HARDWARE MANAGEMENT OF

A LARGE VIRTUAL MEMORY.

D. Abramson,
Dept. of Computer Science,
Monash University.

ABSTRACT

The MONADS II Computer was built in the Computer Science department at Monash University in 1980. Among its many features, the Series II utilizes a capability based addressing scheme, in a large virtual memory.

Standard techniques unfortunately fail to provide an efficient mechanism for translating the Series II virtual addresses into main memory addresses.

Another technique is proposed for mapping very large addresses, which operates very efficiently for a relatively low cost.

The address translation units of two other computers are examined and are compared to the Series II unit.

1.0 Introduction

1.1 The MONADS Project.

The MONADS project began in 1976, with the intention of investigating methods for developing large software systems. An operating system ([1]-[6]) was implemented to execute on the MONADS Series I computer; a modified HP2100A minicomputer [2], [7]. The principles underlying the design of the operating system extend far beyond that system, and indeed can be applied to the development of any large software system.

During the development of the Series I system, it became evident that the available hardware was not entirely suitable for supporting the MONADS software structures. This prompted the building of a second computer, the MONADS Series II [8][9], which is partly based on a vastly modified HP2100A minicomputer.

1.2 MONADS Series II.

The MONADS Series II processor was designed to support an environment sympathetic to the philosophy of the MONADS project. It provides constructs for efficiently managing and supporting the key features of MONADS, some of which are not well 'understood' by other computers, including the Series I processor. This paper describes one of these areas, the Virtual Memory System.

Section 2 portrays a logical view of virtual memory systems whilst Section 3 comments on some standard implementations and their problems. Section 4 describes the Series II address translation unit and Section 5 compares it with two other virtual memory systems.

2.0 Logical View of Memory

2.1 Series II Virtual Memory.

It was demonstrated by the Multics designers [10] in 1964 that it was highly desirable to treat the memory of a processor in a homogeneous manner. From a user's viewpoint, there is no conceptual difference between a block of core (or semiconductor) memory and a block of disc memory; the only practical difference being the way in which the data is retrieved and the relative speed at which it is returned.

It was therefore decided that the Series II processor would provide the user (and system) with a very large virtual memory, and like multics, draw no distinction between fast and slow memory (or between files and arrays).

It was also deemed desirable that programs could be broken into their logical sections, or segments, so that these segments could be treated separately.

Thus the Virtual memory of the Series II is designed as a very large segmented memory (as this is now the only form of storage). Each segment is paged to simplify the enormous task of managing a segmented memory [15].

2.2 Address Translation.

Address translation is the process of mapping an address as viewed from the processor onto the physical address required by the main memory system. If the address to be translated is not resident in main memory a page fault interrupt is caused, and the supervisor program fetches the page into mainstore.

The virtual address is typically divided into two sections, a virtual page number (possibly including a segment identifier) and an offset within page. The address translator maps the virtual page number onto a main memory page number, which is combined with the unaffected offset to form a main memory address. The model translation process is shown in Figure 1.

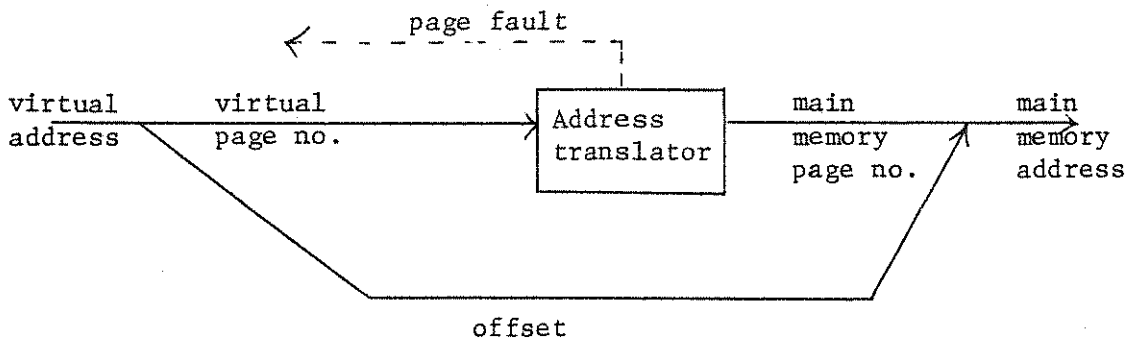


FIGURE 1

3.0 Virtual Memory Systems

3.1 Virtual Memory Categories.

For the purpose of implementing address translation hardware, virtual memory systems may be divided into three categories:

(1) Small Virtual memories
(2) Large Virtual memories with small main memories
(3) Large Virtual memories with large main memories.

