The Real-Time/Minicomputer Laboratory

Michael A. Malcolm
Gary R. Sager

Department of Computer Science
University of Waterloo

Research Report CS-76-11

February 1976

ABSTRACT

The Real-time/Minicomputer Laboratory at the University of Waterloo has several interesting projects underway which pertain to the educational and research uses of minicomputers.

cross software

Several pieces of cross software have been developed at the University of Waterloo to aid in the development of software for minicomputers in the Lab. These include an assembler, relocating linking loader and a library file editor; all are written in Fortran and are used on our Honeywell 6060 to develop software for Data General NOVA minicomputers. We are currently working on more extensive portable cross software to support other minis in the Lab.

real time systems course

The major use of the Lab to date has been for teaching an undergraduate course in real time computing. Students receive "hands on" experience with computer control applications using Laboratory equipment. This involves the use of a computer-controlled model train and slot car race track. Students develop programs on the Honeywell using the cross software, then transfer them to the NOVA via a high-speed communication link for execution and debugging. The cross software allows a class of 30 to 40 students to share one minicomputer for assignments.

"Hub" system for minicomputer support

In order to support a large number of diverse and possibly minimally configured minicomputers, we are developing a star network of minicomputers. The central node, called the *Hub*, will serve to:

a) handle communications protocols for file transfers to and from large computer systems
b) communicate files to and from other minis using simple protocols
c) provide peripheral sharing among the Lab's minicomputers.

The Hub will greatly increase the value of the Laboratory for education and research by:

a) encouraging extensive use of cross software for program development, thereby reducing the "hands on" time required by every user of the Lab
b) providing access to data and devices of many machines
c) significantly reducing the investment required to add machines to the Laboratory.

portable systems software

The Laboratory currently supports research being conducted by several graduate students and faculty members. This research has concentrated on the development of a portable high-level language called "Eh" and a real-time operating system written in Eh which will eventually execute on all machines in the Lab. This will enhance the exchange of software within the Lab, and will provide greater flexibility in the use of the machines.

1. Introduction

The Real-Time/Minicomputer Laboratory in the University of Waterloo's Computer Science Department serves a purpose similar to that of a chemistry or physics laboratory by giving access to a diverse array of minicomputers and peripherals. It is used primarily as a teaching facility for introducing undergraduate students to some practical aspects of computer science. Students are exposed to "hands-on experience" using inexpensive minicomputer equipment to solve various systems programming and real-time applications problems.

During the past year, students of a course titled *real-time applications of minicomputers* have made extensive use of the lab. The lab has also been used for the research of graduate students and faculty members. Hopefully, in another year, facilities of this laboratory can be extended to serve other courses in computer science, such as *computer architecture.*

The lab has been in use since Winter 1974. We started with a borrowed Honeywell 316 which has since been returned to Honeywell. In May 1974 we took delivery of a Data General NOVA/2 having 8K words of core and a cassette unit. By January 1975 the NOVA cross software was working sufficiently well to support a class of 20 students. Since then, we have greatly increased the NOVA software support; this is discussed in Section 2. In May 1975 we acquired a Microdata 1600/30; since then we have developed cross software which supports a high-level language for it. this work is part of the portable software project discussed in Section 5.

Perhaps the most interesting addition to the lab came last summer when an electric train was interfaced to the NOVA. It is used as a source of real-time control problems for students of the real-time applications course. Recently a slot car track has also been added to the lab for the same purpose. These unusual computer peripherals will be discussed in Section 3.

2. Cross software

Both the NOVA and Microdata come with vendor-supplied software such as assemblers, editors, operating systems, etc. This software provides tools for further software development; however, most of it assumes a single-user hands-on environment during editing, assembling, loading and debugging. Thus, it is difficult to support more than 5 or 6 students developing non-trivial programs using only vendor-supplied software, not to mention the inconvenience of using paper tapes and cassettes for storing files and loading programs. We have been able to support over 30 students per term on one machine (the NOVA) using cross software running on the Honeywell 6060 timesharing system (TSS).

