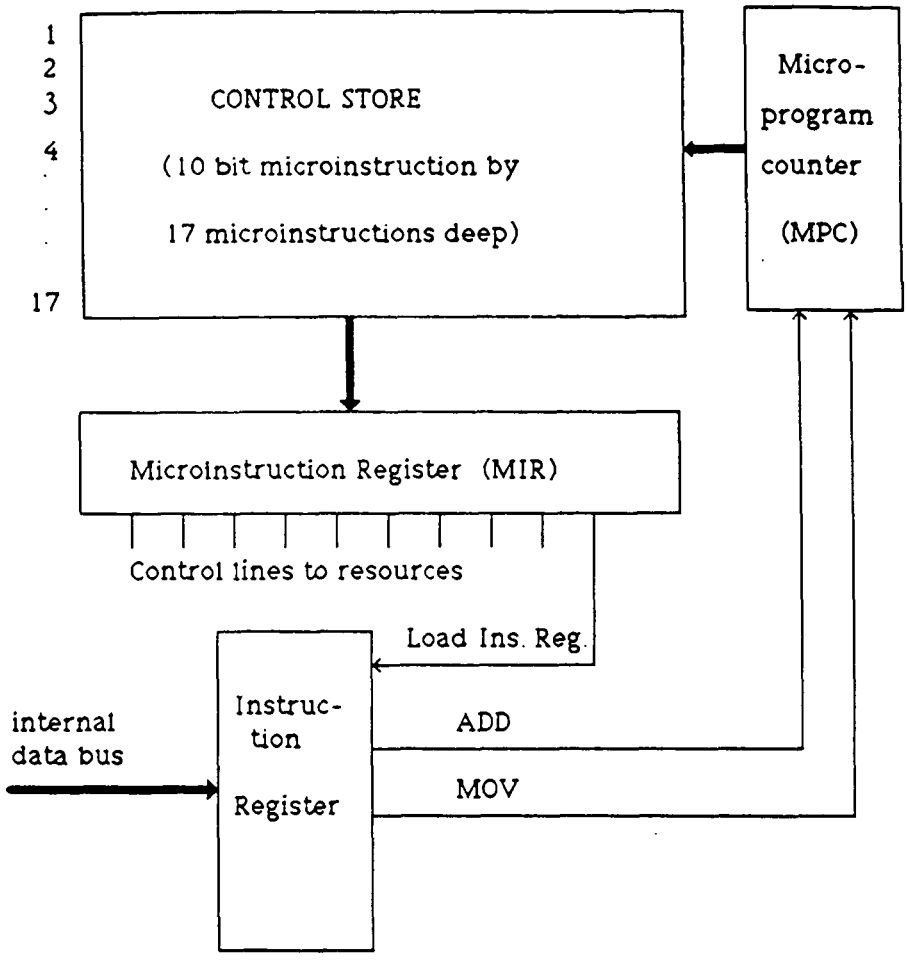
1
2
3
4
17
CONTROL STORE
(10 bit microinstruction by
17 microinstructions deep)
Micro-
program
counter
(MPC)
Microinstruction Register (MIR)
Control lines to resources
Load Ins. Reg.
internal
data bus
Instruc-
tion
Register
ADD
MOV

FIG. 11

STEP ADD X,Y (where X and Y contain addresses of data to add)	Microinstruction 1 Load MBR (ext)	2 Load MBR (int)	3 Load A	4 Load B	5 Load Ins. Reg.	6 Load MAR	7 READ	8 WRITE	9 Incr MAR	10 Load ACC
1. Fetch instruction	0	0	0	0	0	1	1	0	0	0
2. Load data into MBR	1	0	0	0	0	0	0	0	0	0
3. Place in ins. reg.	0	0	0	0	1	0	0	0	0	0
4. Decode	0	0	0	0	0	0	0	0	0	0
5. Fetch address X	0	0	0	0	0	0	1	0	1	0
6. Load data into MBR	1	0	0	0	0	0	0	0	0	0
7. Save MAR in ACC	0	0	0	0	0	0	0	0	0	1
8. Move MBR to MAR	0	0	0	0	0	1	1	0	0	0
9. Load data in MBR	1	0	0	0	0	0	0	0	0	0
10. Move ACC to MAR	0	0	0	0	0	1	0	0	0	0
11. Move MBR to A	0	0	1	0	0	0	0	0	0	0
12. Fetch address Y	0	0	0	0	0	0	1	0	1	0
13. Load data in MBR	1	0	0	0	0	0	0	0	0	0
14. Save MAR in ACC	0	0	0	0	0	0	0	0	0	1
15. Move Y to MAR	0	0	0	0	0	1	1	0	0	0
16. Load data in MBR	1	0	0	0	0	0	0	0	0	0
17. Move ACC to MAR	0	0	0	0	0	1	0	0	0	0
18. Move MBR to B	0	0	0	1	0	0	0	0	0	0
19. Load results in ACC	0	0	0	0	0	0	0	0	0	1

