A COMPARISON OF THE MONADS II AND III COMPUTER SYSTEMS[§]

J Leslie Keedy, David Abramson, John Rosenberg
& David M Rowe

Department of Computer Science
Monash University
CLAYTON VIC 3168

MONADS II AND III are two research computer systems which aim to provide a uniform architecture. Whereas MONADS II is an enhanced HP2100A computer, MONADS III is a new computer being built from "off the shelf" components. After a review of the common architectural aims, the paper discusses their differences, in terms both of efficiency and flexibility and of their implementation of the architecture. It is then explained how the two systems will be used in a single system configuration. Finally the method of achieving software compatibility is described.

§ See also other papers by these authors individually.

1. Introduction

The basic aim of the MONADS project is to investigate improved methods for developing and implementing large software systems (Keedy, 1981). In a companion paper (Keedy, 1982a) it has been shown that many benefits can be derived from rigorously applying the information-hiding principle to the decomposition of systems into modules. However, one of the difficulties which currently stands in the way of this approach is that existing computer architectures do not provide the basic support necessary to achieve this in an efficient and straightforward manner. Hence the MONADS team has in recent years devoted considerable effort to defining a new and more suitable computer architecture (Keedy and Rosenberg, 1982) and to the development of computer systems designed to implement it.

The most important features of the target architecture are:

(i) a large uniform segmented and paged virtual memory (Abramson, 1981; Rosenberg and Keedy, 1981a);

(ii) capability-based protection both for the segments of individual modules and for inter-module calls;

(iii) an addressing environment flexible enough to distinguish the different kinds of data segments required by modules (Keedy, 1982b), including block-structure and dynamic data structures (Gehringer, Keedy and Thomson, 1982);

(iv) process stacks large enough to hold all the local data for all the modules which a process calls;

(v) inter-module call and return mechanisms;

(vi) synchronisation mechanisms suitable for in-process systems (Keedy, Ramamohanarao and Rosenberg,1979; Keedy, Rosenberg and Ramamohanarao, 1982)

An important design goal for the computer systems developed to support this architecture was to achieve a good balance between flexibility and efficiency. Flexibility is important in an experimental environment and can largely be achieved through the generous provision of microcoding facilities. However, efficiency dictates that the most common and fundamental operations be implemented directly in hardware. The considerations which governed our choice of a balance are discussed in a companion paper (Abramson, 1982b). The basic solution was to provide direct hardware support for address translation and memory protection, leaving the implementation of the instruction set and support for modules as microcoded functions.

Initially our resources for developing new computer systems were very limited and in 1979 Abramson undertook a series of extensions to a Hewlett Packard HP2100A, which has become known as the MONADS II system (Abramson, 1981,1982a). Because certain undesirable features of the HP2100A could not be changed, MONADS II was considered from the start to be a pilot system with two important roles. First, it was a testbed for implementing a new address translation unit capable of supporting capability-based addressing for a large uniform virtual memory. Second, it provided an implementation of the new architecture sufficiently powerful to serve as a development base for the new operating system.

In 1981 funds became available to design and build an entirely new computer, MONADS III, without the restrictive features of the MONADS II system. In this paper we examine in some detail the essential features of both systems and compare the two. We also describe how both have a part to play in the final MONADS configuration, and we discuss how a single operating system and supporting software can provide compatibility between them.

2. Flexibility and Efficiency

As indicated above an important goal of both MONADS systems was to achieve a good balance between flexibility and efficiency. Three techniques contributed to this goal in varying degrees for MONADS II and MONADS III.

First, both systems make extensive use of microcoding, which greatly improves efficiency compared with software and on the other hand gives much greater flexibility than hardware. We recognise two levels in the architectural model. Real hardware is used to implement the most fundamental and frequently used operations, in particular address translation, which takes place for each instruction and operand decode. In this respect the two systems are similar. The second level of the architecture is concerned primarily with the definition of the instruction set, both in terms of basic instructions and system management instructions. Ideally in both systems the appropriate implementation would be a complete microcoding of the MONADS III instruction set (Rosenberg, 1982). This is fully realised in MONADS III, but two problems arise with MONADS II. First, for various reasons we retained the HP2100A basic instruction set (although operands for instructions are reinterpreted in terms of the MONADS addressing scheme in a way which is invisible to the HP2100A processor: Abramson, 1982a). Second, and more important, although we were able to extend the writeable control store (WCS) from the initial HP2100A limit of 512 words to 3584 words, this proved to be insufficient to allow the system management instructions to be fully microcoded.