Most of the cross software is written in portable Fortran, including the assembler, relocatable loader, library file editor and emulator for the NOVA, and a micro-assembler and microprocessor simulator for the Microdata. The loader, library file editor and emulator for the NOVA were written locally. The assemblers and the microprocessor simulator were supplied by the vendors; however, the NOVA assembler was debugged and substantially extended locally. We have also developed a NOVA program called the *NOVA/TSS Communication System* (NTCS) which allows a NOVA user to sign onto TSS and transfer files between the two computers. Normally, this is used to transfer a core-image file to a NOVA cassette for subsequent execution.

The basic idea of the cross software is to take advantage of the strong points of the machines involved. We are taking advantage of the interactive multiprogramming system of the Honeywell to allow many students to be simultaneously active editing, assembling and loading programs as well as using the emulator to run some preliminary debugging tests. The Honeywell provides a flexible file system with good protection and backup, thereby obviating the necessity of this level of support on the NOVA itself. The offline support of program storage and development has been useful for systems work, such as maintaining NTCS, as well as student assignments. The Honeywell also provides a high speed line printer for assembly listings, load maps, and core dumps, plus extensive performance reports from the emulator. In a sense, the Honeywell 6060 serves as a sophisticated peripheral of the NOVA.

The NOVA, on the other hand, is used mainly for students to gain hands-on experience with computers, especially in the areas of machine-level debugging and real-time applications. In these environments, the multiuser and file system capabilities would be difficult to provide (especially at any level approaching the quality of the Honeywell). With the cross software, it is possible to support up to 30 to 40 advanced students while still providing each student with ample time as sole user of the NOVA.

An additional advantage is gained by the relocatable loader; it is possible to load programs on the Honeywell using the cross loader, which cannot be loaded on the NOVA due to the core re-

quired for the loader to run on the NOVA; 721 words are used by the second pass of the NOVA relocatable loader.

The connection between the Honeywell and NOVA is through a synchronous communication line running at 19,200 bits per second. This bidirectional file transfer capability proved useful for detecting and isolating bugs in the cross software as well as assuring its compatibility with the Data General NOVA versions. We have also made use of the ability to take a core dump on the NOVA and transfer it to the Honeywell for a post mortem dump analysis. written in a high-level language to run on the Honeywell.

3. real time systems course

A course in *Real-Time Applications of Minicomputers* has been developed at the University of Waterloo during the past year by one of the authors (Malcolm). The course is given at the fourth-year undergraduate and first-year graduate level. A major motivation for developing such a course is to expose students to hands-on minicomputer experience. It also provides a computing environment and a class of problems which few of the students have experienced.

The course is partially hardware oriented, but the main emphasis is software and system design. The students learn in detail how a particular minicomputer works, especially the I/O interface hardware. Most of the programming is done at the machine language and assembly level. Students write a stand-alone program for a real-time application which requires interrupt handling. They also get experience using a modern real-time operating system.

An important part of the course is learning how to structure a complicated program as a set of cooperating sequential processes, or tasks. Students must write and debug such a program utilizing a real-time operating system. A rich source of problems for this exercise is provided by an electric train which was interfaced to our NOVA in Spring of 1975.

Many of the problems encountered when writing a program to control the train are the classical problems of operating systems. For example, since there are two independently controlled trains using the same track, some sort of track allocation algorithm is usally devised to keep them from running into each other. This, of course, involves critical sections, not unlike those of core allocation algorithms. Many students have found that their programming is simplified and the reliability of their programs is enhanced by using a finite automaton to control the train; the problem is then reduced to that of writing the the state-transition tables. Also, there are a number of deadlock situations which can occur. A sophisticated program can avoid some of the deadlocks, and recover from others, by the same methods used in operating systems.

We also study the organization of simple real-time operating systems themselves. Students read parts of the system they are using, and we discuss how to construct one from scratch.

In short, the Laboratory provides an environment for teaching which could not be provided on a large multi-programmed machine. The electric train is a fun and interesting source of programming problems. The problems of critical sections and deadlock are easy to visualize when they occur on a train track, and the importance of their proper solution becomes obvious. For most students, it is the first time in their programming career that no large mysteries are going on inside the computer. They have a detailed understanding of the operating system and the hardware on which their programs run.

