Supporting a Capability-Based Architecture with Silicon

D. Abramson and J. Rosenberg
Department of Computer Science, Monash University, Clayton, VIC

Capability-based addressing was first proposed in 1966, yet very few computers which use the scheme have actually been produced. This can be largely attributed to the complex hardware required to achieve acceptable performance. This paper presents the design of a VLSI chip to support capability style address computations.

1. INTRODUCTION

Capability based addressing was first proposed in 1966 by Dennis and Van Horn [Dennis & Van Horn, 1966] as a technique for structuring and protecting information in the memory of a computer. It is appealing because it solves many of the problems of sharing data and provides a uniform mechanism for controlling access to information [Fabry, 1974].

Several capability based machines have been constructed both in the research environment [England, 1972; Wulf, et al, 1974; Needham, 1977] and commercially [I.B.M., 1978; Intel, 1981]. Unfortunately they have not become popular. One of the main reasons appears to be that they require a higher level of hardware support than conventional Von Neumann machines in order to achieve the same efficiency.

The new technology of custom made VLSI chips poses the question of whether an efficient and cost effective implementation of a capability based processor is now possible. This paper discusses some relevant issues and describes a particular design.

2. CAPABILITY-BASED ADDRESSING IN THE MONADS ARCHITECTURE

The MONADS project began in 1976 at Monash University and has the primary aim of investigating techniques for the development of large and complex software systems. The main technique involves the decomposition of such systems into a number of small information hiding modules [Parnas, 1972; Keedy, 1982]. The technique requires that each module can only access its own data and that interfaces between modules are purely procedural. It is also necessary to allow multiple instances of the same type of module and this requires sharing of code. Capability based addressing supports both of these requirements. This section describes the particular model used in the MONADS architecture.

The MONADS architecture has a uniform virtual memory which holds both computational and permanent (file) data. All data is addressed by a large virtual address consisting of two parts, an address space number and an offset, as shown in Figure 1. Each address space contains data which is related in some way (e.g. the procedures of a module, segments of a file) and the offset identifies a particular byte within the address space. MONADS virtual addresses are capabilities in that they are unique and are never reused.

Each address space may be divided into many segments and these are defined by segment lists. Behind the segmentation scheme is a paged

virtual memory, however, there is no direct relationship between segments and pages. The details of this scheme are described elsewhere [Keedy, 1980] and are not relevant to this paper. All that is important is that each segment list entry describes a region of virtual memory (Figure 2). The base within the segment list entry defines the start of the segment and the limit defines the end. Executing programs can only access segments for which they have a segment list entry. For efficiency reasons segment list entries are never used directly, rather their contents are loaded into special capability registers before use. Thus instructions need only name the capability register in order to refer to a segment of data. Access to individual bytes requires the specification of an additional offset within the segment. This can be contained within the instruction together with the capability register number. Access to arrays and other dynamic data structures requires that the offset within segment be calculated at run time. This is implemented by an indexed addressing mode which causes the contents of an index register to be combined with the offset within the instruction. Thus the generation of an effective virtual address involves the addition of the base within the capability register to the offset within the instruction and the contents of the index register. This is then compared with the limit value in the capability register before the memory reference can proceed. In order to support both word and byte addressing the index register is optionally scaled before the addition.

The addressing scheme described above is very powerful and flexible. It will support access to an individual element within an array of records in a single instruction, and can also be used for stack addressing and program instruction fetching. Several machines implementing this scheme have been proposed and constructed at Monash University. These include MONADS II, MONADS II/2 and MONADS III [Abramson, 1982; Rosenberg, et al, 1982; Rosenberg, 1982; Rowe, 1982; Keedy, et al, 1982].

The latest phase of the project is the design and construction of a small workstation, called MONADS-PC, which supports the MONADS architecture [Rosenberg & Abramson, 1985]. It is invisaged that a number of these workstations would be connected via a high speed local area network [Abramson & Keedy, 1985].