3.1.1 Category (1) refers to systems where the address space viewed from a program is small enough to allow the address translation tables to be held directly in the hardware [7] [11]. Some such systems allow the operating system to swap these tables into and out of the hardware, so that individual processes may execute their own isolated address spaces.

These systems are relatively unaffected by changes in the size of the main memory, as this only alters the width of the translation tables. However, they are greatly affected by the size of the virtual address space, as this alters the length of the translation tables. A simple address translation unit is shown in Figure 2.

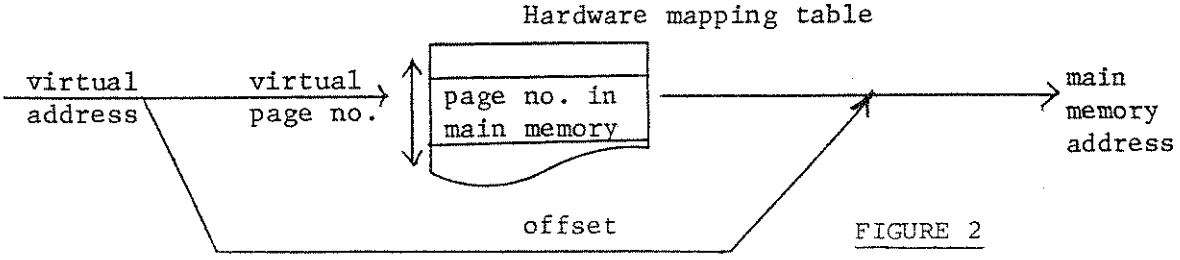


FIGURE 2

3.1.2 In systems with a large virtual memory and a small main memory (c.f. Atlas [12]), address translation is typically accomplished by utilising an associative memory to hold the translation tables. Each page of main memory is associated with one page address register, which holds the virtual address pertaining to that main memory page.

Translation is accomplished by the simultaneous comparison of the contents of each page address register with the page number part of the virtual address; the matching register then pointing to the page in main memory is the required one. Contrary to category (1), this technique is relatively unaffected by changes in the size of the virtual memory. However, it is greatly affected by the size of the main memory, as this alters the number of page address registers and comparators required. Such a scheme is described in Figure 3.

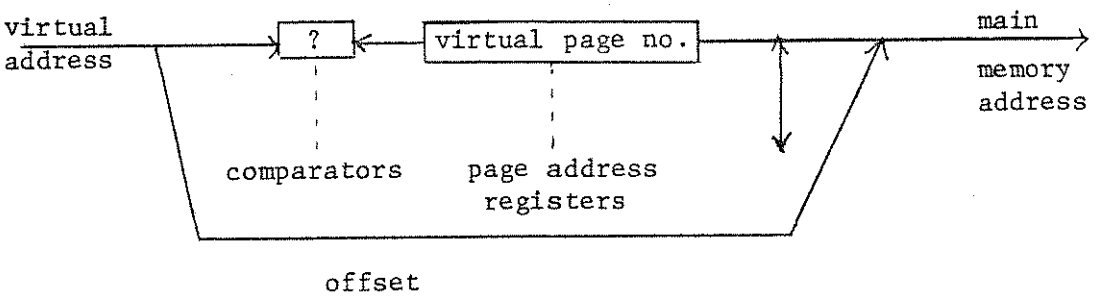


FIGURE 3

The use of an associative memory cannot, unfortunately, be extended to provide translation for category (3) systems due to the number of page address registers and comparators required.

3.1.3 The classical solution for systems with large virtual memories and large main memories (category 3) [10] [13] involves the use of page tables, (and segment tables) which are held either in main memory or virtual memory. These tables, which are maintained and searched by system software and firmware, offer a multi level indexed address translation table.

To achieve respectable translation times, this mechanism is usually augmented by a small associative memory, which holds the most recently used page table entries. This scheme is shown in Figure 4.