To overcome the shortage of WCS in MONADS II in a manner which would be invisible to the rest of the software we used our second technique. This was the development of a new hardware kernel, implemented in a combination of microcode and software, using the principles which had already been proved in the MONADS I system (Rosenberg and Keedy, 1978; Rosenberg, 1979). The effect of course is that MONADS II is much slower than MONADS III in this respect. Support for modules (Keedy, 1982a) is in fact largely implemented in software, although the limited WCS available was used to implement key aspects of the most frequently used of these operations (e.g. the call instruction).

The third technique for improving efficiency could only be realised in a significant way in the MONADS III system. This involved the use of distributed processing units to relieve the main MONADS III processor of auxiliary functions which could be performed in parallel. In particular, all memory management activity (page fault handling, management of address spaces, etc.) is relegated to a Z8000 microprocessor, and a second (unmodified) HP2100A, known as the "minder", performs four other functions (bootstrapping, operator control, fault diagnosis and communication with the departmental VAX 11/780) (Rosenberg, Rowe and Keedy, 1982; Rowe, 1982).

3. Architectural Aspects

While the two MONADS systems implement a common architecture, except in terms of the basic instruction set, there are some significant differences in detail which limit the functionality of MONADS II.

Perhaps the most significant difference is the size of the virtual memory. MONADS III supports a virtual space of 2^32 address spaces, each with a potential length of 2^27 bytes. MONADS II, on the other hand, is restricted to 2^16 address spaces each with a potential length of 2^16 bytes. The importance of these differences becomes clear when it is recalled that there is no provision for a separate filestore and that virtual addresses are used to guarantee the uniqueness of names in capabilities (Abramson, 1982b; Keedy, 1982b). In effect the number of address spaces limits the number of stacks, files and code modules which can exist throughout the lifetime of the system. With 2^32 address spaces MONADS III will not find this restrictive, even if it has a lifetime of say 20 years. But with 2^16 address spaces, MONADS II is unlikely to survive for more than about a year or two, if used continuously as an independent system. In view of the objective of MONADS II to serve as a pilot system this should not turn out to be a serious problem. Should this prove to be an error of judgement, however, two solutions could be adopted. One would be to develop a garbage collector which retrieves unused and unreferenced old address space numbers for reuse. The other alternative, which we are seriously considering at present, is to extend the address translation unit and capability registers to support 2^32 address spaces.

The size of an address space limits the size of a single segment (or stack or heap) in MONADS II to 64K bytes. While this is not unreasonable for a pilot system, it would be restrictive in an environment intended to support serious database or artificial intelligence applications. The problem can be overcome by forcing the software to decompose its logical segments into 64 K byte sub-segments, but the solution is not elegant. Unfortunately the nature of the HP2100A precludes the possibility of removing this restriction with hardware modifications. On the other hand MONADS III allows any segment, stack or heap to have a maximum length of 128 Mbytes, which is considerably larger than in any other existing computer.

Another architectural difference is the actual implementation of capability registers. Whereas MONADS III provides a homogeneous set of capability registers, it was convenient in modifying the HP2100A for MONADS II to implement, in addition to the 16 standard capability registers, 6 special registers (three related to stack addressing, one to code addressing, one to the addressing of constants and other code-related data, and one related to the HP2100A base leaf). Fortunately this affects only the MONADS II code generator and the hardware kernel.