Hands-on use of the machine also adds an element of realism which comes as a small shock to some of the students. Students must turn the machine on and off themselves. Sometimes the proper cable isn't plugged in or a terminal is set to the the wrong baud rate. Or an interface isn't functioning properly, etc. Students learn to become somewhat skeptical of the hardware when debugging a program; and they learn to account for possible (likely?) malfunctions of sensors and control circuits.

Recently we have added a slot car set to the lab. It uses two channels of analog input to control the speeds of the two cars, and provides encoded sensor numbers and interrupts whenever one of the 64 phototransistor sensors is crossed. The sensor data is provided in such a way that two minicomputers can race each other. No programs have been written for the slot cars yet, but we expect them to provide more severe real-time constraints than the train.

4. "Hub" system for minicomputer support

In Section 1 we mentioned that cross software (which executes on the Honeywell 6060) had been developed for the Microdata minicomputer. This implies that programs must be transmitted to the Microdata. Since NOVA software already existed

for moving files from the Honeywell, and the Microdata requires a TWX teletype for loading programs (which we didn't have at the time, and didn't wish to purchase), we decided to use the NOVA as a host machine for moving software from the Honeywell to the Microdata. To do the communication from the NOVA to the Microdata, we developed a program which makes the NOVA emulate a teletype. The emulated teletype paper tape reader and punch are in actuality the NOVA cassettes. The Microdata teletype interface line speed was increased to 4800 bits per second; thus program loading is considerably faster than with a real teletype. The disadvantage of this technique is that it ties up the NOVA whenever someone wishes to use the Microdata. This arrangement has been adaquate for the software development phase in which only 1 or 2 hours of debugging time was required per day. However, now that the compiler for the Microdata is working and the machine is gaining users, a more economic means of loading programs (i.e., simulating a teletype) must be found.

As a result of the experience with the Microdata, and our anticipation of similar experiences in the future, we are currently developing a *Hub* computer which will serve as a "file transfer machine". The Hub will be connected to the Honeywell 6060, the Math Faculty's PDP-11/45 UNIX timesharing system, and to each of the other minicomputers in the Lab. A Texas Instruments 990/10 with a small disc will be used for the Hub. It will primarily be used for moving files between the two time sharing systems (Honeywell TSS and UNIX) and the attached minicomputers. It will also provide an ability to share minicomputer peripherals between the machines in the Lab.

There are several advantages to the central file system, including a major savings in the reduced duplication of peripheral equipment, and reduced duplication of communications software. Another important advantage of designing a minicomputer laboratory around a Hub machine is that having fewer peripheral devices per minicomputer results in less maintenance per minicomputer; this is a major consideration for the person in charge of the Lab. The intentionally simple protocols of the Hub will allow it to serve as a bootstrap (initial program loading) device for attached mini and microcomputers. Hence, it will be possible to add minimally configured computers to the lab in the future. This will be expecially valuable if we decide to add minicomputers for undergraduate students to use in computer architecture courses. Thus, for example, first or second year students could edit and assemble PDP-11 programs using UNIX or TSS, and later transmit them (using the Hub) to a stripped PDP-11 (costing under $5000 each) for a stand-alone debugging session. Experience with the NOVA indicates that this arrangement can yield an order of magnitude increase in the hands-on service rendered by a minicomputer.

Obviously the Hub, being the central node in a star network, must be extremely reliable. The minimally-configured machines attached to the Hub will be essentially useless during Hub down times. We hope to attain high reliability of the Hub hardware through redundancy of hardware components: Another project in the Faculty of Mathematics will require a number of Texas Instruments 990 machines; hence we can justify a stock of spare parts.

One of the machines which will be attached to the Hub is the PHOTON phototypesetter which produced this document. (It is controlled by another Microdata minicomputer.)