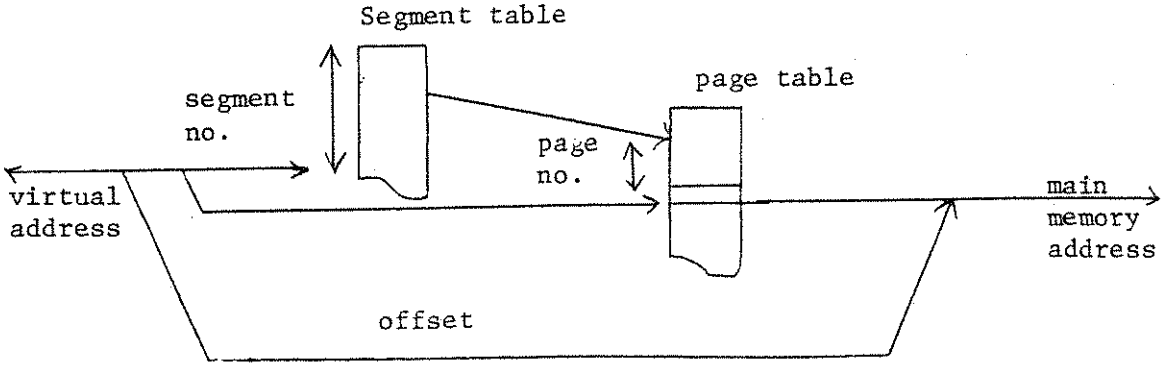


FIGURE 4

3.2 This classical solution would unfortunately fail to provide an efficient address translation mechanism for the Series II processor. The addressing style of Series II (which is capability based)[22] would introduce a very large number of small segments [14]. In addition, the page tables, which would be too large to reside in main memory, would be forced into virtual memory. This would greatly complicate the address translation software, and cuase the translation process to become extremely inefficient.

3.3 Logically, the solution adopted by systems such as Atlas (category (2) systems) would offer the best performance. However, the production of an associative memory capable of managing the many thousand pages of main memory, has not yet become feasible.

The next section describes an address translation unit which emulates a large associative memory very efficiently.

4.0 The Series II Virtual Memory System

4.1 The Virtual Address.

The virtual memory of the Series II processor is addressed by a 31 bit virtual address, which is unforgeable and unique across the system. From the viewpoint of the virtual memory hardware, this address <segment no (16 bits), page no, (6 bits) displacement (9 bits)> may be considered as 22 bits of virtual page identifier, and 9 bits of within page displacement, <virtual page no (22 bits), displacement (9 bits)>

Because the address is unique, the mapping hardware need never concern itself with the identity of the executing process and the system software need never swap mapping entries when switching processes.

4.2 The Physical Address.

The Series II physical address, for the main memory, consists of 13 bits of page number and 9 bits of within page displacement. This 22 bit address <page no (13 bits), displacement (4 bits)> allows a maximum main memory size of 8 MB.

4.3 Control of the Virtual Memory.

Control of the virtual memory may be divided into three sections, namely:

(1) Mapping Hardware
(2) Swap out software
(3) Swap in software.

Sections (2) and (3) are responsible for moving pages out of and into main memory respectively, and are managed by the Series II Hardware Kernel [16]. This software is fully described in a companion paper [17]. The remainder of this paper is concerned only with section (1).

4.4 Mapping Hardware.

The Mapping Hardware is solely responsible for mapping the 31 bit virtual address onto a 22 bit physical address. The external appearance is characterized in Figure 5.

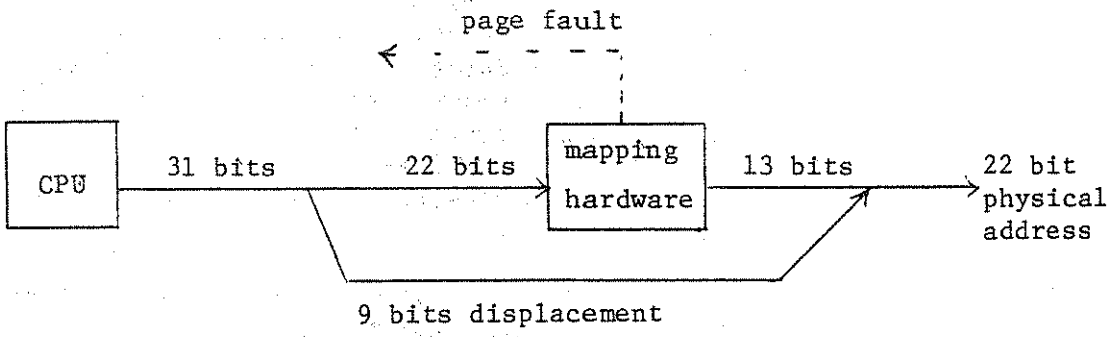


FIGURE 5

When presented with a virtual address, the mapping hardware may either produce a physical address, or a page fault signal. If the page required is in main memory, a physical address will be produced. However, if the page required is in secondary memory, a page fault interrupt will be signalled to the system, in order that remedial action may be taken.

