M E M O

To: CSL/SSL

Date: November 8, 1974

From: Ben Wegbreit

Location: Palo Alto

Subject: Alto Virtual Memory

Organization: PARC/CSL

File: <VANJEPMOND>VMS.PUB;11

1. SCOPE

This is the complete description of the Alto virtual memory address translation design at the functional and register-transfer level. It specifies addressing modes, hardware required to map them, and expected peformance as determined by simulations.

2. INTRODUCTION

2.1 Background

The virtual memory system for the Alto provides an address space of 2↑24 words at the micro-instruction level. All language systems can use this virtual address space. Translation of virtual addresses into physical addresses is done with a small set of non-associative mapping registers.

Five constraints were considered primary and guided the design:

(1) a large linear address space - 2↑24 words (primarily for Lisp)

(2) most addresses issued from the processor to the memory represented in 16-bits, since the arithmetic unit and data paths on the Alto are only 16-bits wide

(3) hardware translation of full 24-bit addresses when this addressing mode is used

(4) small overhead for address translation

(5) low cost - comparable to or less than the rest of the processor

A number of alternatives to the chosen design were considered and rejected as failing to meet, or not cost effective under these constraints:

(1) translation table in high-speed memory to map virtual addresses into physical addresses (as in Visc) - rejected since the size of virtual memory makes the cost of storage for such a table prohibitive.

(2) a set of associative registers for the most recently used pages (as in the BBN pager) - rejected since hardware for associative registers is relatively expensive and difficult to procure.

(3) a segmentation scheme with explicit overlays of file segments into the address space (as in the PDP 11/45) - rejected since it requires more preplanning by the programmer.

2.2 Main Idea

The basic addressing structure presented by the virtual memory system is a large linear space, of 2↑24 16-bit words. It is possible, in principle, to make all memory references via full 24-bit addresses.

However, we may make the empirical observation that for most programs, memory references are not distributed at random in the address space; typically they are relatively small displacements from a small set of infrequently changing base points. Examples include:

(E1) procedure parameters and local variables, as displacements from a frame pointer

(E2) the transfer vector and other system-wide data, as displacements from a global base.

(E3) instance variables (i.e. own variables of Algol, local statics of BCPL and Mesa), as displacements from an instance pointer.

(E4) elements of an array or record, as displacements from the array or record base.

(E5) instructions of a procedure, as displacements from the procedure head.

Measurements on Mesa and BCPL indicate that on the order of 90% of memory references can be naturally described in this way.

With normal page sizes, the base points need not necessarily fall on page boundaries. Also, the set of words which may be logically represented by displacements from some given base may fill several pages. Finally, note that this set of words, S1, may be part of a larger set, S2, which should be moved in and out of secondary storage together, if the probability is high that when some item in S1 is referenced then some item in S2-S1 will be referenced soon. For example, (E5) has this property if procedures are grouped into modules of related procedures. Similarly, (E1) has this property if a stack is used to hold frames: the stack top, consisting of one or more pages, should be kept together. In these cases, the base point for relative addressing moves within the larger set.

Define a page group to be this larger set, i.e. a group of contiguous virtual pages all of which are to be obtained when any of them are referenced. We impose the requirement that the pages of a page group are always to be brought into contiguous physical pages. Hence, given two addresses within a page group, their difference is the same in physical as in virtual space. A page group is described by its virtual start address and its length; if in physical memory, it also has a temporary physical start address. The length is necessarily no larger than the size of physical memory - 2↑16 words.

A page group differs from a segment as follows: As the term is normally used, a segment is a purely logical entity: it may be smaller than a page or larger than physical memory. In some sense, a page group is an implementation-constrained approximation to the concept of segment.

The single idea which underlies the virtual memory system is this: Most addresses can be expressed as displacements within a page group from a base pointer PA in the virtual address space and mapped as displacements from a

corresponding base pointer P in the physical address space. For such addresses, the specification is short (16 bits for the displacement) and translation is fast (add P to the displacement). We refer to this as Displacement or D-mode addressing.

The virtual memory system provides hardware on the memory interface board for 16 such base points. Specifically, there are 16 base registers, each of which describes a page group currently resident in physical memory and contains:

CL - the low address of the page group in physical memory space

CH - the high address of the page group in physical memory space

P - a base pointer in physical space, within the page group

a representation of the correspondence between physical and virtual space. (It proves convenient to store P-PA)

protection bits - discussed in Section 5.6.

