mode allows five positions of the multiplex channel bus *O*-register to be gated to multiplex tags-in. This feature is used in determining whether to execute or bypass the basic test after a system reset. The diagnostic latch is used to latch the priority latch which, in turn, blocks all traps.

The suppress malfunction trap latch, which is controlled by a microprogram instruction, was added for use when in diagnostic mode. With the suppress malfunction trap latch ON, the machine will operate in check stop mode and allow zero test stops. When the processor is in check stop mode it can be stopped by machine checks. With the suppress malfunction trap latch OFF, the processor operates in disable mode, and it will not stop on machine checks or zero test stops.

Another addition to the Model 30 CPU logic circuitry was the even latch. This latch, which is under microprogram control, causes the output from the ALU to be in even parity. Thus, the microprogram can put even parity data in any register or R/W storage location. Turning this latch ON also blocks the "zero" and "four" bits from one side of the ALU output and can therefore cause an ALU check.

Another way of causing ALU checks is with the ALU check latch. This latch is controlled by a microprogram instruction and can cause ALU checks if the processor is in diagnostic mode.

This one test has illustrated the use of all the special hardware latches added to the CPU circuitry plus the function of various positions on the CHECK CONTROL switch. Normally, when no diagnostic or service function is needed, the processor runs with all switches set to PROCESS. Another test, ROS scan, demonstrates the use of the ROS SCAN postion of the ROS CONTROL switch.

B. ROS SCAN

This program scans every word in READ-ONLY storage checking for the correct parity in all the fields of the ROS word that are parity checked. Several problems must be addressed in scanning the READ-ONLY storage. There are always one or more unpopulated boards in the CCROS, thus the ROS scan program must be able to recognize an unpopulated board and ignore the errors caused by reading out a blank word. Some of the words being scanned are *CF* stop words which normally stop the CPU clock; thus some means must be provided to inhibit stopping on *CF* stop words. It is also possible for *A*-register checks to occur when reading out certain words. The words involved are used to gate the bus-in lines to the *A*-register. Since there are no data on the bus-in lines at this time, an *A*-register check will occur. Thus, some means must be provided to prevent *A*-register checks from occurring when these words are scanned. Many other special diagnostic functions are required for this test, but these which have been mentioned are sufficient to show the power and flexibility available when using microprogramming for diagnostics.

VII. CONCLUSION

Creating a diagnostic package using microprogramming provides many capabilities that would not be available if diagnostics were provided in machine language only. The use of microdiagnostics reduces the amount of hard core to less than 1 percent of the total CPU logic circuitry, which might exceed 50 percent otherwise. Also, diagnostic microprogramming allows absolute control of the CPU operation at all times. This degree of control is not possible with a machine language program or normal microprogramming. Microprogramming combined with diagnostic hardware makes providing automatic fault location for the Model 30 CPU possible. Excluding the manual controls and the circuitry located in the CPU which is used with the console typewriter, more than 90 percent of the failures can be detected and located. Further, about 1/2 percent resolution is achieved for approximately 85 percent of the failures, i.e., approximately 85 percent of the failures are fault-located to four or less SLT cards.

Without microprogramming it would be impossible to provide a meaningful and practical test for some of the hardware, such as the ROS. There is no feasible way for a machine language program to scan through every position of ROS. The selector channel FLT was made possible due to the successful combining of machine language programming and microprogramming. Not enough ROS addresses were available to perform a complete job with microprogramming alone, and the machine language program was unable to manipulate the hardware well enough to detect and locate an adequate number of failures. By creating a macro–microprogram (combination machine language program and microprogram) approximately 93 percent of the failures in the selector channel can be detected and located automatically.

Another application of microprogramming was the I/O MICRO. This microprogram simplifies programming of I/O operations to such an extent that an individual totally unfamiliar with the system could be taught to perform I/O operations in a few minutes. This microprogram allows anyone using it to issue any allowable command to any I/O device on any I/O channel.

On future systems, the need for total automatic fault location and more powerful service aids will become even more critical. The microdiagnostics for the System 360 Model 30 represents one of the early steps taken to meet this need.