4.5 Internal Operation.

The mapping hardware is organised as a high speed sparsely occupied hash table, with embedded overflow chains, as characterized in Figure 6.

The unit consists of three main components, a hashing unit, a hash table and a comparator.

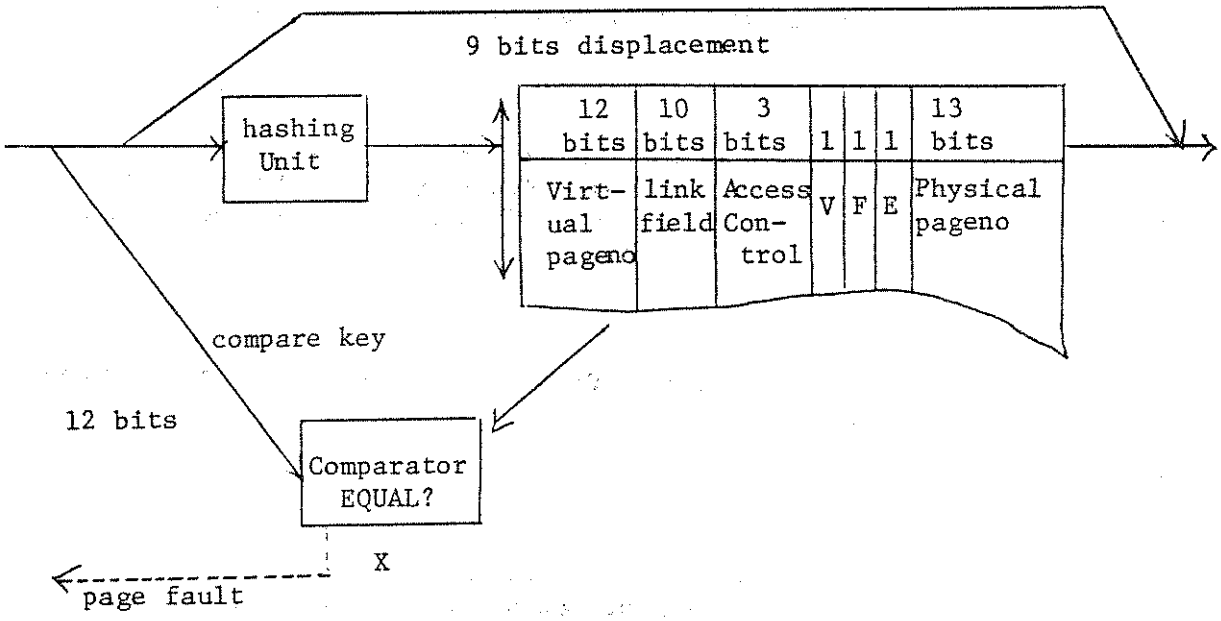


FIGURE 6

4.5.1 Hashing Unit.

The hashing unit accepts a 22 bit virtual page number and generates a 10 bits uniformly distributed index into the hash table.

The current version of the hashing unit uses low order bits from both the segment field and the page field of the virtual address (See 4.9.2.)

4.5.2 Hash Table.

The hash table is implemented using very high speed bipolar memory. Each cell is addressed by a 10 bit hash key, and contains seven fields:

(1)	virtual address identifier		12 bits
(2)	Physical page number		13 bits
(3)	Access field		3 bits
(4)	Valid field	(V)	1 bit
(5)	Link field		10 bits
(6)	Foreigner field	(F)	1 bit
(7)	end of chain field	(E)	1 bit

4.5.2.1 Virtual Address Identifier.

This field contains the remaining 12 bits of the virtual page number not used by the hashing function. All other information in the cell pertains to this virtual page.

4.5.2.2 Physical Page number.

The field holds the page number to which the virtual page is mapped.

