INTRODUCTION

User-microprogrammable computers have been generally available since the early 1970s, although in the past few years they have become quite popular. The primary reason for the increased popularity is the decrease in price made possible by technological advances in high-speed memories. Also, the computer manufacturing industry is looking towards microprogramming for increasing throughput of large operating systems, which cannot be replaced because of the large investment for them in software.

Our experiences with a microprogramming or emulation laboratory have come from two points of view—research and instruction. The research has explored several areas including emulation of computers, microprogramming languages and emulation of language machines. The main emphasis in instructional aspects is to illustrate the problems of microprogramming and the difference between it and conventional programming by practical examples programmed in our emulation laboratory.

To provide some insight into our use of the laboratory, a short history of its development will be given. This is followed by our experience with the laboratory in both instructional and research applications.

DEVELOPMENT OF THE EMULATION LABORATORY

In 1971 the Department of Computing Science laid the groundwork for acquiring a minicomputer laboratory for undergraduate instruction. This laboratory was specified to have many diverse computers including one with microprogramming capability. The Microdata 1600 was selected, since it was small and cheap and fit in well with the rest of the laboratory. The machine was purchased in 1972 with 8K bytes of mainstore, 256 16-bit words of writable control store (WCS), and an ASR Teletype as an I/O device. Subsequently, another 512 words of WCS have been added, and an interface to a digital (eight-bit parallel) cassette has been built.

When the department was formulating the mini-lab it was envisaged that a medium sized machine microprogrammed to emulate all of the minicomputers would be acquired. A special grant was obtained in the spring of 1973, and the QM-I was selected in the fall of that year; the only other choice was the Burroughs B1700 which was rejected for reasons of cost, restrictions on usage and the fact that it did not support interrupt-driven I/O. The department ordered the QM-I initially with 32K of mainstore, 3K of control store, 256 words of nanostore, and support peripherals including tapes, disks, a CRT console terminal, a line printer and a card reader. Over the last five years the system has grown to that illustrated in Figure 1.

In late 1973 a Varian V73 was acquired. Although that computer was intended mainly for use in the mini-lab it was purchased with 512 words of WCS, 8K 16-bit words of mainstore and an ASR 33. Since then a dual floppy disk unit has been added and software support developed in-house. The V73 has been used in the minicomputer class with good success and, since the floppies were added, has been utilized in the microprogramming course.

TEACHING WITH THE EMULATION LABORATORY

The Computing Science Department offers a graduate-level course (CMPT 512—Advanced Minicomputer Systems) which deals primarily with microprogramming. Typically, 10 to 15 students enroll in the course, which covers the structure of emulators and the topics outlined in the text, along with a brief introduction to the current microprogramming research being done in this department as well as at other sites. In the following discussion the method of using the machines will be mentioned, as well as the learning aspects gained from programming each machine.

Several assignments to teach the fundamental theories and practices of microprogramming have been developed. The main criterion for microprogramming assignments is an algorithm which references mainstore, employs several conditional branches and is familiar, repetitive and computationally simple with easy to check results. Several examples of basic assignments are producing a count of the number of one bits from locations 'A' to 'B' in mainstore; generating a parity bit for a word; and the assignment referred to in the following discussion, sorting into ascending order mainstore...
from ‘A’ to ‘B.’ The parameters ‘A’ and ‘B’ were to be entered at run-time, as operands for a ‘sort’ opcode, or in the case of the QM-l, read in from the console terminal. The assignment illustrates the following concepts associated with microprogramming—mainstore accessing, parameter passing, and complex condition testing.

Microdata 1600

The Microdata 1600 has a vertical micro-instruction word of width 16. Internally the busses are eight bits wide, and there are 16 internal file registers. To minimize the complexity (workload) of the assignment, the students were not
required to program the Microdata to print the results, but rather a system utility was invoked to verify the correctness of the results produced by their sorter.

The scenario, used by students to accomplish the assignment, was to edit and cross-assemble the source on the university's service computer, punch the binary (usually quite small) onto paper tape and load it into the Microdata using the utility program AROS. They would then proceed to debug the program by inserting halts (or using the address-compare stop feature of the 1600 front panel) at critical points, single-stepping through small sections, correcting the results and re-editing the source. There are a few important points in the previous technique which should be noticed. A service computer was used to generate the binaries, even though an assembler exists on the Microdata. This is because it utilizes the various components as they were intended, the Microdata to microprogram, and the service computer for software development. Others have also found a service computer helpful in increasing throughput on a microprogrammed computer. The second point concerns the usage of the front panel for debugging microcode. Even though a simulator exists it is not only awkward to use, as are most simulators, but also slow to load from paper tape—which can be circumvented through new cassette support. With the exceptionally good front panel support given to micro-debugging, the students did not find the simulator worthwhile.