2.3 Other Functions of the Virtual Memory System

In addition to mapping addresses expressed as displacements from a base pointer, the virtual memory system provides for:

(1) changing PA, and hence P, within a page group

(2) addresses described as displacements from the start of the page group (rather than from PA)

(3) loading a base register

(4) addresses which must be specified as full 24-bit virtual addresses.

(5) checking, on all memory references, that protection constraints are observed.

Except for details (discussed in Sections 6.3 and 5.6) items (3) and (5) are obvious. We consider (1), (2), and (4).

On procedure call, the frame pointer must be changed to base the called procedure. For simple hierarchical control implemented on a stack, the new value may be expressed as a displacement (usually known at compile time) from the previous value. The new value may be computed by using a D-mode specification to form the physical address of the new frame base. This physical address may then be loaded into the base pointer P(B). More generally, let PA' be the new value of a virtual base pointer after some change of state. The correspondent P' may be obtained by using the virtual memory system to map PA': the resulting physical address is P' and may be loaded into P(B). Only the address translation is carried out; there is no need to actually run the memory. We refer to this as Translate or T usage class. (The other usage classes are the standard protection classes: Read, Write, and Execute.)

Because PA may move within a page group, it is sometimes useful to have a mode of 16-bit addressing which is independent of PA. For example, consider a storage pool which lies wholly within a page group. The addresses of blocks within the pool can be described as displacements relative to the start of the page group, can be stored in that form, and can be supplied to the memory interface in that form. We refer to this as E-mode addressing. In effect, E-mode provides a second base pointer for each page group which is constrained to reference the first word of the group.

A virtual address which cannot be expressed as a 16-bit displacement from a previously set-up base pointer must be represented as a full 24-bit virtual address. Such addresses may be divided into functional classes such as:

(1) the code base for a newly entered procedure which may be in a different page group than its caller

(2) the frame pointer for a newly entered procedure, when the total storage for all active procedure frames is so large that it cannot all be fit into a single page group. (This may be the case for Smalltalk instances.)

(3) non-specific read accesses

(4) non-specific write accesses.

For each such functional class, there is a reasonably high probability that successive accesses are well-localized, for example, that two consecutive non-specific reads will access the same page group.

A single base register may be allocated to each class. Using Full or F-mode addressing, a 24-bit virtual address may then be supplied to the memory interface along with a base register number B. If the virtual address falls within the page group described by B, then it is translated by the virtual memory system into a physical address. Otherwise a translation fault occurs.

To handle translation faults of this sort, a hash table is kept, having one entry for each virtual page currently resident in physical memory. For each entry, the key is the virtual page number; other information includes a description of the page group to which the virtual page belongs. (Note that since the hash key is a page number, there will be a distinct hash table entry for each page in a page group). Hardware in the virtual memory system converts the page number of the virtual address which caused the translation fault into an initial probe in the hash table. Checking the initial probe entry and reprobing in the event of hash-collision is the responsibility of the processor. When the correct entry is found, the information thus obtained is sufficient to load some base register B to base that page group. The memory request can then be repeated through B and will be correctly translated.

Mapping a 24-bit virtual address by the base register associated with a functional class is simply a guess that the page group for the address is the same as the page group for the previous address of that functional class. In the event the guess is wrong, a hash lookup is required, which is a relatively slow operation. To decrease the probability of requiring a hash lookup, it is sometimes desirable to associate multiple base registers with each functional class. The hardware provides for multiplicity of 2.

That is, the 16 base registers can be used as 8 pairs. For each pair, say A and Z, state information in the memory interface treats one base register of the pair as current and the other as an alternate. Consider a state in which A is current. In F-mode addressing, a 24-bit virtual address is supplied by the processor to the memory interface along with the number of some pair. The memory interface tries first to map the virtual address using the page group of the current register of that pair, i.e. A. Failing that, it tries Z and, further, changes state so that Z becomes current. Hence, in F-mode addressing if a fault does not occur then the new state of the memory is such that the current register of the pair is the one which succeeded, so successive references to the same page group through a pair succeed on the current one. Further, a fault occurs only when both registers of a pair fail and, in this

case, the new current register of the pair is the old alternate. If the new current register is reloaded, then an LRU replacement algorithm will be realized.

In summary, there are four modes D,E,F and G which specify how address data supplied by the processor is interpreted by the virtual memory system. Each memory request specifies a mode, a usage class (Read, Write, Execute, or T), and address data. The virtual memory system either carries out the specified request or reports a fault.