4.5.2.3 Access Control Field

This field consists of three access control bits, controlling read, write and execute access respectively.

If a reference to a page contravenes any of the access control bits, an interrupt is generated and the reference is aborted.

4.5.2.4 Valid field.

This field specifies whether the virtual address identifier field contains a valid address. If an address hashes to an invalid cell, a page fault signal is generated.

4.5.2.5 Link field.

If more than one virtual address hashes to a given cell (i.e. clashes occur), an overflow chain is maintained by using another unused cell in the hash table. This field holds the address of the next cell in the overflow chain. It is maintained by the kernel software and is searched by the mapping hardware.

4.5.2.6 Foreigner field.

An address is foreign to a cell if its hash key value is not equal to the cell number in which it is held. If an address is foreign, the foreign bit is set. This bit is required because the Virtual address identifier field does not hold the entire virtual page number.

4.5.2.7 End of chain field.

This bit is set to signify that the cell is at the end of a link chain.

4.5.3 The comparator.

This unit tests the virtual address identifier field, and those bits of the virtual page number not used by the hashing unit, for equality. If equal, the physical page number field value is used as the translated page number.

4.6 Mapping Unit Operation.

Operation of the mapping unit is summarised in Figure 7. The entire translation process is managed by the mapping hardware, including the following of overflow chains. Control is returned to the software once the memory reference is completed (or after a page fault).