3. IMPLEMENTATION OF MONADS-PC

The MONADS-PC computer comprises three main subsystems as shown in Figure 3. The memory and I/O subsystem is implemented using standard MULTIBUS [Intel, 1978] and off-the-shelf memory and device controllers and is of no concern to this paper. The address translation subsystem is responsible for translating the large virtual addresses (60 bits on MONADS-PC, composed of a 32 bit address space number and a 28 bit offset) into main memory addresses and includes a cache to reduce memory traffic. This section is also not relevant to this paper. The main processor subsystem is responsible for interpreting and executing machine instructions. It is the main processor which must perform the address calculations described above and finally generate a virtual address. In the following sections the current implementation is described and an alternative faster solution is discussed.

3.1. Current Implementation

The main processor consists of a microprogrammed control unit and a regular three bus structure arithmetic unit. The basic cycle time is 175 nanoseconds. The buses are all 32 bits in width and connect the registers and ALU as shown in Figure 4. The 16 capability registers are held in the dual ported RAM file and there are two separate 32 bit memory address

registers to hold the virtual address. There are three index registers for combination with any of the capability registers and these are also held in the dual ported RAM. The offset within segment may be directly accessed from the instruction register, as can the capability register number and the index register number. One input of the ALU can be scaled to convert a word offset into a byte address.

There are special capability registers for stack and program instruction fetching which are held in the dual ported RAM. These are used in a similar manner to the normal capability registers.

The following control sequence is required in order to generate a virtual address:

(1) capability_register.address_space -> memory_address_space_register

(2) capability_register.base + instruction_register.offset -> hold_register

(3) scale(index_register) + hold_register -> memory_offset_register

(4) if (memory_offset_register - capability_register.limit) > 0
    cause an error
    else
    perform memory operation.

Thus four machine cycles are required for every memory reference instruction. This same sequence must be performed each time an instruction is fetched. There are a number of notable attributes of this scheme.

(1) There is no special hardware to support the address calculation. The ALU and a register file would be required for a conventional machine.

(2) The first and second step could have been overlapped had there been a wider data pathway.

(3) The two addition operations could have been performed faster had the three operands been available in one cycle. Moreover a better addition algorithm could have avoided two carry propagation times.

The next section considers a faster implementation which uses direct hardware support for the address calculation.

3.2. A Faster Implementation

Figure 5 shows a block diagram of the proposed address computation unit. The unit would connect to the general data pathways shown previously and the registers could be loaded and retrieved via the normal data buses. The hardware is split into two sections, one which generates the address space number (which requires no computation, and is not shown in the Figure) and the other to calculate the offset within address space. Clearly the first consists only of a small RAM file indexed by capability register number. For clarity only two bits of the offset calculation hardware are shown.

Corresponding bits of the capability_register.base, the index_register and the instruction_register.offset are combined by a normal adder circuit, using the carry input as the third operand (this technique was first proposed by Wallace for the implementation of fast multiplication [Wallace, 1964]). This stage requires no carry propagation. The sum and carry outputs of these adders are then combined by a carry propagation adder to produce the final virtual address offset. The limit is then subtracted from the offset to form an error signal. (A faster solution would involve using an additional three level tree for

adding/subtracting the base, index, offset and limit to generate the error signal in parallel with the virtual address offset computation. This would avoid the subtraction propagation time.)

The advantage of this scheme is that it could perform the full address calculation in about one machine cycle. The scheme was not used in the MONADS-PC processor because of the amount of extra hardware required over the central processor. It was estimated that it would require about 35 TTL chips. (The parallel scheme would require many more chips.) The regular structure and simple internal components suggest that a VLSI implementation would be appropriate.