5. Portable systems software

The Hub network described in the previous section will provide an excellent environment for research into portable minicomputer software. We are currently engaged in the design of portable systems software for minicomputers. This project involves implementing a portable set of systems software and porting it to each of the machines in the Lab as well as to the Honeywell and UNIX. The universal availability of the systems software will not only reduce the amount of software Lab users will need to become familiar with, it will also allow them to use the software as cross-software on any of the machines.

As the first stage of the project, we have designed a high-level implementation language (called Eh) which will be common to all machines. While a high-level language does not guarantee portability, the design of the language can discourage or prevent portable programming; therefore, the prime design criterion for Eh has been to insure that it will permit and encourage portable programming. To a large extent, this simply requires that the semantics of the language be well defined. Thus, detailed documentation of the language is necessary [2]. In addition, we are developing a set of certification programs which are designed to detect incorrect implementation of the semantics of Eh; once the certification programs are successfully executed, the portable systems software should execute correctly.

The Eh compiler consists of two phases: the first phase does the syntax analysis and outputs intermediate language. The intermediate language

resembles the order code for a stack oriented machine. During the second phase of compilation, the intermediate language is translated to the target machine language.

Porting Eh to a new machine involves a rewrite of the second phase of the compiler. In this respect, the portability of the Eh compiler is much like that of many other portable compilers. However, there is one important difference: Many compilers output assembler code, then rely on an existing assembler and linking/relocating loader for the final transformation into an executable program. Other compilers generate output suitable for input to the linking/relocating loader. In either case, the portability of the compiler depends upon the capabilities of the loaders found on various machines. Were we to assume a "least common denominator" of assemblers and/or loaders available on minicomputers, we would be forced to assume so little that we could not implement software of reasonable complexity. Reliance on existing assemblers and/or loaders has been a weak point for many portable systems.

Our approach has been to implement a machine-independent loader in Eh. This "universal loader", called *ULD*, accepts a machine description along with relocatable load modules output by the Eh compiler to create executable programs. Thus, ULD will execute on and load for all machines in the Lab (as well as the Honeywell and UNIX).

We must also implement an assembler for each machine to interface machine dependent code with programs written in Eh. Although this may appear to be more work than can be justified, some observations should be made: Since we do not rely heavily on the assembler, it need not be efficient or powerful. We have designed and implemented a table-driven portable assembler called *TLA* which can be easily modified to assemble code for a different machine [7]. Converting TLA to generate code for a new machine consists of creating a table of opcodes and rewriting six small subroutines (around 200 lines of Eh) which specify the semantics of the opcode classes. TLA has already been converted to assemble code for the Microdata 1600/30, the Data General NOVA 2, and the TI 990/4. Many other machines were studied while TLA was being designed; in all cases, TLA can match the vendor's assembler syntax closely, the major departure being the pseudo-ops, which have been standardized across all versions of TLA. The output of the assembler is load code for ULD.

Since the second phase of the compiler outputs load code for ULD as well, it is able to link to TLA assembly code. We have tested ULD by loading programs for the Honeywell 6060, Data General NOVA 2 and Microdata 1600/30. ULD has executed on the Honeywell 6060 and Microdata 1600/30; load code for the Honeywell was generated by hand, that for the NOVA was generated by TLA, while load code for the Microdata was generated by compilations of Eh and TLA programs.

The internal structure of the three machines we have loaded for shows a great deal of variation:

(1) Honeywell 6060: 36 bit word, word addressing, 18 bit address, the address may appear in the left or right half of the word

(2) NOVA: 16 bit word, word addressing, 8 or 15 bit address

(3) Microdata: 16 bit word, byte addressing, 8 or 15 bit address, variable-length instructions

ULD is designed to search libraries of load code. This requires tools to build and maintain libraries. To this end, we have designed and implemented a Library Editor which will maintain libraries of ULD input on any machine for any machine [3]. It is written in Eh, and can be ported with the other components of the system with virtually no change.