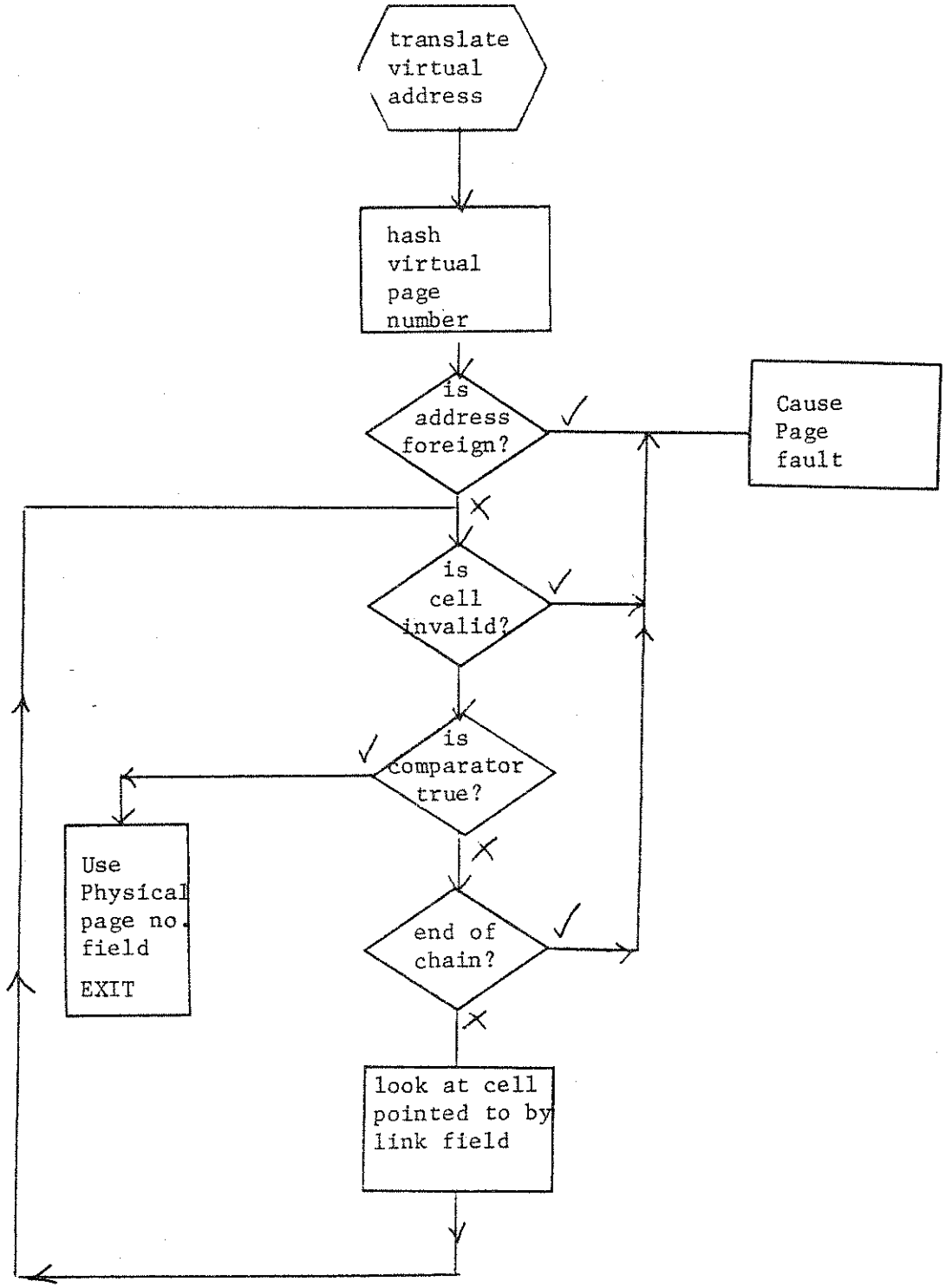


FIGURE 7

4.7 'Peek' Operation

For certain critical system software, it is important to know whether a page reference will cause a page fault. This may be achieved with the 'peek' operation.

When a peek operation is performed on a virtual address, the mapping unit attempts to translate the address. The software may then examine a Series II status register (SVR) to determine whether the reference would have caused a page fault. If the peek operation indicates that no page fault would have occured, then the reference is regarded as a normal (non peek) reference.

However, if the status register indicates a page fault condition, then the page must be fetched into main memory.

4.8 Control of the Mapping Hardware.

To the software responsible for initializing and maintaining the data in the mapping hardware, the hash table and associated registers appear in a special segment, the memory control segment (segment $\emptyset$). Values may be saved into or read from the various fields of the hash table by executing memory reference instructions on this segment.

4.8.1 Insertion and deletion.

Algorithms for inserting and deleting entries from the translation unit are fully defined in [23], and thus are not described here. Correct operation of the hardware demands that the software responsible for insertions and deletions conforms to these algorithms.

4.9 Performance of the Translation Unit.

4.9.1 Loading Factor.

A potential danger with using a hash table is that the number of collisions to any one cell (or clashes), and the average chain length. may become unacceptably high. Acceptable performance can, however, be obtained if the hash table is sparsely occupied (i.e. low loading factor). Providing that the hashing unit generates a uniform distribution of hash keys, the expected number of probes to retrieve an item in the hash table can be calculated from

$$E = 1 + \alpha/2$$

where α = loading factor [18], [23].

The current version of the Series II processor has a hash table size four times the number of pages in physical memory, so in this version $E = 1.125$ which is acceptably low.

(Note that for a true associative memory, $E = 1$)

4.9.2 Hashing Function.

The hash table performance is also affected by the efficiency of the hashing function, which should guarantee a uniform distribution of hash keys. The current version of the hashing unit uses a combination of low order bits from both the segment field and the page field of the virtual address. Should this function yield poor results, experiments may be made with more complex hashing functions.

4.9.3 Timing.

Figure 11 shows the timing delays inherent in the Series II address translation unit. It can be seen that the minimum access time will be

$$t_{min} = \emptyset + 50 + 50 + (300 \sim 700) \text{ ns}$$

$$= 400 \sim 800 \text{ ns}$$

On average

$$t_{av} = (400 \sim 800) + (E-1) \times 100$$

$$= 412 \sim 812 \text{ ns}$$

The variation in the main memory time is dependent on the cycle stealing of the refresh hardware for the dynamic memories used.