Microdiagnostics for the Standard Computer MLP-900 Processor

RONALD M. GUFFIN

Abstract—This note describes the microdiagnostic approach being used on the Standard Computer MLP-900 processor. This machine is microprogrammable and designed for general-purpose emulation, simulation, and interpretation. As a consequence it has a number of features that make it unusually well suited for the application of microdiagnostics. These include a READ–WRITE control memory, one clock microinstructions, a general-purpose microinstruction repertoire, almost complete microprogram access to the hardware, and the ability to specify test operands as immediate values. These features plus the highly encoded form of microprogramming used greatly

Manuscript received August 15, 1970.

The author is with Standard Computer Corporation, Santa Ana, Calif. 92705.

facilitate the design and programming of component-oriented test programs. This results in a microdiagnostic which is very compact, has a very short execution time, and requires a relatively short development time. Other advantages include a relatively low hardcore, very thorough testing, and high fault resolution.

Index Terms—Diagnosis, fault detection, microdiagnostic, MLP-900.

INTRODUCTION

One of the significant problems associated with the use of data processing equipment is that of keeping a system operational despite the occurrence of faults. Even though the CPU has generally been the most reliable component in the system, it has been the most difficult to maintain, primarily because of its complexity, but also paradoxically because of its reliability, which tends to limit the amount of troubleshooting experience that is attained. To alleviate this problem, it has been common practice to provide a diagnostic program which is designed to detect the presence of a fault and indicate in some fashion where it is. The simpler programs for this purpose generally suffer from one or more of the following shortcomings: lack of thoroughness, low fault resolution, or a large hardcore.[1] The more sophisticated approaches have to some extent overcome these difficulties, but have often involved special hardware at extra cost and required lengthy and expensive development cycles, particularly for the more complex machines.

Until fairly recently, diagnostic programs have been written in assembly language because this was the closest that the diagnostic programmer could get to the hardware. With the advent of microprogrammed processors and the availability of READ–WRITE control stores, it has become apparent that better results could be achieved by writing diagnostics at the microprogram level [1], [2]. It is the purpose of this note to describe the microdiagnostic technique used with the Standard Computer MLP-900 CPU and some of the features of this processor that greatly facilitate this approach.

GENERAL CHARACTERISTICS OF THE MLP-900 CPU

The Inner Computer Concept

One of the distinguishing features of the computer systems manufactured by Standard Computer Corporation is the concept of the computer within a computer [3]. Within this context the MLP-900 processor may be thought of as an inner computer, complete with its own instruction repertoire, program memory (i.e., control memory), and even its own interrupt system. These instructions require one hardware clock cycle to execute, and henceforth to avoid confusion will be called ministeps. Programs written with these ministeps will be called MINIFLOW® programs to distinguish them from higher level programs. The control memory may be READ–WRITE which provides complete flexibility in writing emulators, interpreters, or other MINIFLOW programs such as microdiagnostics.

The outer computer consists of the MLP-900 processor, with MINIFLOW programs for up to four languages, a main core memory, and suitable I/O equipment. Its instruction repertoire can be any conventional machine language such as 940, 360, etc., or a completely new language that, for example, might be especially designed to interpret problem-oriented instructions [4]–[6].

The inner computer may be operated in at least two different ways. One mode would be similar to the way conventional computers are operated now. That is, MINIFLOW programs to perform a number of data processing functions could be stored in main memory. Each one could be first overlayed in control memory, under control of a primitive operating system, then executed, and the results outputted. This process would then be repeated for each subsequent job or job segment. Another way to operate the inner computer might be called the subroutine mode. In this case, higher level instructions (i.e., higher than ministeps) would be accessed from main memory and used to call a MINIFLOW subroutine, resident in control memory, to perform the higher level task. The latter, of course, is the mode which would be used for emulation.