Finally, to provide an environment for writing real-time applications programs, we have designed and implemented a small real-time executive in Eh, called *Thoth* [6]. Thoth provides multitasking capabilities such as dynamic scheduling, task creation and deletion, inter-task communication and synchronization of tasks. It has been designed to provide a very small yet effective set of primitives. A minimally configured version of Thoth involves about 600 lines of Eh (including comments) and around 100 lines of TLA assembler code, depending on the machine. On a 16-bit minicomputer, such a configuration of Thoth should require between 2K and 3K words of storage, based on experience with the Microdata. This compact system will be a useful tool for implementing Lab support software. In addition, it will serve as a vehicle for teaching basic concepts for such courses as real-time applications and operating systems.

6. Reflections

The motivation for setting up the Real-Time/Minicomputer Lab has been to provide a diversity of minicomputers and peripherals and the appropriate cross software and communication links to make them useful for teaching reasonable numbers of students and conducting a modest amount of research. We hope eventually to have around 6 different mainframes in the Lab. In this sense we have only begun to achieve our original objectives.

Having a variety of machines has distinct pedagogical advantages, and it provides stimulation for certain kinds of research (particularly software portability research). However, we wish to warn prospective builders of such laboratories that the multi-vendor objective has some disadvantages. For example, in order to acquire such a variety of machines, one must deal with a variety of computer salesmen. The variety of different machines leads to a variety of different manuals, diagnostic programs, operating systems, programming languages, etc. Each machine requires a different preventive maintenance program. One generally has to deal with a different organization for each different machine when it needs repair. And it usually isn't practical to stock spare parts for one-of-a-kind minicomputers or to invest in training local repair personnel in the intricacies of each different machine. Custom-built equipment, such as our model train and slot car interfaces, can add to the maintenance difficulties. Our design of the Hub system discussed in Section 4 was partially motivated by the potential savings in hardware maintenance.

The diversity of vendor-supplied software presents a far more severe problem than that of hardware maintence. The number of manuals, listings, paper tapes, and bugs to keep track of can be overwhelming in itself. This often frightens away potential users of the Lab. This is compounded by the fact that many of the more interesting minicomputers are intended for the OEM market or are very new and have little or no software support. We also find that we sometimes wish to run the same applications program on more than one machine; but portability of vendor-supplied software is practically impossible. Our portability project has been inspired by the variety (as well as the low quality of some) of the vendor-supplied software. Certainly, in this case, necessity has been the mother of invention. We hope that our portable systems software project will provide solutions to many of these problems.

In spite of the difficulties, we believe that it is highly desirable to have a variety of machines in the Lab. It is the diverse array of equipment which is continually combined in new and interesting ways that provides a rich and stimulating environment.

7. Acknowledgements

We are pleased to acknowledge the help of several students, faculty and staff members in the creation and development of the real-time/minicomputer laboratory. We especially wish to thank Gord Agnew, Rick Beach, Reinaldo Braga, Randy Bruce, Ernie Chang, Nancy Kemp, Rick Madter, Eric Manning, Lawrie Melen, Jim Morris, Laurie Rogers, Gerhardt Roth, Gary Stafford, and Fred Young. In addition, many of the students who have taken Math 479a have helped in debugging and designing the cross software.

We gratefully acknowledge the National Research Council of Canada, and the Waterloo Research Institute, for their support, in part, of the research discussed in this paper.

8. References

[1] Braga, R., M. A. Malcolm and G. R. Sager, "A design for a portable programming system", Department of Computer Science, University of Waterloo, Research Report CS-75-29, November 1975.

[2] Braga, R., "Description of the programming language Eh", unpublished manuscript, 1976.

[3] Bruce, R. D., "Library editor reference manual", unpublished manuscript, 1976.

[4] Lennon, William J., "A minicomputer network for support of real-time research", *ACM 1974 Annual Conference*, San Diego, California, November 1974.

[5] Malcolm, M. A. and G. R. Sager, "Report on real-time/minicomputer laboratory", Department of Computer Science, University of Waterloo, Research Report CS-75-23, September 1975.

[6] Melen, L., "A portable real-time executive", unpublished manuscript, 1976.

[6] Sager, G. R., "Emulation for system measurement/debugging", in J. R. Bell

and C. G. Bell, *Minicomputer Software*, North-Holland, to appear 1976.

[7] Stafford, G. J., "The Last Assembler reference manual", unpublished manuscript, 1976.