Besides the basic microprogramming concepts just discussed, the assignment illustrated the following problems—multi-precision arithmetic as 16-bit words were sorted; the largest negative number causes complex condition testing in the sort; simple parallelism as mainstore fetches/stores can be overlapped with other processing; and simple timing problems (the 'U' register is finicky).

In addition to describing the general architectural features of the Microdata, the lectures included a discussion of I/O and used a teletype echo microprogram as an example. Concurrent I/O (a poor man's DMA) was discussed in relation to the small amount of hardware required to implement this relatively powerful concept (similar to the IBM 360/50 channel implementation).

As a machine for teaching elementary microprogramming the Microdata 1600 is very good. The format of its instruction set is quite close to that of conventional machines (as are most vertical microprogrammable machines), it is a general purpose machine and yet it illustrates some of the elementary problems in microprogramming.

**Varian V73**

The Varian V73 has a horizontal micro-instruction word of width 64. The fields which make up this word have up to four levels of encoding. The branching capability is very general, although ordering control store words to take good advantage of the branching is quite complex. The assignment was, once again, to sort 16-bit words of memory between specified (variable) limits.

Most students prepared source tapes offline on a service computer with an editor, file system, and other useful utilities such as cross-reference programs. The sources were then transferred to the Varian where they were read from paper tape with the micro-assembler, MIDAS. A modified version of Micro-util was then used to load, execute and debug the binaries. As there is very little front panel support for microprogram debugging, the debug portion was quite trying for the students. The front panel is an I/O device which must be supported in microcode before any internal registers may be displayed. On the Microdata quite the opposite is true; the front panel is an intelligent piece of hardware requiring no software support. Varian intended microprograms to be debugged from an attached processor which could step, trace, etc., the V73 micro machine.

Notice that once again the students used service machines to develop the source code. Most likely the reason for not using the program preparation facilities on the Varian is because they are awkward to use (paper-tape-based). This could have been rectified by adding several thousand dollars worth of main store, disks, tapes and printers to the V73 and using Varian's operating system (VORTEX or MOS), but then the system would be so large that it would not be cost-effective to allow individual students hands-on experience. The assembler supplied by Varian is quite primitive (the service computer's editor helped the students use mnemonics), and a more powerful one has been developed elsewhere.

The Varian illustrates the multi-way branch (for opcode decoding), horizontal microprogramming, and the difficulties of using conventional program design methods for designing microprograms. Also the independent I/O control store and processor are a unique feature discussed in the lectures.

**Nanodata QM-1**

The QM-1 offers both vertical (control store) and horizontal (nanostore) microprogramming. The higher-level control store is a fully readable and writable general-purpose memory whose word width is 18 bits. Eighteen bits is also the width of mainstore, the internal registers, and the ALU and shifter (although there is also a 16-bit mode). The lower level nanostore is a fully writable, executable memory whose word length is 360 bits. Two 72-bit fields are activated from the five possible) during each 75 nsec machine cycle. The architectural organization is very parallel, allowing several functional units to perform simultaneously. The QM-1 was designed as an emulation tool, and as such has no native (or most efficient) instruction set. Support software executes on a NOVA emulation, and although reasonable peripherals are attached to the QM-1, the operating system is quite primitive by today's standards.

Since NCS (Nanodata Control System) only supports one terminal it is not time/cost-effective to allow online entry of student programs. Therefore, once again, a service computer was used to produce the source statements for the assemble, read them onto disk, assemble, load and debug them on the QM-1. We have found offline source preparation
so useful that a high-speed interface to a time-sharing PDP 11/45 is being built.

In the class assignment on the QM-1, the students were to program the sort algorithm in nanocode, and output the resulting sorted array onto the console terminal. The I/O portion was microcoded in the MULTI\(^*\) micro-instruction set. The combination of these elements taught the students the interfaces between the three major stores in the QM-1, and the parallelism and flexibility of the QM-1.

The QM-1 performs very well as a teaching tool. The very horizontal architecture (and parallelism) of the nanomachine illustrate two major theoretical research areas in microprogramming—optimization theory and high-level microprogramming languages. Also, the two-level micro-instruction structure provides a methodology for generating efficient emulators (nanocoding the instruction set for run-time efficiency and microcoding the I/O and console functions for programmer efficiency).