4. A SILICON IMPLEMENTATION

4.1. General Structure

The address calculator chip is connected to the main processor as shown in Figure 6. The address space number is simply read from the address space RAM. The offset is loaded into the address calculator as soon as it is available and the within address space offset is computed. Because the offset is 28 bits in length the input and output lines on the chip must be shared. The only additional input signals are the capability register number (4 bits) and the index register number (2 bits). The error signal is returned as an output signal.

The chip is structured using the bit slice technique. Thus the basic element consists of one bit of the base register, one bit of the index register, one bit of the offset register, one bit of the three way adder, one bit of the carry propagation adder and one bit of subtractor. These single bit slices can then be concatenated to form an offset of any length. Each bit slice is structured using the Mead and Conway [Mead & Conway, 1980] design approach.

4.2. Implementation Details

The registers in the bit slices are dynamic and thus each has a refresh line and a load line. The base and limit registers are connected to precharged base and limit buses. These two buses extend down the chip in one direction, thus the height of the chip is determined by the number of capability registers. Each register must actively discharge the bus to assert a zero value. The initial implementation only provides 4 capability registers. The index registers are also connected to a precharged bus, but there are only three registers available. The index mode of addressing may be disabled by a kill-index signal which discharges the index bus. This mode is useful if the addressing mode does not need an index register value.

The three way adder is built from two function blocks. The offset register is connected to the diffusion lines and the base and index buses are connected to the poly control lines. The output of the two function blocks is a sum signal and a carry signal. These are connected to a full adder cell. The adder cell is taken from the OM2 processor [Mead & Conway, 1980 (chapter 5); Newkirk & Mathews, 1983] and uses a precharged carry chain with carry kill logic. The output of the adder cell is connected to another adder for the limit subtraction. It may have been possible to remove the subtractor unit and reuse the adder circuit, however, this was not realized until much of the design had been completed.

The chip makes quite heavy use of precharged buses, both for data transfer and also for carry propagation. Whilst this complicates the overall timing of the chip, it is fast. Each bit slice measures 100 by 800 lambda for 4 capability registers and three index registers. Each extra

capability register would increase the size of each slice by 100 by 70 lambda. Thus a 28 bit slice, without the extra control logic and pad space, would occupy 7 by 2 mm with lambda = 2.5 microns, or 14 sq.mm. These figures indicate that, whilst expensive, it would be possible to fit a 28 bit unit on one multi project chip. At the time of writing this paper the timing was being analysed and figures were not available.

The chip design was constructed using the KIC2 editor and associated conversion tools. MOSSIM was used for logic simulation and SPICE was used for timing analysis.

5. CONCLUSION

Most of the design, testing and analysis has been completed for one bit of the chip. Each slice is designed to be connected end-on with the next slice, making expansion easy. The capability registers and base-limit buses were also designed so that they could be extended if extra registers were required. Our experience in the MONADS project is that 16 registers is ample, and it may be possible to manage with less.

It is unlikely that a full 28 bit unit will be fabricated due to the cost, however, it is hoped that a 4 bit slice will be built and tested. Future research will be to design a full chip set to implement the MONADS architecture. Candidates for special purpose chips are address translation, cache management and microprogram control. It may be possible to use the current commercial bit slice components for the data pathways given that the address calculations are performed by a special purpose chip.

Some useful functions which were not included in the chip would be an auto increment/decrement mode on the index register. This mode could then be used for stack addressing and for program instruction fetching. Another function not implemented is the scaling of the index register value for conversion of word offsets into byte offsets. Whilst this is logically simple it was not included in the initial design as it complicates each bit slice.

In this paper we have examined the current implementation of address generation in the MONADS architecture, and have described a faster solution. The alternative was not chosen for a TTL version because of the board space required, but can be efficiently implemented using VLSI techniques.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the influence that Professor Les Keedy, the instigator of the project, has had on the MONADS architecture. Thanks are due to Professor Chris Wallace for reminding us of how to add many numbers together. Thanks also go to S.Y. Tam and L.O. Sim for their help in implementing parts of the chip.