FIG. 12

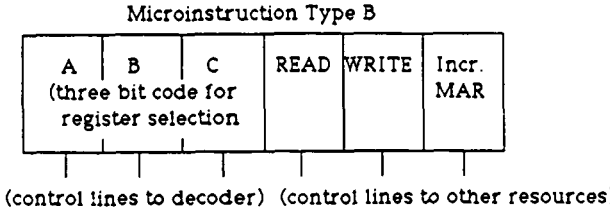

FIG. 13

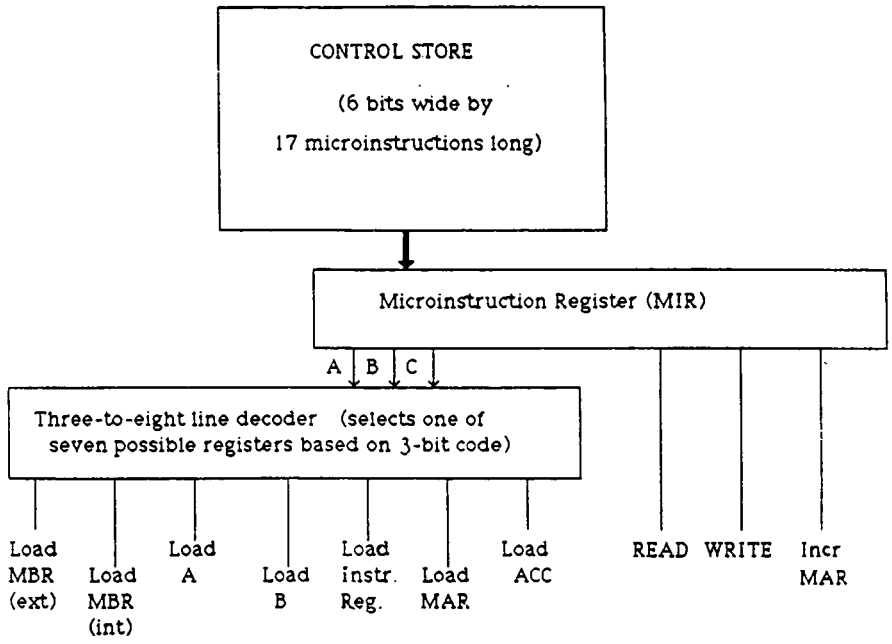

FIG. 14

STEP / Different Microinstruction Types / CODE	1 Load MBR (ext)	2 Load MBR (int)	3 Load A	4 Load B	5 Load Ins. Reg.	6 Load MAR	7 READ	8 WRITE	9 Incr MAR	10 Load ACC
	Microinstructions - stored in nanostore									
000 Fetch instruction	0	0	0	0	0	1	1	0	0	0
001 Load data into MBR	1	0	0	0	0	0	0	0	0	0
010 Load instr. reg.	0	0	0	0	1	0	0	0	0	0
011 Fetch data (X or Y)	0	0	0	0	0	0	1	0	1	0
100 Move MAR toACC	0	0	0	0	0	0	0	0	0	1
101 Move ACC to MAR	0	0	0	0	0	1	0	0	0	0
110 Move MBR to A	0	0	1	0	0	0	0	0	0	1
111 Move MBR to B	0	0	0	1	0	0	0	0	0	0

FIG. 15

Steps	Code	
1	000	Control Store for Nanoinstructions for ADD X,Y ,
2	001	where X and Y contain addresses of data to add
3	010	(these codes may be considered "nanoinstructions")
4	100	--- (* Note: this is a "do nothing" instruction as used here;
5	011	it has no substantive effect and is chosen to
6	001	allow us to have 8 instead of 9 different
7	100	types of microinstructions)
8	000	
9	001	
10	101	
11	110	
12	011	
13	001	
14	100	
15	000	
16	001	
17	101	
18	111	
19	100	