**EMULATION RESEARCH**

Although three microprogrammable computers are available for emulation research, the QM-1 has been used for most research projects. Quite likely the preference for the Nanodata machine is a reflection of its flexibility as a universal host and the fact that it has a disk file system. The emulation research performed at the University of Alberta can be subdivided into three major areas—emulation of hardware computers (conventional instruction sets), tools for developing emulations (microprogramming languages) and language machine emulations. The current status or results of these projects are summarized in the following sections: the reader is referred to the papers referenced for details.

**Conventional machine emulations**

A PDP 11/10 emulator has been constructed and evaluated for the QM-1.\(^9\) This emulator would be a class ‘A’ emulator, as defined by Flynn;\(^9\) except that it does not check for odd PC values, or handle the trace trap bit in the PS. These features were not included in the emulation only because the implementor did not feel that their usefulness outweighed the overhead involved in the nanoprograms. The PDP-11 emulator successfully executes standard instruction diagnostics, memory, tape and disk exercisers, and also the DOS-11 and MINI-UNIX operating systems. No changes were made to these programs; the MINI-UNIX operating system ran successfully the day it arrived.

After the PDP-11 emulation was thoroughly debugged and MINI-UNIX was obtained, a profiling feature was added to instrument operating system efficiency studies. This program counter-sampling allows one to find where a system is spending the majority of its time. By nanocoding a few identified functions, we have reduced the execution time of a benchmark from over 16 minutes to under six minutes. Further studies with this profiling mechanism need to be done to tune the emulation to an operating system.

Multiple concurrent emulations, sharing a common micro-coded I/O section, have been investigated\(^11\) on the QM-1. The QM-1 has been found suitable as a host for multiple emulations, but several problems are yet to be resolved. Device-sharing is the major problem, especially with devices such as magnetic tapes—how does the emulator control program determine when an emulator is finished with the tape? The micro-operations passed to the emulated controller are too small to determine the intentions of the emulated system. (Open and close calls would be required to give the ECP enough information to know when it may allocate the drive to another emulator.) Successful experiments have been performed with dual Nova emulators with a manual task switch facility, sharing the console terminal, disk and clock.

**Microprogramming languages**

Research has been done on the design and implementation of high-level microprogramming languages for both the QM-1 micro-instruction set (MULTI) and the lower-level nanoprograms. A nano-level language presents difficult problems to the language designer, as illustrated in the Lizard language.\(^12\) After examining the problems of building efficient nanocode, the researcher concluded that an automatic translator would not be cost-effective compared to human nanocoders. Although this result is somewhat discouraging and shows that theory and practice are sometimes disjoint, we have not terminated our research in this area and have achieved better results at the micro-level.

CQ,\(^13\) a high-level microprogramming language based on the programming language C (produced at Bell Telephone Laboratories), produces code in a slightly extended MULTI instruction set. The compiler (including a MULTI assembler and linkeditor) for CQ is being developed on a time-sharing PDP-11 using the UNIX operating system. It is believed that once this system is operational it will provide much better software development tools than the current QM-1 system has to offer.

**Language machines**

Language machine research within the department is being done at two levels—high-level language machines, and intermediate-level language machines. APL is the target language for the high-level language machines. A general plan for implementation has been drawn up,\(^14\) and the indexing portion has been coded\(^15\) on the QM-1. Also, research is being performed on the multi-user scheduling portion of the system.

Intermediate-level languages are those languages which fall between high-level languages (FORTRAN, Pascal, APL, COBOL, . . . ) and conventional assembler language. These languages are intermediate in both syntax and semantics. An example would be the Pascal 'P' machine for which a microcoded interpreter has been written on the QM-1 by UCSD. Our research group has formulated a methodology...
Teaching and Research Experiences with an Emulation Laboratory

for evaluating intermediate language machines (machines which interpret/emulate intermediate-level languages). To date the methodology has only been shown feasible by modeling on a service computer; actual experiments will be performed on the QM-1.

CONCLUSION

Throughout the previous discussion the notion of using a service computer to support an emulation machine frequently appears. This is quite reasonable when the computers are thought of as tools used to build an emulator. The two functions, development and execution, could be combined on one computer (as Nanodata has done on the QM-1), but this leads to inefficient or primitive development tools, or to host machines which are not universal (for example the Varian).

Having surveyed instructional and research usage of an emulation laboratory our experience indicates that a suitably supported universal host provides an excellent vehicle for exploring many areas. An important area is the relationship between high-level languages (or algorithms written in them) and the instruction sets (architectural machines) which these languages are translated into (or interpreted by).

REFERENCES