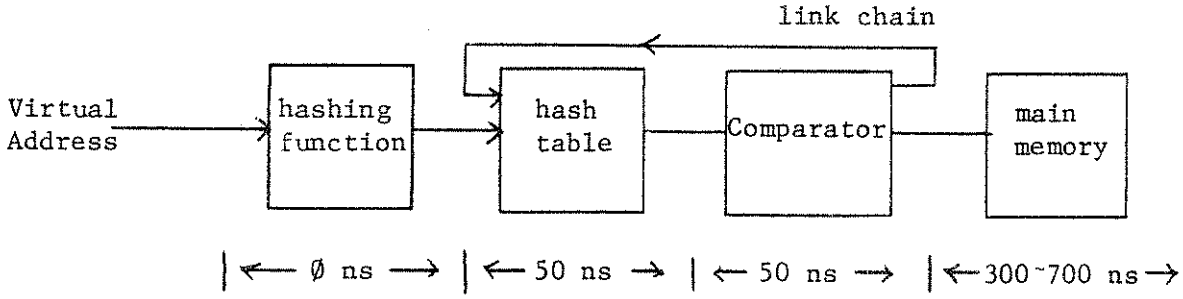


FIGURE 8

4.9.4 Expansion of main memory.

To maintain acceptable performance, the value E must be kept low, thus on addition of main memory the hash table size must be expanded proportionally. This increase in size does not necessarily affect any of the fields within the hash table. If the hash table is divided into blocks, links may be restricted to the 'block' of hash table in which they exist.

4.9.5 Optimization.

Performance of the hash table may be optimized by overlapping the comparison of the virtual page identifier in the current cell to the virtual page number, with the fetch of the cell linked to the current cell. This optimization, however, has little effect if the value E-1 is low.

5.0 Alternative Solutions

Two other computers have made attempts at solving the problem of translating very long virtual addresses, both using unconventional techniques.

5.1 MU6-G [19]

The MU6-G was recently developed at Manchester University as a "high performance machine useful for general and scientific applications". Among its many features, MU6-G includes a Memory Access Controller, or MAC, for translating virtual to physical addresses.

Memory Managment

The virtual address format in the MU6-G processor is as follows

```
< process no.   (8 bits)
  segment no.   (8 bits),
  block no.     (7 bits),
  bit no.       (14 bits)>
```

MU6-G associates one Page Address Register (PAR) with each physical page, as in Atlas [12]. However, rather than using a fully associative mechanism, which would be prohibitively expensive, a sequential search is made of these registers.

PARs are organized in banks of 256 locations, which gives an average translation time a little under 6μs. This unacceptably high time is reduced by the use of a translation look aside buffer. If the main memory cache on MU6-G is ignored, a mainstore access time of 750 ns is achieved.

5.2 The IBM System/38

The IBM System 38 [20] [21] is the most recent computer in the IBM range, and was developed by the General Systems Division in 1978.

Like the Series II, the System/38 translates an extremely long virtual address into a smaller mainstore address.

The translation process involves the use of a sparse hash table in the main memory of the System/38. Two tables are involved; the hash index table and the page directory. The page directory serves as an overflow table, unlike the Series II address translator which uses overflow chains embedded in the hash table. In addition, the hash index table, which is equivalent to the hash table in Series II, is held in the main memory of the system/38. This memory is far slower than the high speed bipolar RAM used in Series II.

Acceptable translation times are only achieved by the addition of a translation look aside buffer.

No timing values have been published for the System/38.

5.3 Conclusion.

Both MU6-G and System/38 require the use of high speed look aside buffers to achieve respectable translation times. The design of such buffers is usually similar to the design of the Series II mapping unit, i.e. as high speed hash tables, but with no overflow strategy, and usually of much smaller size. When the lookaside buffers in MU6-G and System/38 fail to translate an address, some form of slower overflow strategy is utilized. When a clash condition in the Series II mapping unit occurs, (which is less likely than a lookaside buffer failure) a high speed search is made within the hash table.

The paper has demonstrated that the address translation technique utilized by the MONADS Series II computer is practical, and offers very acceptable performance. It suggests that the Series II translation unit should offer equal or superior performance to the MU6-G or System/38 translation units.

Acknowledgments

The author wishes to acknowledge the following people, without whose help this work would never have been possible.

Professor Chris Wallace,
Dr. Les. Keedy,
Dr. John Rosenberg,
Mr. Brian Wallis.

References

[1] Keedy, J.L. (1978) "The MONADS Operating System", Proc. of 8th Australian Computer Conference.

[2] Wallace, C.S. (1978) "Memory and Addressing Extensions to a HP2100A", Proc. of the 8th Australian Computer Conference.

[3] Rosenberg, J. and Keedy, J.L. (1978) "The MONADS Hardware Kernel", Proc. of the 8th Australian Computer Conference.

[4] Ramamohanarao, K. and Keedy, J.L. (1978) "Job Management in the MONADS Operating System", Proc. of the 8th Australian Computer Conference.

[5] Richards, I. and Keedy, J.L. (1978) "Subsystem management in the MONADS Operating System", Proc. of the 8th Australian Computer Conference.