General Organization and Operation

The MLP-900 processor consists of three major functional parts: the operating engine (OE), the control engine (CE), and the control memory (CM). The OE is essentially an arithmetic unit capable of 36- and 8-bit data manipulations. It contains 32 general registers whose contents may be used directly in arithmetic and logical operations, and up to 1024 auxiliary registers that may be used to store constants, tables, etc. Thirty-two mask registers are provided to facilitate partial word or disconnected field operations. Also included are up to four pairs of external bus interface registers that may be used to communicate with external devices such as main memory, I/O controllers, etc. A pair of instruction registers is provided to retain the target level instruction when operating in the emulation mode. The CE is basically a program control unit which controls ministep sequencing and maintains a wide range of inner and outer computer status information. The latter is retained in 256 state flip-flops which may be tested for branching purposes. Also included in the CE are 16 8-bit pointer registers that may be used to indirectly address OE registers or for counting purposes such as loop iterations. The CM stores the MINIFLOW programs in two independent sections which, respectively, contain the even and odd addressed locations, thus allowing a pair of 32-bit ministeps to be accessed simultaneously. Ministep addressing is provided for up to 64K of control memory, although current packaging density permits only 4K to be implemented. This may be either READ–ONLY or READ–WRITE (RW) in any proportion, but will probably be largely RW in most installations.

In operation the OE and CE may be thought of as two independent processors, each with their own ministep repertoire. If the two sequential ministeps that are accessed at each clock are an OE type followed by a CE type, then both will be executed simultaneously. Otherwise only the first ministep is executed.

[1] Hardcore is used here to mean the part of the machine that must work for the diagnostic to be loaded and run.

® MINIFLOW is a trademark of Standard Computer Corporation.

Functional Capabilities

The MLP-900 is capable of executing 103 basically different ministeps, 70 of these being OE types and 33 CE types. This does not include minor variations that are possible through various control bits. All ministeps are executed in one clock cycle, although in a few cases the cycle is expanded to two clock periods to allow certain logic signals sufficient time to settle. A vertical form of microprogramming is used instead of the more common horizontal approach. With this technique, individual bits or fields in the ministep are not dedicated to control specific microoperations in the processor, such as Shift Left 4, but are used in a more general way to indicate the type of operation to be done and to specify other information required to perform the operation, such as operand source register addresses. This results in a more highly encoded ministep (32 bits plus parity) that is shorter and thus makes more efficient use of control memory but with some loss in performance. Because of the short encoded format, the resulting microprogramming language is very similar to most common assembly languages and thus is much easier to program than a microprogram language based on the horizontal approach.

The ministeps executed by the OE are categorized into seven classes which are labeled by the mnemonics GEAR, CEDE, SHIN, CHAR, CHAL, GENT, and TEXT. The word format shown in Fig. 1 for the GEAR class is typical of the others.

The functional capabilities of each class of OE ministep are described briefly as follows.

GEAR (GEneral ARithmetic): This class consists of 16 ministeps that allow 36-bit masked arithmetic and logical operations to be performed on operands stored in the general registers.

CEDE (Conditional External Data Exchange): This class consists of 13 ministeps that allow various kinds of external bus operations to be initiated and terminated. Their primary use is for controlling the flow of data between main memory and the MLP-900 processor.

SHIN (SHift INstruction): This class of 11 ministeps permits various kinds of shift operations to be performed on operands stored in the general registers. It includes special one-step operations to assist in performing multiply, divide, and normalization.

CHAR (CHAracter Right): This class of 14 ministeps allows various decimal arithmetic, binary arithmetic, and logical operations to be performed on 8-bit byte pairs located in the general registers. Four bytes are located in the rightmost 32 bits of each word and provision is made to allow automatic byte scanning from left to right.

CHAL (CHAracter Left): This class is functionally identical to the CHAR class except that byte scanning is done from right to left.

GENT (GENeral Transfer): This class of two ministeps provides a means for moving data between register groups within the OE, such as from the general registers to the auxiliary registers.

TEXT (Transfer EXTernal): This class of ministep is similar to the CEDE class but will be used to control the flow of data between peripheral controllers and the MLP-900 processor.

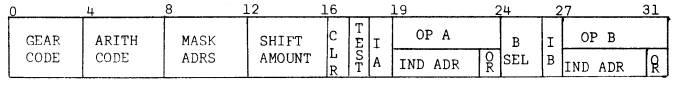

Fig. 1. Word format for the GEAR class of ministep.