In MONADS II we adopted the approach used in MONADS I (Wallace, 1978) of implementing 16 sets of capability registers. This has the advantage that 16 processes can be concurrently active with very efficient process switches. In practice, however, this limits the system to 16 active users, unless a complex register set allocation scheme is implemented in the hardware kernel. For MONADS II this was not considered a restriction, because it was never planned to support more than 16 users. This idea was abandoned in MONADS III because we did not want to limit the number of active users and because inter-module calls (which also require switching of capability registers) are expected to occur much more often than process switches. Consequently MONADS III has only one set of capability registers, which are switched both on inter-module calls and process switches.

The limitations of using the HP2100A instruction set inevitably constrain the efficiency of program execution on MONADS II, compared with MONADS III. For example literal offsets in instructions are limited to three bits, when used in combination with the standard capability registers, and unlike MONADS III, where a single operand can combine literal offsets

and index registers to achieve fast indexing for arrays of records (Rosenberg, 1982), MONADS II does not support such complex addressing modes.

In other respects, for example the structure of capability lists and the organisation of control information in address spaces, the two systems are fully compatible.

4. Configuration

The above mentioned points make clear that MONADS III is intended as a general purpose system which will be useful for serious workloads. MONADS II on the other hand has built-in restrictions affecting the number and size of address spaces, the number of active users, etc. Also, the hardware and instruction set of MONADS III are designed to support faster processing and larger workloads. However, there is one important omission in the MONADS III processor: it makes no provision for input-output operations to external peripherals (e.g. line printers). The reason for this is that the MONADS III processor is not designed as a stand alone processor. In line with the philosophy of distributing processing activity among a series of processors, the intention is to provide a separate input-output processor which operates concurrently with the main processor. Given the existence of MONADS II, with its peripheral handling capability inherited largely from the basic HP2100A, it is planned to use this as the I/O processor for MONADS III (Rosenberg, Rowe and Keedy, 1982; Rowe 1982).

5. Software for the MONADS Systems

Although the MONADS II and MONADS III systems provide the same basic target architecture in terms of virtual memory organisation, process stack organisation, support for modules, etc., they clearly do not have a level of compatibility which would allow compiled code from one system to be executed directly on the other. This is evident from the fact that they have quite different basic instruction sets, different basic data formats, different address space sizes, etc. Yet it would clearly have been counterproductive to develop quite separate software systems for the two, and indeed this would have defeated the objective of using MONADS II as a development base for MONADS III software.

To ensure that the differences in software for the two systems were kept to a minimum, we have used the information-hiding principle once again, this time to hide the hardware differences from most modules. This involved limiting knowledge of specific hardware features to two modules, the kernel and the compiler code generator (Keedy and Rosenberg, 1981; Rosenberg and Keedy, 1981b).

As mentioned in section 2, compatibility for the system management instructions is achieved through the implementation of a hardware kernel in MONADS II, which provides the same functionality as the system management instructions of MONADS III.

The kernel interface, however, is concerned only with operating system aspects of the architecture. The basic instruction sets of the two systems and associated features such as registers, data formats, etc. are also quite different in some respects, largely because of reliance on HP2100A features in MONADS II. Compatibility at this level is achieved through the use of high level languages (at present Pascal and a Modula 2-like language). However, to avoid re-implementing complete compilers for the two systems, a single intermediate code interface has been defined. All compilers produce code in this intermediate form and separate code generators are used to

compile from this into the appropriate machine code for MONADS II and III. Again, the code generator for MONADS II is a substantial piece of software, but because the MONADS III instruction set definition is close to the intermediate language definition (Rosenberg, 1982) its code generator will be very simple and small. Hence, the task of moving software from MONADS II to MONADS III will be achieved simply by developing new and much smaller hardware kernel and code generator software for MONADS III and recompiling the operating system and any other modules developed initially for MONADS II. Likewise it will be possible to transfer MONADS III software back to MONADS II.

ACKNOWLEDGEMENTS. The authors acknowledge with thanks the work of other members of the MONADS team who have contributed to the design and implementation of the MONADS II and MONADS III hardware and software, especially Peter Dawson, Mark Evered, Mark Halpern, Ed. Gehringer, Glenda Patterson, Kotigiri Ramamohanarao, Ian Richards, John Thomson, Brian Wallis and John Wells.

