ON A GENERAL PROPERTY OF MEMORY MAPPING TABLES

Karl Reed
Department of Computing
Royal Melbourne Institute of Technology
G.P.O. Box 2476V
Melbourne
Victoria 3001
Australia

ABSTRACT

The paper shows that memory mapping tables can be used to implement the display registers used in providing architectural support for block-structured languages such as Algol 60. This allows full lexical level addressing to be implemented on so-called von-Neuman machines.

The problems of fragmentation of the paged address space are explored, and machines with memory mapping schemes capable of supporting the proposals identified.

Attention is drawn to the similarity between segmented and paged schemes, and it is suggested that the latter may be used to support the former.

Keywords: Memory mapping, page tables, display, segmentation, virtual memory.

CR Categories: 4.12, 4.13, 6.21.

1. INTRODUCTION

It is generally considered that the so-called von-Neuman architectures are different from a lexical level addressed stack architecture such as that implemented by Burroughs on the B6700 (Doran 1979 page 8, Bishop and Barron p. 116, Organick 1973 page vii) because display registers (see Dijkstra 1961) and the associated addressing mechanism are not present.

In practice, display registers can be simulated in software on machines with linear address spaces and programs written in languages assuming a linear address space are made to run on the B6700.

The fact that there are a number of stack based computer architectures, the Hewlet Packard 3000 (Hewlett Packard 1976B), the ICL2900 (Buckle 1978), the MU5 (Morris and Ibbett 1979) and a machine based upon the PDP11 (Tanenbaum 1978) which do not contain display registers is of no relevance to this paper. Our intention is to show that it is possible to support the display-based lexical level addressing on a machine with a paged linear address space.

This is because the construction of the linear address of an on-stack operand or pointer in a lexical-level address architecture such as the B6700 involves algorithmic steps identical to those used for a paged linear address space even though the details of implementation differ. This similarity exists between paged, lexical-level and segmented systems because the operand addresses prior to evaluation are all two-dimensional (Randell & Kuehner 1968 p.297, Denning 1970, Doran 1979). Logical addresses in all cases have the form

$$A_\ell = (\alpha_1, \alpha_2) \qquad (1)$$

where the binary pair (α_1, α_2) is suitably encoded in an instruction's address field.

Physical addresses, on a linear address space are calculated by a mechanism algorithmically equivalent to

$$A_p = M(\alpha_1) + \alpha_2 \qquad (2)$$

where M maps α_1 onto a range of physical addresses.

The similarities between the three mechanisms can be studied by recognizing that each consists of two separate components.

a) Operand Access, the algorithm used to calculate on operand address,

b) Address Space Maintenance, the steps taken during procedure call-like operations which alter the accessible address space, to ensure that operand access is valid. (See also Bishop and Barron.)

We show that operand access equivalence can be maintained for paged and lexical-level addressing on a paged machine, and suggest that a similar equivalence can be established between all three on a paged machine.

If fact, we will show that memory mapping systems such as that provided for process space address translation on the Digital Equipment VAX11/780 could be used without modification.

2. THE ADDRESSING SCHEMES

2.1.1 Operand Access in a Paged Virtual Memory

Paged virtual memory systems based upon one-level store concept (Kilburn et al 1961) are well known and will not be described in detail here. See instead any of the following:-
Randell & Keuhner (1968), Tanenbaum (1976), or Denning (1970).

A logical address α_ℓ consists of the ordered pair $(\alpha'_\ell, \alpha''_\ell)$ where α'_ℓ is the logical page number, and α''_ℓ is the word within page.

Because the number of words in a page is normally a power of two, and physical pages are allocated sequentially with the first usually allocated starting at physical word zero, the fact that the physical address is obtained by

$$\alpha_p = T(\alpha'_\ell) + \alpha''_\ell \qquad (3)$$

where T is a mapping of logical page number onto physical address is obscured (See Denning, 1970 p.163).

Instead, the physical address is obtained by physically concatenating the word within page field, α''_ℓ, with the contents of the page translation table which provides the most significant bits. Equation (3) is actually

$$\alpha_p = T_p(\alpha'_\ell) \,||\, \alpha''_\ell \qquad (4)$$

where T_p maps logical page numbers to physical numbers.

The logical address space is contiguous in the sense that data structures may cross arbitrary numbers of pages, and indexing across page boundaries is permitted.