REFERENCES

ABRAMSON, D.A., 1983. The MONADS II Computer System. Proc. 6th. Australian Computer Science Conference, Sydney, pp. 1-10.

ABRAMSON, D.A. & KEEDY, J.L., 1985. Implementing a Large Virtual Memory in a Distributed Computing System. Proc. 18th. Hawaii International Conference on System Sciences, Honolulu.

DENNIS, J.B. & VAN HORN, E.C., 1966. Programming Semantics for Multiprogrammed Computations. Comm. ACM, 9, 3, pp. 143-155.

ENGLAND, D.M., 1972. Architectural Features of System 250. Infotech State of the ART Report 14, Operating Systems, pp. 395-426.

FABRY, R.S., 1974. Capability Based Addressing. Comm. ACM, 17, 7, pp. 403-412.

I.B.M., 1978. IBM System/38 Technical Developments. General Systems Division, I.B.M.

INTEL, 1978. Intel Multibus Specification. Manual Order Number 9800683-02, Intel Corporation.

INTEL, 1981. Introduction to the iAPX432 Architecture. Intel Corporation Manual Order No. 171821-001.

KEEDY, J.L., 1980. Paging and Small Segments: A Memory Management Model. Proc. 8th. World Computer Congress, IFIP-80, Melbourne, pp. 337-342.

KEEDY, J.L., 1982. The MONADS View of Software Modules. Proc. 9th. Australian Computer Conference, Hobart, pp. 560-574.

KEEDY, J.L., ABRAMSON, D.A., ROSENBERG, J. & ROWE, D.M., 1982. A Comparison of the MONADS II and III Computer Systems. Proc. 9th. Australian Computer Conference, Hobart, pp. 581-587.

MEAD, C. & CONWAY, L., 1980. Introduction to VLSI Systems. Addison-Wesley Publishing Company.

NEEDHAM, R.M., 1977. The CAP Project - An Interim Evaluation. Proc. of 6th. ACM Symposium on Operating System Principles, pp. 17-22.

NEWKIRK, J. & MATHEWS, R., 1983. The VLSI Designer's Library. Addison-Wesley Publishing Company.

PARNAS, D.L., 1972. On the Criteria to be Used in Decomposing Systems into Modules. Comm. ACM, 15, 12, pp. 1053-1058.

ROSENBERG, J., ROWE, D.M. & KEEDY, J.L., 1982. An Overview of the MONADS Series III Architecture. Proc. 5th. Australian Computer Science Conference, Perth, pp. 58-67.

ROSENBERG, J., 1982. The MONADS Series III Instruction Set. Proc. 9th. Australian Computer Conference, Hobart, pp. 704-718.

ROSENBERG, J. & ABRAMSON, D.A., 1985. MONADS-PC - A Capability-Based Workstation to Support Software Engineering. Proc. 18th Hawaii International Conference on System Sciences, Honolulu.

ROWE, D.M., 1982. MONADS III Inter-unit Communication. Proc. 9th. Australian Computer Conference, Hobart, pp. 719-729.

WALLACE, C.S., 1964. A Suggestion for a Fast Multiplier. IEEE Trans. on Electronic Computers, Vol. EC-14, No. 1, pp. 14-17.

WULF, W., COHEN, E., CORWIN, W., JONES, A., LEVIN, R., PIERSON, C. & POLLACK, F., 1974. HYDRA - The Kernel of a Multiprocessor Operating System. Comm. ACM, 17, 6, pp. 337-345.

4th AUSTRALIAN MICROELECTRONICS CONFERENCE

Migrating Systems to Silicon

MAY 13 - 15 1985

SYDNEY AUSTRALIA

PROCEEDINGS

Sponsored by:

THE INSTITUTION OF RADIO AND ELECTRONICS ENGINEERS AUSTRALIA

THE INSTITUTION OF ENGINEERS AUSTRALIA

THE INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERS INC.

THE AUSTRALIAN COMPUTER SOCIETY