3. EXPERIMENTAL DATA

The above reasoning depends on two empirical observations:

(O1) that most addresses can be expressed as relatively small displacements from previously established base points, i.e., D and E modes suffice for most references;

(O2) that for each functional class of addresses which cannot be so expressed, successive accesses are well-localized, i.e., that there is a high probability that a series of consecutive accesses for a functional class will be on in the same one or two page groups.

Quantative data on the validity of these observations has been collected by simulating the execution of programs in the three languages expected to make the greatest use of the Alto: Mesa, Lisp, and Bcpl. At the present time, Mesa data is most complete.

3.1 Mesa

In Mesa, the 24-bit addressing modes are used for three functional classes:

(C1) transfers of control such that it is not guaranteed that the called procedure is in the same syntactic module as its caller

(C2) read accesses from an address computed by the user, e.g., array references, string manipulation and pointer chasing, but excluding references to parameters, local variables, own variables, and instruction fetch.

(C3) write access from an address computed by the user.

(Additional discussion of this model can be found in "The Implementation of Mesa on Alto" 8/21/74.) The frequency of each such functional class relative to the total number of memory accesses was measured. The sum of these frequencies is the ratio of 24-bit addresses to all addresses and gives quantitative data supporting empirical observation (O1).

Each memory reference specified by a full 24-bit virtual address might or might not require reloading a base register, depending on whether or not the 24-bit virtual address falls outside the page group(s) of the base register or registers in F and G mode, respectively. For each functional class and each addressing mode, the conditional reloading rate is defined to be the probability that such an access will require reloading a base register. This was also measured, giving quantitative data supporting empirical observation (O2). The weighted mean reloading rate is the sum over the three functional classes of the products (frequency * conditional reloading rate). This is computed twice: once assuming that all 24-bit addresses are F-mode and once assuming G-mode. This is called the overhead caused by the virtual memory system due to base register reloading.

When using G-mode, there is another overhead due to considering multiple base registers sequentially (rather than in parallel as would be the case with associative registers.) The effect of this may be computed as follows. On a G-mode reference, either one or two base registers will be examined - one if the current is correct and two otherwise. The average number thus considered is referred to as the conditional probe depth. Since two registers are considered in G-mode if and only if the corresponding F-mode request would fault, it follows that the conditional probe depth is the F-mode reloading rate + 1. The probe depth for 16-bit addresses is, of course, always 1. The sum over all memory accesses of the product (frequency * probe depth) gives the weighted mean probe depth. This measures the overhead in G-mode caused by considering multiple base registers sequentially.

The above frequency, reloading rates, and probe depths were measured during the execution of five Mesa programs:

(1) compiler - the standard Mesa compiler for Maxc.

(2) analyzer - a program written by E. Satterthwaite to statically analyze Mesa object code

(3) formatter - a text formatter written by E. Satterthwaite.

(4) sequentiallizer - a program written by J. Melvin for sequentiallizing an NLS-like structured file.

(5) debugging session - online, interactive debugging using the Mesa debugging system

In each case, the programs were run for several million instructions.

(1) compiler

	24-bit control transfer	24-bit user reads	24-bit user writes	total (weighted mean)
frequency	2.2%	6.2%	1.1%	9.5%
F reloading rate	99.7%	40.8%	35.7%	(5.1%)
G reloading rate	33.3%	21.5%	24.4%	(2.3%)
G probe depth	2.0	1.41	1.36	(1.09)

(2) analyzer

	24-bit control transfer	24-bit user read	24-bit user write	total (weighted mean)
frequency	0.6%	3.1%	3.2%	6.9%
F reloading rate	77.4%	35.9%	0.9%	(1.6%)
G reloading rate	8.3%	7.7%	0.4%	(0.3%)
G probe depth	1.77	1.36	1.01	(1.07)

(3) formatter

	24-bit control transfer	24-bit user reads	24-bit user writes	total or (weighted mean)
frequency	1.7%	2.4%	1.3%	5.4%
F reloading rate	99.7%	25.1%	2.0%	(2.3%)
G reloading rate	3.0%	3.0%	0.4%	(0.1%)
G probe depth	2.0	1.25	1.02	(1.02)