2.1.2 Address Space Maintenance in a Paged Virtual Memory

The contents of the page tables are not altered when procedure calls or exits are performed, the performance of the system being indistinguishable from a linear, non-virtual address space.

The translation process is shown in Figure 1.

2.2 Lexical Level Addressing

2.2.1 Operand Access in a Lexical Level Stack

Operands, which may be on stack variables or pointers to on or off stack items are addressed with the aid of display registers. The operand address takes the form of an ordered pair $(\ell\ell, i)$ where

- $\ell\ell$ is the lexical level of the block containing the object, and
- i is it's position relative to the start of the lexical level.

The state of a lexical level stack is shown in Figure 2 taken from Doran (Fig.II.5, p.51), complete with static and dynamic links (see also Barron & Bishop).

The address of the object in the linear address space onto which the stack is mapped is

$$\alpha = D(\ell\ell) + i \qquad (5)$$

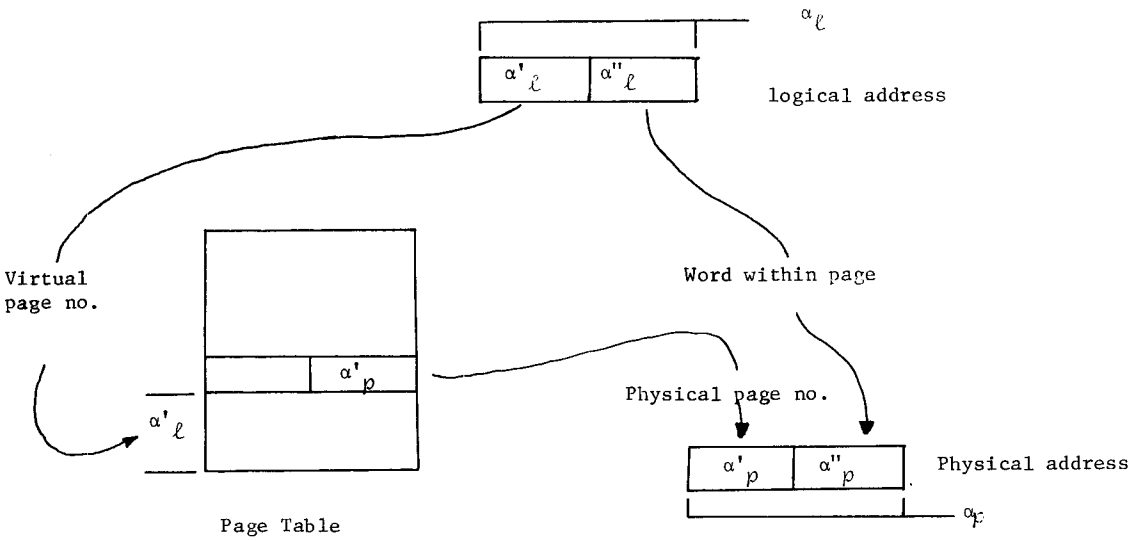


Fig. 1. Typical Virtual Memory Address Generation

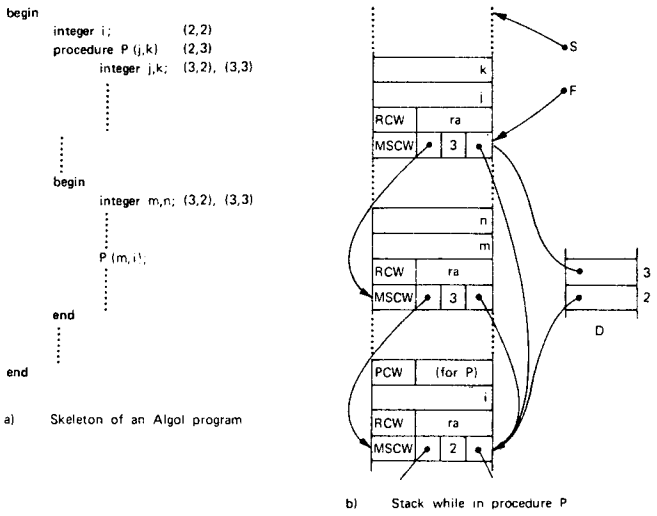


Figure 2. Typical lexical level stack with static and dynamic links

D or display registers on the B6700 contain a 20 bit physical address pointing to the first word of what we will call a lexical stack frame, however, this could have been an address in a linear virtual address space. The maximum number of items which may appear on a lexical level is 2^{13}, and address couples occupy 14 bits (Refer to B6700 reference manual, and Organick 1973 and Doran 1979). A total of 32 display registers are provided.