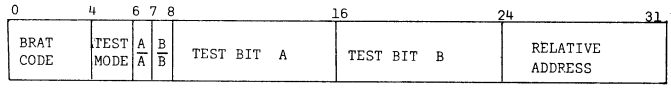

Fig. 2. Word format for the BRAT class of ministep.

The ministeps executed by the CE are categorized into eight classes that are labeled by the mnemonics BRAT, BENT, BORE, BRAD, BEAD, BLOT, MAST, and MOVE. The word format shown in Fig. 2 for the BRAT ministep is generally typical for the others. The functional capabilities of each class of CE ministep are described briefly as follows.

BRAT (BRAnch and Test): This class of four ministeps allows a miniflow branch to be made to a relative address depending on the Boolean value of two State F/F's.

BENT (Branch and ENTer): This class of four ministeps is identical to the BRAT class except that if the branch condition is true, the current continuation address is also stored into a subroutine stack.

BORE (Branch Or REturn): This class of four ministeps is identical to the BRAT class except that if the branch condition is false, a return occurs to the address located at the top of the subroutine stack.

BRAD (BRAnch and moDify): This ministep allows a relative miniflow branch to be made depending on the value of one State F/F and a pointer register to be decremented or incremented.

BEAD (Branch with Extended ADdress): This class of four ministeps allows absolute, relative, or indexed (i.e., using a pointer) branches to be made based on 16-bit addresses. These are useful for global branching and table entry functions.

BLOT (BLOck Transfer): This class of six ministeps facilitates the movement of blocks of data within the processor and into and out of the processor. Data, of course, may be ministeps.

MAST (MAnipulate STatus): This class of four ministeps allows various kinds of logical manipulations to be performed on State F/F's.

MOVE (Move): This class of six ministeps permits various types of 8- and 16-bit data transfers to be made within the CE.

In addition to the programming facilities, the MLP-900 includes several other special functional capabilities that have relevance to the microdiagnostic. These are described briefly as follows.

Action Requests: These are analogous to program interrupts except that they are active at the MINIFLOW level. There are 32 action requests and they are set by the occurrence of various types of machine errors, exceptional conditions, and external signals. The highest priority 16 action requests, if not masked or locked out, cause an immediate

breakout from the ministep sequence. A subroutine entry occurs and control is transferred to a dedicated location in CM. The lowest priority 16 action requests operate similarly, only their breakout is allowed only on certain types of CEDE ministeps.

Language Boards: These are special logic cards which may be added to the MLP-900 to perform certain functions that are unique to a particular language. The motivation for including them is to achieve highly efficient emulation. The language boards can assist in achieving this objective by performing field extraction from target level instructions, address modification for accessing operands from main memory, and several other language-dependent minor functions. Language boards for up to four different languages may be present in the machine at one time and may be selected on a dynamic basis.

MINIFLOW Status Word (MSW): This word is like the program status word that is often found in third generation machines, except that it is effective at the MINIFLOW level. Its functions include selecting the active language board, the operating mode, the external bus to be used for main memory communication, and the masking of action requests. Part of the MSW is a 16-bit address field that makes it possible to initialize the processor and transfer MINIFLOW control after a CM load operation.

Hard Loader: A short MINIFLOW load routine is incorporated in a small dedicated area of READ–ONLY CM. This routine makes use of the BLOT ministep and a special load control word (LCW) that is associated with each block of 255 words or less to be loaded. The LCW is always accessed first and indicates how many words are to be loaded, where they are to be loaded (i.e., CM or other processor registers), and whether another block load is to be done after the current one is completed. Loading is done from main memory and may be initiated manually or by MINIFLOW program.