(4) sequentializer

	24-bit control transfer	24-bit user read	24-bit user write	total or (weighted mean)
frequency	4.4%	2.3%	0.1%	6.7%
F reloading rate	100%	24.6%	23.7%	(5.0%)
G reloading rate	4.3%	9.5%	9.3%	(0.4%)
G probe depth	2.2	1.24	1.24	(1.05)

(5) debugging session

	24-bit control transfer	24-bit user read	24-bit user write	total or (weighted mean)
frequency	2.8%	5.7%	0.6%	7.9%
F reloading rate	99.1%	35.7%	12.2%	(3.9%)
G reloading rate	29.9%	13.8%	4.9%	(1.4%)
G probe depth	1.05	1.30	1.12	(1.04)

In summary, note that

(1) the frequency of 24-bit addressing mode ranges from 9.5% to 5.4%

(2) with G-mode addressing used for 24-bit addresses, the reloading rate ranges from 2.3% to 0.1%.

(3) the weighted mean probe depth in G-mode is essentially 1, i.e. the effect of sequential examination of base registers is negligible.

3.2 Lisp

Comparable data is being collected for Bytelisp. Preliminary results have generally tended to conform to those of Mesa. The most important exception is that the absence of modules (or, equivalently, a block compiler) in Bytelisp implies that the probability of control transfer requiring the reloading of a base register is higher than in Mesa.

3.3 Bcpl

Given the current state of the Bcpl simulator, it is difficult to collect identical data for Bcpl. The difficulty arises in transfers of control since the Bcpl simulator handles only selected modules, leaving the others to run directly. Hence, there is no way of collecting data on the frequency of 24-bit addressing mode for control transfers or of the effect of using G-mode addressing on the reloading rate for control transfers. Instead, the data below shows the frequency of inter-module control transfer - a number which may be smaller than the frequency of 24-bit addressing for control transfers. Also, it assumes that F-mode addressing is used for control transfers; hence, any inter-module control transfer is assumed to require reloading a base register.

The following data was obtained as the composite of two modules of the Bcpl compiler working on one program (experiments #1 and #2 of "Some Statistics for Virtual Memory Fans" 7/3/74). The number of instructions simulated for those two modules was 86,400.

	inter-module control transfer	24-bit user reads	24-bit user writes	total or (weighted mean)
frequency	3.9%	9.0%	3.9%	16.8%
F reloading rate	100%	12.1%	35.2%	(6.3%)
G reloading rate	(note 1)	8.5%	2.7%	(4.8%)
G probe depth	(note 2)	1.12	1.33	(1.02)

Note 1 - no data available, treated as 100%

Note 2 - no data available, treated as 1

4. ADDRESSING

A memory request is described by an 8-bit specifier, consisting of:

4-bit C field - giving partial specification of the base register
2-bit mode field - D-mode, E-mode, F-mode, or G-mode
2-bit usage class - Read, Write, Execute, or Translate

We denote bits i through j of register R by R[i:j]. Bits are numbered from the left starting at zero.

4.1. Specification of a Base Register

On all memory requests, the C-field is combined with state information in the memory interface to obtain a base register number B, as follows: C is a 4-bit field. C[0:2] specifies one of 8 base register pairs. Let S(C[0:2]) be the state bit of that pair. Then

B[0:2] = C[0:2]
B[3] = C[3] xor S(C[0:2])

This allows either paired or non-paired usage of the base registers, as follows:

(1) to obtain paired-addressing -- use G-mode and set the low order bit of C to zero when assembling the micro-instruction

(2) to obtain non-paired addressing -- use F-mode and set the status bit of a pair to zero when loading either base register of the pair.

In modes D, E, and F, only B is used in translation. Define B's alternate, B', as follows:

B'[0:2] = C[0:2]
B'[3] = C[3] xor not (S (C[0:2]))

In mode G, B is tried and if it fails, then B' is formed by setting S(C[0:2]) to its complement and trying B'.

4.2. Page Groups

Each base register controls a page group resident in physical memory. Such page groups may be described by:

VL = 16-bit page # in virtual address space, of the low page of a page group.

CL = 8-bit page # in the physical address space, of the low page of a page group.

CH = 8-bit page # in the physical address space, of the high page of a page group.