Consecutive lexical stack frames need not be contiguous on the linear address space. Stack frames are variable length. The provision of a word address in the D registers reduces fragmentation of the linear address space.

2.2.2 Address Space Maintenance in a Lexical Level Stack

Procedure call and exit operators must adjust the contents of the D registers to ensure that operand addressing is correct. The method used on the B6700 (see Organick, Doran & Burroughs for details) relies upon the static and dynamic chains passing through the Mark Stack Control Words (MSCW) as shown in Figure 2. The algorithms are sufficiently well known not to warrant further description.

2.3 Segemented Memory

The essential similarities between segmented and paged address calculations have been noted elsewhere (Tanenbaum 1976 and Randell and Kuehner 1963), and are clear from Denning's discussion of the two methods (Denning 1970 pp.161-164). In the interests of brevity, we focus upon the possibility of using a paged system to achieve lexical level addressing.

3. USE OF PAGED VIRTUAL MEMORY TO ACHIEVE LEXICAL LEVEL ADDRESSING

3.1 Equivalence of Paged and Lexical Level Addressing

We have demonstrated the equivalence of these two methods from the foregoing. It is necessary now to show that the problems of storage fragmentation, space allocation and address space maintenance can be handled at least as well as they can on the B6700.

Particular issues are:-

a) The need for variable sized lexical stack frames.

b) Variance between optimum page and lexical stack frame sizes, and

c) Minimization of logical and physical storage fragmentation.

3.2 An Appropriate Memory Mapping Scheme

Consider a logical-physical page size of p words, where p is a power of two, with a one word page-table-entry (PTE). Further, consider that the page table for any process including the operating system is held in a logical address space called the page-table space (PTS), which is also paged. In other words, the logical to physical address translation for any process is described by a contiguous set of words in the PTS. This is shown in Figure 3 below.

This is identical to the scheme used on the VAX11/780(DEC 1979)for non-system address translation, and to the two level translation on the Data General MV/8000 (Data General page 37). It is also similar to a paged segmented system. (See Denning 1970).