Diagnostic Aids

A rather extensive set of diagnostic aids is available to the service engineer in the event of a processor fault. At the manual level, as a backup to the microdiagnostic, there are two single-clock modes and over 400 light-emitting diode indicators for monitoring the processor state. Additional facilities include two single-step modes at the program level; a breakpoint feature that permits processor stops to be made on main memory, control memory addresses, or on a processor error; a repeat facility which allows MINIFLOW looping without requiring the MINIFLOW program to be modified; and a scope synchronizing feature that provides a sync pulse based on an arbitrary CM or main memory address. Additionally, built-in facilities allow either CM or main memory to be tested without test programs.

One of the outstanding characteristics of the MLP-900 design is the degree to which it facilitates diagnostic programming. Features which provide this capability include a READ–WRITE control memory, one-clock ministeps, a general-purpose repertoire, nearly full MINIFLOW access to the hardware, and the ability to specify test operands as immediate values. The READ–WRITE control memory, of course, is essential if diagnostics at the MINIFLOW level are to be used at all. The one-clock ministeps are important because they eliminate the sequential network analysis problem, thus making test design easier and also contributing to higher fault resolution. The general-purpose repertoire provides a great deal of flexibility in devising efficient tests and makes programming easier, which contributes to a shorter development cycle. Having full MINIFLOW access to the hardware is, of course, essential to achieving thoroughness, and greatly facilitates test design by avoiding awkward indirect test procedures. Finally, the ability to specify test operands, in both the OE and CE, as immediate values makes test setup very convenient and efficient. Those readers interested in more details are referred to [7] and [8].

MLP-900 Microdiagnostic Description

Design Objectives

The MLP-900 microdiagnostic was developed to serve a number of different, although related, purposes. The initial application was to facilitate the checkout of the prototype machine. Later on in the machine life cycle it will serve as an acceptance test, a readiness test, and a fault location tool. In order to serve these purposes, the microdiagnostic has been designed to exercise all functional areas of the machine with the objective of first detecting whether any fault exists and, if one does, of providing the service engineer with enough information to allow him to quickly isolate it. In order to maximize the possibility that the microdiagnostic will provide useful fault location information, it has been designed with a carefully selected sequence of bootstrapping tests. Our primary design goals thus have been thoroughness, high resolution, and minimum hardcore. Some secondary goals have been to design efficient tests that avoid an excessively large MINIFLOW program and to design relatively simple, straightforward tests that would be understandable to those using them for troubleshooting.

Organization

The MLP-900 microdiagnostic consists of 22 small test modules, organized into five sections, as shown in Table I. The test modules were selected so as to test a natural functional area and to fit within a 512-word bank of control memory. The latter has several advantages including complete applicability to any machine configuration and a better chance for the microdiagnostic to run successfully and provide a bootstrapping sequence. A number of test options, such as looping on individual modules or on the entire microdiagnostic, are provided through sense switches available on the maintenance panel.

Operation

Operation of the microdiagnostic assumes that the test modules, along with appropriate load control words, have been previously loaded into main memory. A special CM load mode which is selectable on the maintenance panel allows manual initiation of microdiagnostic loading into the processor using the hard loader. This process can include not only the loading of CM with ministeps, but the

TABLE I
MAJOR MICRODIAGNOSTIC SECTIONS

Section	Test Area
1	Control Engine Ministeps and Facilities
2	Operating Engine Ministeps and Facilities
3	Special Functions
4	Language Boards
5	Test Control and Stop

loading of constants into general or auxiliary registers, mask patterns into mask registers, or the loading of a MINIFLOW status word. The latter would normally be done after all the ministeps and constants for a test module had been loaded, and would initialize the processor for the test and transfer control to it. Execution of the test module then begins and continues until a fault is detected, which causes a MINIFLOW halt to occur (if selected). If no faults are detected, then execution continues on to the end of the module where a branch ministep transfers control back to the loader, which then loads the next module, and the whole process is repeated over again. Loading and execution of modules continues, assuming no faults are detected, until the last module—test control and stop—is loaded. Depending on sense switch settings, the processor will then either stop, loop on the entire microdiagnostic, or initiate execution of a functional level diagnostic.