R,W,E = protection bits for Read, Write, Execute

Additionally, each base register has a base pointer PA which contains an address somewhere within the page group; PA is a full 24-bit virtual address. The physical address corresponding to PA is defined to be P, a 16-bit address. Since P-CL*2↑8 = PA-VL*2↑8, we have

P = PA + CL*2↑8 - VL*2↑8

The correspondence between physical and virtual addresses in the page group is specified by the 16-bit quantity Q, defined

Q = CL - VL = P - PA

4.3. Addressing modes (c.f. Figure 1.)

(a) D - supplies a 16-bit displacement from PA(B)

(b) E - supplies a 16-bit displacement from VL(B)

(c) F - supplies a 24-bit virtual address purported to be within the page group described by B

(d) G - supplies a 24-bit virtual address purported to be within either the page group described by B or by B's alternate.

4.4. Translation

(D_c, E_c, and F_c below are 24-bit quantities.)

(a) the core address corresponding to D is D_c = D+P provided that $CL \leq D_c[8:15] \leq CH$.

(b) the core address corresponding to E is E_c = E+CL*2↑8 provided that $E_c[0:7] = 0$ and $E_c[8:15] \leq CH$

(c) the core address corresponding to F is F_c = F+Q*2↑8 provided that $CL \leq F_c[8:15] \leq CH$ and $F_c[0:7]=0$

(d) identical to F except that B and then B alternate are tried in turn.

Note: In D-mode, a negative displacement n is expressed as the twos complement of the absolute value of n. (Negative displacements in the other modes are meaningless.)

4.5 Usage Classes

There are four usage classes which specify how the memory is to be run and the protection class to be selected:

R - read memory & read-protect selected
W - write memory & write-protect selected
E - read memory & execute-protect selected
T - form MAR in the normal way, but do not run the memory

The first three are standard. The fourth is used for two purposes:

(1) incrementally changing the value of a base pointer P(B) within the page group described by B.

(2) testing whether a base register or a register pair maps a 24-bit virtual address which is to be used as a new base pointer PA.

5. HARDWARE

The memory interface board contains the following components used in the virtual memory system:

S - a 8x1 memory which holds the state bits for the base register pairs

M - a 16x32 memory which holds the rest of the data for base registers

C - a 4-bit register used in forming B

B - a 4-bit Base number register

MIR - a 16-bit Memory Interface Register

WXR - an 8-bit Word Number Register

a 16-bit adder

MAR - a 16-bit Memory Address Register

3 8-bit comparators

a PROM hasher

H - a 16-bit read-out register

Data paths between these are shown in Figure 2.

5.1. Base Registers

Each base register consists of 4 sub-registers, 3 protection bits, and one dirty bit

```
CL - an 8-bit low core page #
CH - an 8-bit high core page #
P - a 16-bit core address (PA+CL*2↑8-VL*2↑8)
Q - a 16-bit page # (CL-VL)
R - read permitted bit
W - write permitted bit
E - execute permitted bit
D - dirty bit
```

Total # of bits per base register = 52.

5.2. State bits

Each even-odd pair of base registers has a state bit, S, used in forming the base register number B.

5.3. Address Input Format

Addresses are supplied to the memory interface board from the ALU output as follows:

(a) D-mode - one word

(b) E-mode - one word

(c) F-mode and G-mode - word number 1 whose high byte is ignored and whose low byte is the first byte of the page number, followed by word number 2 whose high byte is the second byte of the page number and whose low byte is the word number in the page. These two words are supplied on two consecutive minor cycles.

5.4. Adder

The 16-bit adder is split so that it will either:

(1) carry 0 into bit 15 and carry out of bit 8 into bit 7, or

(2) carry 0 into bit 7 and carry out of bit 0 into bit 15.

Hence, it can either:

(1) perform normal 16-bit addition, or

(2) add two numbers with their halves swapped to produce a number with its halves swapped. Because of the way page numbers are alligned in words, this option is useful in F and G modes. We denote this form of addition as A $ B.

5.5. Translation

The following register transfer operations are performed:

(a) D-mode

```
MIR←ALU output
MAR←MIR+P(B)
start memory; test CL(B)≤MAR[0:7]≤CH[B] and fault if not
```

(b) E-mode

same as D-mode except that (CL(B),zeros) is used in place of P(B)

(c) F-mode

```
MIR[8:15]←ALUout[8:15] (first byte of page #)
MIR[0:7]←ALUout[0:7]; WAR←ALUout[8:15]
MAR←(Q(B)[8:15],Q(B)[0:7]) $ MIR and fault if carry out of bit 8
test CL(B)≤MAR[0:7]≤CH(B) and MAR[8:15]=0 and fault if not
MAR[8:15]←WAR and start memory
```