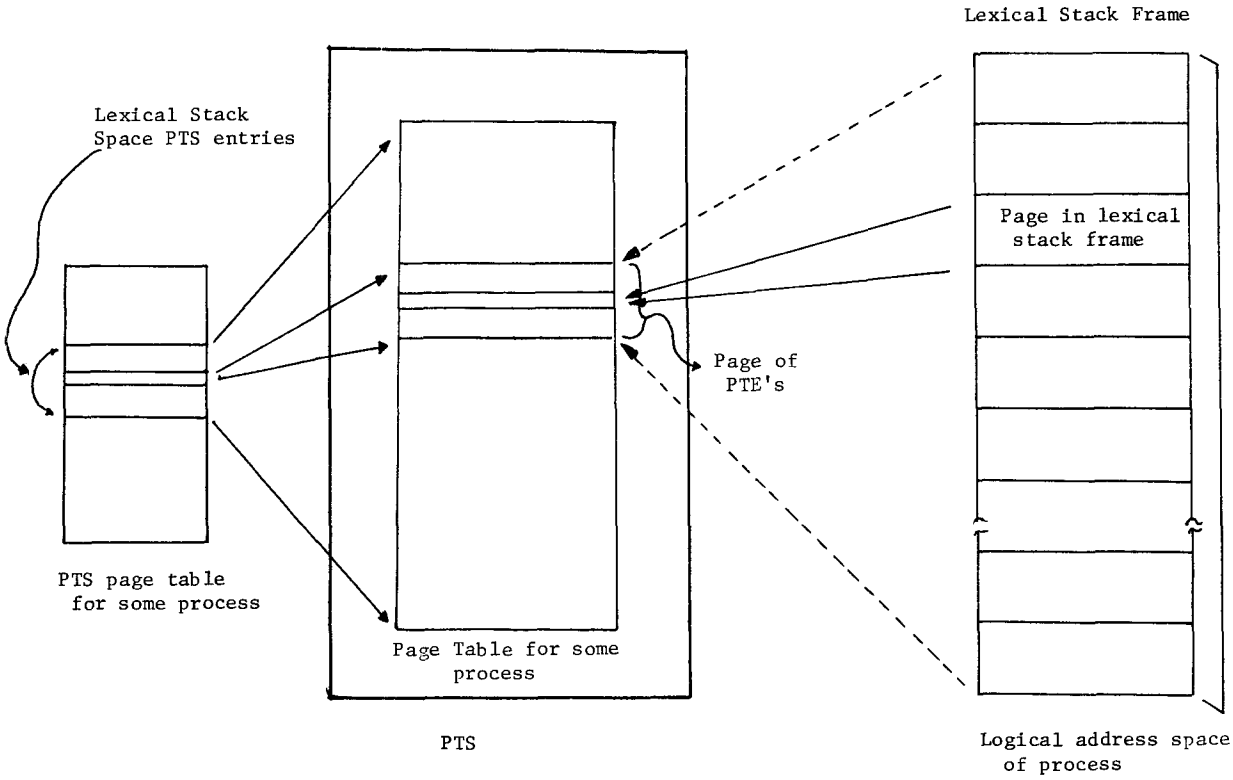


Fig. 3. Address Translation Scheme

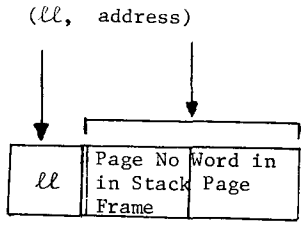
Let us allocate p pages of the logical address space to any lexical level's stack frame, that is p^2 words.

Further, let us limit the total number of lexical levels available to a process to p, so that the lexical level stack is limited to p^3 words at any instant. This need not be contiguous on the logical address space for the process, although stack frames must be contiguous. The p entries in the PTS page table must now be treated as D registers. Compilers producing code requiring lexical level addressing will generate address couples of the form ($\ell\ell$, address) in which the actual word address within a lexical level's stack frame is specified. This will be a word within page number, relative to the start of the lexical level, as shown in Figure 4.

Lexical level zero may start on some arbitrary page, say number L, which is a multiple of p^2, in which case the loader may alter the $\ell\ell$ field of an address to

$$L/p + (\ell\ell - 1)$$

producing a page number in the linear logical address space.



Instruction Address Field

Fig. 4. Compiler Generated Address Couple

Stack frames may be constructed of physical pages in an arbitrary fashion, and paged in and out of physical memory as required. Internal fragmentation is limited to half a physical page per stack frame, and only as many physical pages as required need be allocated.

However, a p^3 word block of the logical address space is unavailable for any other purpose.

The scheme proposed here corresponds to modifying the B6700 implementation so that:-

i) The stack is mapped on to paged linear virtual address space,

ii) The number of pages in a stack frame is a power of 2,

iii) Display registers contain a stack frame number.

3.3 A Suggested Value for p

Assuming that we have a 32 bit word machine with 32 bit address constants, a reasonable value for p would be 128 words, that is 512 bytes on a byte addressable machine. A lexical stack frame would then be 16 Kilowords, and a total of 2 Megawords or 8 Megabytes of the logical address space would be reserved for the lexical stack, a small fraction of the 2^{30} words available.

3.4 Maintenance of the Address Space

Access to the lexical level stack is valid provided procedure entry and exit operators update the display, or in this case, the PTS page table entries performing this function. These operators would also be required to maintain static and dynamic stack links.

Special instructions would need to be provided to achieve this result, there being eight possible address space type transitions, as shown in Table 1.

Table 1 - Procedure Transitions

Operator	Address Space Usage	
	From	To
Entry)	Linear	Linear
Exit)	Linear	Lexical Level
	Lexical Level	Lexical Level
	Lexical Level	Linear

The algorithms for maintaining the display registers are well known (See Doran, Organick and Burroughs). An existing machine with user microcode capability and appropriate memory mapping could be modified to support the lexical level addressing by producing microcode versions of these operators which updated the PTS page entries.

A "doubly mapped" virtual memory scheme was chosen to ensure that a single page table entry could serve as a display register, which holds a logical rather than a physical address, separating stack and virtual address space maintenance issues.