[6] Georgiades, A., Richards, I. and Keedy, J.L. (1978) "A File System for the MONADS Operating System", Proc. of the 8th Australian Computer Conference.

[7] Hagan, R.A. (1980), MSc. Thesis "Virtual Memory Hardware for a HP2100A Minicomputer".

[8] Abramson, D.A. (1980) "A Users Guide to the MONADS Extended Hardware" MONADS Internal Report No. 9.

[9] Abramson, D.A. (1980) "A Users Guide to the MONADS Memory Management Hardware", MONADS Internal Report No. 10.

[10] Organick, E.I.(1972) "The MULTICS System: An Examination of its Structure", MIT Press, Cambridge, MAS. & London.

[11] Hewlett Packard, "The HP21MX Reference Manual"

[12] Kilburn, T., Edwards, D.B.E., Lanigan, M.J. and Sumner, F.H. (1962) "One Level Storage System", I.R.E. Trans. Electronic Computation, EC-11 No. 2, pp 223-234.

[13] Prime, "The System Architecture Reference Guide", PDR 3060, Section 2.

[14] Keedy, J.L. (1980) "Paging and Small Segments: A Memory Management Model", Proc. of IFIP World Congress 1980.

[15] Randell, B. (1969), "A Note on Storage Fragmentation and Program Segmentation", Comms. of A.C.M., July 1969, Vol. 12, No. 7, pp 365-372.

References (contd.)

[16] Wallis, B. (1980) "A Hardware Kernel of the MONADS Series II computer", Honours Report - Dept. of Computer Science, Monash University.

[17] Rosenberg, J. and Keedy, J.L. (1981) "Software Management of a Large Virtual Memory", Proc. of ACSC 4, Brisbane 1981.

[18] Morris, R. (1968) "Scatter Storage Techniques", Comms. of A.C.M., Jan. 1968, pp 38-43.

[19] Edwards, D.B.G., Knowles, A.E. and Woods, J.V. (1980) "The MU6-G. A New Design to Achieve Mainframe Performance from a Mini Sized Computer", Proc. of the 7th Annual Symposium on Computer Architecture, 1980 pp 161 - 167.

[20] Houdek, M.E. and Mitchell, G.R. "Translating a Large Virtual Address" IBM System/38 Technical Developments (1978) pp 19-21.

[21] Hoffman, R.L. and Soltis, F.G. "Hardware Organization of the System/38" IBM System/38 Technical Developments (1978) pp 19-21.

[22] Fabry, R.S. (1974) "Capability-based Addressing", Comms. of A.C.M., July 1974, Vol. 17, No. 7, pp 403-411.

[23] Knuth, D.E., "The Art of Computer Programming", Vol. 3, Section 6.4, pp 506 - 549.

[Editor's note: This paper and its companion, "Software Management of a Large Virtual Memory", by J. Rosenberg and J.L. Keedy, together fall within the page limit for two contributed papers.]

AUSTRALIAN COMPUTER SCIENCE

COMMUNICATIONS

Volume 3, Number 1

May 1981

PROCEEDINGS OF THE

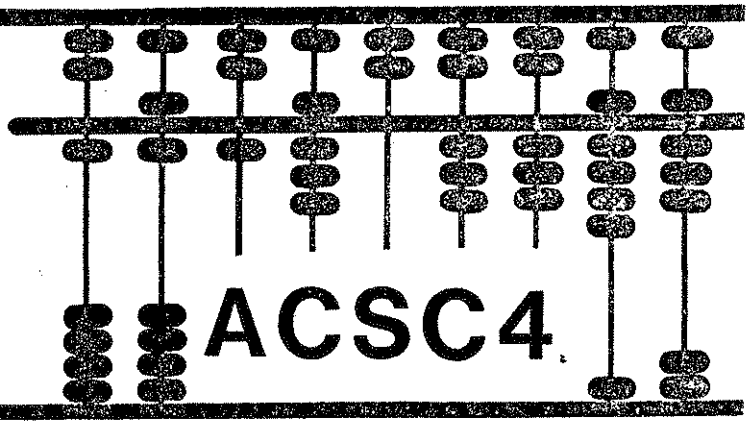
FOURTH AUSTRALIAN COMPUTER SCIENCE CONFERENCE

Department of Computer Science

University of Queensland

St Lucia, Brisbane

Queensland 4067



ISSN 0157-3055