We also gratefully acknowledge the funding provided by the Australian Research Grants Scheme (grants F77/15337 and F80/15191), by the Monash Special Research Fund (grant SC18/79) and by the Australian Computer Research Board (grant SB3/11).

REFERENCES

Abramson, D. (1981) "Hardware Management of a Large Virtual Memory", Proc. 4th Australian Computer Science Conference, Brisbane (Australian Computer Science Communications 3, 1, pp. 1-13).

Abramson, D. (1982a) "A Technique for Enhancing Processor Architecture", Proc. 5th Australian Computer Science Conference, Perth (Australian Computer Science Communications 4, 1, pp. 47-57).

Abramson, D. (1982b) "Hardware for Capability Based Addressing", Proc. 9th Australian Computer Conference, Hobart.

Gehringer, E.F., Keedy, J.L. and Thomson, J.V. (1982) "Dynamic Data Structure Management in MONADS III", Proc. 5th Australian Computer Science Conference, Perth,pp 68-78.

Keedy, J.L. (1981) "A Progress Report on the MONADS Project" Australian Computer Science Communications, 3, 2, pp. 270-277.

Keedy, J.L. (1982a) "The MONADS View of Software Modules", Proc. 9th Australian Computer Conference, Hobart.

Keedy, J.L. (1982b) "Support for Information-Hiding Modules in the MONADS Architecture" (submitted for publication).

Keedy, J.L., Ramamohanarao, K. and Rosenberg, J. (1979) "On Implementing Semaphores with Sets", The Computer Journal, 22, 2, pp.146-150.

Keedy, J.L. and Rosenberg, J. (1981) "Information Hiding - A Key to Successful Software Engineering", Proc. Conference on Computers in Engineering, 1981, Institution of Engineers, Australia,

Publication No. 81/8, pp. 1-5.

Keedy, J.L. and Rosenberg, J. (1982) "The MONADS III Computer Design: A System to Support Software Engineering" (submitted for publication).

Keedy, J.L., Rosenberg, J. and Ramamohanarao, K. (1982) "On Synchronising Readers and Writers with Semaphores", The Computer Journal, pp. 121-125.

Rosenberg, J. (1979) "The Concept of a Hardware Kernel and its Implementation on a Minicomputer", Ph.D. Thesis, Dept. of Computer Science, Monash University.

Rosenberg, J. (1982) "The MONADS Series III Instruction Set", Proc. 9th Australian Computer Conference, Hobart.

Rosenberg, J. and Keedy, J.L. (1978) "The MONADS Hardware Kernel", Proc. 8th Australian Computer Conference, Canberra, pp. 1542-1552.

Rosenberg, J. and Keedy, J.L. (1981a) "Software Management of a Large Virtual Memory", Proc. 4th Australian Computer Science Conference, Brisbane,pp. 173-181.

Rosenberg, J. and Keedy, J.L. (1981b) "Information Hiding - A Case Study", Proc. Conference on Computers in Engineering, 1981, Institution of Engineers, Australia, Publication No. 81/8, pp. 6-9.

Rosenberg, J., Rowe, D.M. and Keedy, J.L. (1982) "An Overview of the MONADS Series III Architecture", Proc. 5th Australian Computer Science Conference, Perth,pp 58-67.

Rowe, D.M. (1982) "MONADS III Inter-Unit Communication", Proc. 9th Australian Computer Conference, Hobart.

Wallace, C.S. (1978) "Memory and Addressing Extensions to a HP2100A" Proc. 8th Australian Computer Conference, pp. 1796-1811, Canberra.