(d) G-mode

```
MIR[8:15]←ALUout[8:15] (first byte of page #)
MIR[0:7]←ALUout[0:7]; WAR←ALUout[8:15]
MAR←(Q(B)[8:15],Q(B)[0:7]) $ MIR and test for no carry out of bit 8
test CL(B)≤MAR[0:7]≤CH(B) and MAR[8:15]=0
if the tests are successful
     then MAR[8:15]←WAR and start memory
     else complement S(B[0:2]), form the new B,
     and continue with:

MAR←(Q(B)[8:15],Q(B)[0:7]) $ MIR and fault if carry out of bit 8
test CL(B)≤MAR[0:7]≤CH(B) and MAR[8:15]=0 and fault if not
MAR[8:15]←WAR and start memory
```

5.6. Protection

The three protection bits associated with a page group when implemented are Readable, Writable, and Executable, on a page group basis. (By Readable, we mean a fetch from memory other than instruction fetch.) Each memory request selects a base register and supplies a usage class:

```
01 - instruction fetch; selects E
10 - other read; selects R
11 - write; selects W
00 - translate only but no actual memory reference; no select
```

If a protection bit is selected by the memory request and is off for that page group, then a protection fault occurs.

5.7. Fault Processing

There are three reasons why a memory request may fail to complete:

(i) protection violation, in that an otherwise legal memory request specifies a usage class Read, Write, or Execute which is not permitted for the page group.

(ii) logical errors in attempting to address outside a page group; e.g.

bounds violations on a frame pointer will be logical errors in most implementations. Such errors should simply be reported back to the processor.

(iii) translation fault, in that the base register used was a guess at the correct one and if the virtual address falls outside of the specified page group, then (some) base register should be loaded by consulting the hash table. Only F and G-mode accesses can cause translation fault.

In consulting the hash table when a translation fault occurs, it proves convenient to divide the processing between the virtual memory system and the processor. Specifically, the initial probe into the hash table is computed by the virtual memory system on a read request there is initiated; all subsequent steps are carried out by the processor. This obtains the first probe entry rapidly while allowing flexibility in the more complex job of testing for collisions and reprobing if that is necessary.

The initial probe is obtained by transforming the contents of MIR (a swapped page number) into an address in the hash table. The table consists of 2-word entries, is 4 pages long, and starts on a page boundary whose page number is a multiple of 4. The initial probe is obtained as follows:

(1) probe[0:5] = page #/4 of the table start address

(2) probe[6:14] is a function of MIR:

(2.1) probe[6:8] = MIR[13:7]
(2.2) probe[9] = MIR[6] xor MIR[2] xor MIR[4]
(2.3) probe[10] = MIR[1] xor MIR[3]
(2.4) for i from 1 to 4 probe[10+i] = MIR[7+i] xor MIR[11+i]

(3) probe[15] = 0

The hash of MIR was chosen so that consecutive pages of the address space hash 17 pages apart, so that page groups of up to 32 entries are guaranteed not to collide, and so that all bits of the page number enter into the hash. The resulting 16-bit number is the page address of the initial probe and is put back into MIR. Transformation takes one minor cycle. Transferring the probe to MAR and starting the memory takes another minor cycle, the word selected by the probe and the word following that appear on the bus 4 and 5 cycles later, respectively.

The memory interface 1 unit makes no other special provisions for hash table lookup. Checking to see the first probe is correct, making subsequent probes if it is not, initiating the reloading of some base register, and finally performing the F or G mode memory request again are all the responsibility of the processor. Note that these activities require the original 24-bit virtual address. Hence, it is necessary to store the full 24-bit address in R registers whenever using F or G-mode addressing.

5.8 Dirty bit

To aid the page manager in deciding which pages to remove from the resident set, it is useful to know which pages have been modified while in physical memory. These pages are said to be dirty and must be written back to the disk if removed from the resident set. On every non-faulting write request through base register B, Y(B) is set to 1. In all other cases, Y(B) is unchanged.

5.9 Read-out register - H

More or less as an accident of the design, there are 7 quantities which cannot be conveniently read out of the memory interface: {CH(B), R(B), W(B), E(B), Y(B), S(B), and the usage class of an initiating microinstruction}. The first 4 of these are needed for hardware debugging; the usage class is needed to determine the request type when a protection fault occurs. H is provided as the cheapest means for reading out these 7 quantities: on every memory request H is loaded with {CH(B), R(B), W(B), E(B), Y(B), S(B), and usage class} - 15 bits in all. (Note that R, W, E are the protection bits for the page group of B while usage class is part of the memory request specification; their compatibility, while necessary for a request to succeed, is not relevent to forming H.) H may be read out onto the BUS as discussed in Section 7.3.