A singly mapped virtual memory would require either large pages, or small pages and a special treatment for the lexical level stack space in which page table entries for these contained logical page numbers. The last arrangement would mean that more than one page table entry would need to be updated whenever the contents of a D register was altered, significantly increasing the overheads associated with procedure entry - exit operators.

The possibility of mapping directly to physical address spaces is a topic for future work.

4, ADDITIONAL PROPERTIES OF THE ADDRESSING SCHEME

4.1 Parameter Transfer and Off-Stack Addressing

The use of an indirect or deferred mode when referencing the lexical level stack provides a means of addressing off-stack items such as arrays, data structures and parameters. The indirect address is a logical address and, if it points to the lexical level stack space, automatically a lexical level address. Parameters can of course be passed readily in this matter.

The mechanism provided also allows for the passing of parameters between procedures running in either address space.

5. CONCLUSIONS

We have shown that lexical level addressing can be implemented on paged linear virtual address machine which has a double or two level mapping scheme. This corresponds to a paged-segmented address space as described by Denning (1970 p.164) and implemented on the IBM 370 (Case and Padegs) and the ICL 2900 (Buckle) for example.

Stack frames can be allocated freely in the remaining linear virtual address-space, and fragmentation is limited to the contiguous component of linear virtual address-space allocated to the lexical level stack. Internal fragmentation is limited to one half page per lexical stack frame actually in use.

It could be shown by similar reasoning that a segmented paged addressing scheme could support lexical level addressing, and that a linear paged addressing scheme could support a segmented address space.

This analysis offers computer architects a means of providing hardware support for three different address space schemes using one underlying addressing mechanism and special procedure entry-exit operators.

The key to the analysis is the fact that there are two different components of an addressing scheme, as already mentioned the access mechanism and the address-space maintenance mechanism. This has also been noted by Bishop and Barron.

6. ACKNOWLEDGEMENT

I wish to acknowledge the help and encouragement of Professor C.S. Wallace in the work that lead to this paper, and the critical comments of those who referred an earlier draft.

REFERENCES

{Bishop, and Barron}
Bishop, J.M. and Barron, D.W.
"Procedure calling and structured architecture"
Computer Journal Vol.23 No.2

{Buckle (1978)}
Buckle, J.K.
"The ICL 2900 Series"
MacMillan Press Ltd.
London and Basingstoke

{Burroughs}
B6700 Information Processing Systems
Reference Manual
Burroughs Corp. Detroit
Item AA190266, AA119114

{Case and Padegs (1978)}
Case, R.P. and Padegs, A.
"Architecture of the IBM System/370"
Communications of the ACM, Vol.21 No.1
Jan. 1978, pp.73-96

{Data General (1980)}
"Eclipse MV/8000 Principles of Operation"
Data General Corporation
April 1980

{DEC 1978}
DEC
"Vax 11/780 Hardware Handbook"
Digital Equipment Corporation 1978

{Denning (1970)}
Denning, P.J.
"Virtual Memory"
Computing Surveys, Vol.2 No.3, pp.153-189
Sep. 1970

{Dijkstra (1961)}
Dijkstra, E.W.
"Making a Translator for Algol 60"
in "Annual Review in Automatic
Programming," Vol.3, pp.347-356 (1963)

{Doran (1979)}
Doran, R.W.
"Computer Architecture: A Structured Approach"
Academic Press, 1979

{Hewlett Packard (1976)}
HP Series II Computer System
System Reference Manual
30000-90020

{Morris and Ibbett (1979)}
Morris, D. and Ibbett, R.
"The MU5 Computer System"
The MacMillan Press 1979

{Organick (1973)}
Organick, E.J.
"Computer System Organization:
the B5700/B6700 Series"
Academic Press 1973

{Randell and Kuehner (1973)}
Randell, B. and Kuehner, C.J.
"Dynamic Storage Allocation Systems"
Communications of the ACM, Vol.11 No.5,
May 1968, pp.297-305

{Tanenbaum (1976)}
Tanenbaum, A.S.
"Structured Computer Organization"
Prentice-Hall 1976

{Tanenbaum (1978)}
Tanenbaum, A.S.
"Implications of structured Programming
for Machine Architecture"
Communications of the ACM, Vol.21 No.3
Mar. 1978, pp.237-246