General Discussion

Our experience in developing the microdiagnostics for the MLP-900 has led to the following general observations. The first is that it is a significant advantage to be able to write diagnostics at the MINIFLOW (i.e., microprogram) level and a second is that a processor like the MLP-900 greatly facilitates the task of developing a fault isolation diagnostic. One of the most striking advantages is the ease of designing and writing component-oriented tests. This comes about because of the one-clock repertoire of ministeps and the ability to get at all areas of the processor for testing. This results in a microdiagnostic that is very compact (about 6K ministeps), particularly for a processor as complex as the MLP-900 (over 20 000 equivalent gates). This kind of efficiency leads to low-cost development (about 12 man-months effort) and short development time (about 6 months). The running time for the entire microdiagnostic is only about 6 ms, which is relatively fast.

Another advantage of the ability to write the diagnostic at the miniflow level is the resolution with which a fault can be isolated. The one-clock repertoire coupled with the ability to get at and test detail sections of the logic make it possible to isolate down to the gate level in most cases. These features make it unnecessary to add special hardware to achieve high resolution, thus saving the cost of this hardware as well as avoiding the need to test it.

One limitation of the microdiagnostic approach described here is that it cannot replace a functional diagnostic if the inner computer is emulating a higher level language (i.e., it cannot test the emulation MINIFLOW program). Obviously, a higher level diagnostic or test program is required here to properly test functional operation of the emulated machine, including IO, etc.

Because of the vertical form of microprogramming used in the MLP-900, the MINIFLOW language is very similar to any assembly language. As a consequence, no special software support is required to develop the microdiagnostic. All that is needed is an assembler, which is also used for developing emulators, and a loader for getting the MINIFLOW program into main memory. The former, called ICAP II [9], provides the usual type of assembly listing.

CONCLUSIONS

The results achieved in developing a microdiagnostic for the MLP-900 processor indicate that the ability to write diagnostic programs at the MINIFLOW level can be a very effective and powerful technique. In general, it can permit better results to be achieved in less time and at lower cost. However, it is worth noting that other machine features, such as a general-purpose ministep repertoire and the facility of having almost complete MINIFLOW access to the hardware, contribute as much or more towards achieving an efficient and effective diagnostic. The best results are thus achieved where microdiagnostics are used in conjunction with a machine architecture oriented for diagnosibility. It is the conclusion of the author that microdiagnostics, where applicable, represent a significant advance in the art of computer diagnostic techniques and an important addition to the tools of the maintenance engineer.

APPENDIX

Given below are two programming examples which illustrate the simplicity of the assembly language and the ease with which hardware tests can be coded.

Example 1—Operating Engine Adder Test: The following test loads two test operands into general registers 1 and 2, adds them, compares the result with that expected, and halts the processor if the answer is wrong.

Operator	Operands	Comments
MVT	GO1, !X7DF7DF7DF	Move a long immediate value of 7DF7DF7DF into general register 1.
MVT	GO2, !x041041041	Move a long immediate value of 041041041 into general register 2.
ADD	GO1, GO2	Add the contents of general registers 1 and 2 and put the result in general register 1.
XOR	GO1, !x820820820	Compare result with expected value and set zero result flag if equal.
MAUNB	ZRF1. ERS, CKC	Move the logical OR of the zero result flag and the complement of the error stop switch into the clock control flip-flop (i.e., turn off the clock if the result was not zero and error stop is selected).

Example 2—Control Engine Pointer Zero Sense Test: The following test loads pointer register 1 with a value of

zero and halts the processor if the pointer zero sense signal is not true.

Operator	Operands	Comments
MSI	0, P01	Move a short immediate value of 0 into pointer register 1.
MAUNB	ZSI01, ERS, CKC	Move the logical OR of the pointer 1 zero sense signal and the complement of the error stop switch into the clock control flip-flop (i.e., turn off the clock if the zero sense signal is false and error stop is selected).

Acknowledgment

The author would like to thank B. Smith for his advice on programming aspects of the MLP-900 microdiagnostics and for programming many of the modules.