COMPUTER-BASED MANAGEMENT AND DELIVERY OF SPECIAL EDUCATION SERVICES

Mike Lally & Iain Macleod

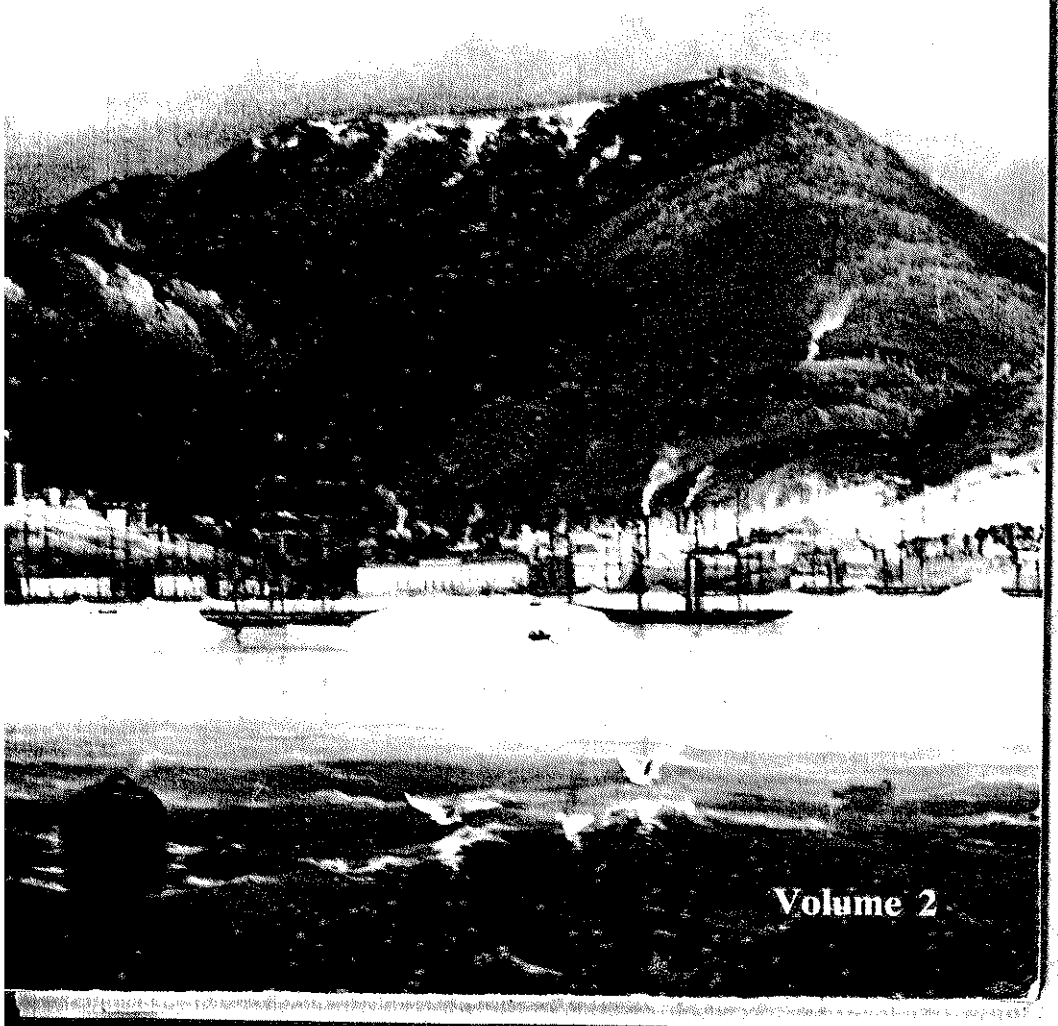
Department of Engineering Physics
Research School of Physical Sciences
The Australian National University
PO Box 4
CANBERRA ACT 2600

Many students require specialised educational services to achieve to their potential. Factors which may isolate students from appropriate services include cultural background, geographical remoteness and intellectual characteristics. Competence in reading, handwriting and simple number concepts is required for much subsequent learning. Timely intervention with students having difficulty with basic skills is, therefore, especially important.

Computer-assisted instruction in basic skills is proving to be very effective. Together with current microprocessor and data communications technology, this type of instruction enables implementation of a computer network which facilitates management and delivery of specialised educational services at the point of need. This network promotes co-operation between specialist and classroom teachers in catering for individual students.

Proceedings of the

Ninth Australian Computer Conference



Volume 2