References

[1] R. C. Calhoun, "Diagnostics at the microprogramming level," *Modern Data*, May 1969, p. 58.
[2] N. Bartow and R. McGuire, "System/360 model 85 microdiagnostics," in *1970 Spring Joint Comput. Conf. AFIPS Conf. Proc.*, vol. 36. Montvale, N. J.: AFIPS Press, 1970.
[3] L. Rakoczi, "The computer-within-a-computer, A fourth generation concept," *Comput. Group News*, Mar. 1969, pp. 14–20.
[4] A. Melbourne and J. Pugmire, "A small computer for the direct processing of FORTRAN statements," *Comput. J.*, vol. 8, Apr. 1965, pp. 24–28.
[5] H. Weber, "A microprogrammed implementation of Euler on IBM system/360 model 30," *Commun. Ass. Comput. Mach.*, Sept. 1967, pp. 549–558.
[6] H. W. Lawson, Jr., "Programming—language—oriented instruction streams," *IEEE Trans. Comput.*, vol. C-17, May 1968, pp. 476–485.
[7] H. W. Lawson, Jr., and B. K. Smith, "Functional characteristics of a multilingual processor," this issue, pp. 732–742.
[8] ——, *MLP-900 Multi-Lingual Processor, Principles of Operation*, Standard Computer Corp., Santa Ana, Calif., Form 809001-6.
[9] ——, *ICAP II Language, Programmer's Manual*, Standard Computer Corp., Santa Ana, Calif., Form 801027.

Microprogramming and Numerical Analysis

BRUCE D. SHRIVER, SR.

Abstract—This short note presents the view that microprogramming affords a technology that can provide a powerful, efficient, and versatile computer facility for numerical analysts. Particular areas of application are discussed. Fundamental questions are raised relevant to the exploitation of this technology by numerical analysts.

Index Terms—Algorithm implementation, matrix manipulation, numerical analysis, numeric efficiency, storage.

I. Introduction

For the purpose of this discussion let us define numerical analysis as the discipline that provides algorithmic processes for the numerical (arithmetic) solution of mathematical problems. Such processes are provided to a wide spectrum of users, from those who have little or no background in mathematics to those who themselves are engaged in numerical analysis research.

Each class of users expects performance criteria to be met when using an implementation of a particular numerical method on a digital computer. There have been recent discussions about such criteria [1] and methods to certify whether such criteria are met Cody [2]. Aside from the obvious concern that the implementation of the algorithm provide either accurate and meaningful results[1] or an indication of the lack thereof, most users are indeed concerned with the efficiency (cost) of the implementation, not only in terms of low execution time, but also with respect to the storage requirements for both the program code and the data.

The need to design production run programs, to implement the algorithm within the capabilities of an existing computer configuration and language processors, and to live within time and dollar constraints of existing funding are important factors to be considered. "Efficiency" will herein connote the time–cost–accuracy–availability, etc., tradeoffs involved.

II. Basic Questions About Algorithm Implementation

Given the fact that we wish to implement algorithms on a digital machine, several basic questions arise.

1) What structure should a digital computer system have for the efficient processing of a wide class of numerical algorithms? Should it be a special, multi-, or general-purpose machine? What degree of concurrency can it, or should it, possess? Should it process instructions sequentially or have a highly parallel organization?

2) What should be the primitive data types of this machine: bit, bytes, words, integers, floating-point numbers, matrices, etc.?

3) What primitive machine operations should there be which *access* and *manipulate* the primitive data types? What type and base arithmetic should be available?

These basic questions are said to concern the "environment" (structure, primitive data types, and operations) within which an algorithm is implemented. This note presents the view that microprogramming affords a technology whereby the environment for processing a given class of numerical algorithms can be efficiently realized.

It is noted that partial answers to the above and related questions are often reflected in the form of the programming language and operating system by which the user addresses the machine. If a language, such as Fortran, suits his purpose, he essentially can look at the computer system as a Fortran machine. If his needs go deeper, he may

Manuscript received August 15, 1970; revised December 16, 1970.

B. D. Shriver, Sr., is with the Department of Computer Science, State University of New York at Buffalo, Amherst, N. Y. 14226.

[1] As an example, to the user of "black-box" procedures (application packages), this means such items as conditioning, stability, rounding error properties, and *a posteriori* error bounds should be built into the procedure.