6. HASH TABLE

The hash table contains an entry for each page currently resident in physical memory. The table key is the page number; data in the entry describes the page group to which the page belongs.

6.1. Structure

(a) Each entry in the hash table consists of

PN - 16-bit page # in virtual space

CP - 8-bit page number in physical memory of the physical page which corresponds to PN

CL - 8-bit physical page # of the low page of the page group

CH - 8-bit physical page # of the high page of the page group

R,W,E,Y,Ref - Bits for Readable, Writable, Executable, Dirty and Referenced.

(b) The table is organized as two arrays:

one of 2-word entries, holding PN and CP,R,W,E,Y,Ref
one of 1-word entries, holding CH,CL

(c) The hash table is 1/2 full; hence, it requires 2*3*256 = 1.5K words.

6.2. Looking up a page in the hash table

The initial probe is computed in the memory interface board and is used to start a memory fetch from the probed entry (c.f. Section 5.7) of the first array. Four and five cycles after the initial probe fetch is initiated, words containing PN and {CP,zeros,R,W,E,Y,Ref} appear on the bus, respectively. The initial probe is correct iff PN=page number of the original memory reference. If not, reprobing is necessary.

I suggest linear-reprobing since it is fastest to compute and with the table 50% loaded, the mean number of probes required with linear reprobing (about 1.5) is not much worse than the mean number of probes required with double hashing (about 1.3). Note, however, that the memory interface hardware makes no commitment as to what reprobing technique is used.

In any reprobing method, it is necessary to recover the initial probe. This may be done by Recover (BUS←MAR); c.f. Section 7.3.

6.3. Loading the base registers from the hash table

(a) Relations

CP-CL = PN-VL
Q = CL-VL = CP-PN
P = CP'2↑8

(b) Data to be moved

(i) extract R,W,E,Y, & put into R(B), W(B), E(B), Y(B)
(ii) form CP'2↑8 & put into P(B)
(iii) compute CP-PN & put into Q(B)
(iv) get CH,CL & put into CH(B),CL(B)

(c) Computation

At the start, PN is in some R-register, say R1, and (CP,zeros,R,W, E,Y,Ref) is in some other register, say R2.

```
1) MIR←physical address of entry in 2nd array

2) T←R2
   R(B),W(B),E(B),Y(B)←MIR[11:14]   ; loads protection and dirty bits

3) P(B)←L←177400 and T              ; loads P(B) with CP'2↑8
   MAR←MIR                          ; MAR = physical address in 2nd table

4) R2←L cycle 8                     ; R2 = CP

5) T←R1                             ; T = PN

6) Q(B)←R2-T                        ; loads Q(B) with CP-PN

7) CH,CL(B)←MD                      ; loads CH,CL(B)
```

6.4. Referenced bit

The Ref bit in the hash table is taken over directly from Peter's memo "The Alto Lisp Map", 5/30/74. Its purpose is to assist the page manager in deciding what pages have been recently referenced and which (therefore other) pages should be thrown out. In brief, a queue of recently referenced pages is kept. Every some-milliseconds, all the Ref bits are turned off. Whenever the base-register fault manager loads a base register with a page group for which the Ref bit is off, it turns the bit on. Hence, recently referenced page groups have their Ref bit on. When the some-millisecond mass turnoff is carried out, page groups discovered to have their Ref bits on are moved to the front of the queue. Hence, the most recently referenced pages are in the front of the queue and older references trail off to the end.

6.5 Hash Table Location

Because of the way in which the memory interface computes the initial probe into the hash table when an address translation failure occurs, it is necessary to fix the location of the first hash table array - the one which holds PN, CP, P, W, E, Y, and Ref. Further, because of the way in which the probe is formed, the array must start on a page boundary whose number is a multiple of 4.

7. INTEGRATION INTO THE ALTO

Three currently unused function combinations are used to control and interrogate the memory interface hardware:

(1) memref - memory reference

(2) testfault - test for bounds or protection fault

(3) miscmem - miscellaneous virtual memory system operations

7.1 Memory References

When memref is selected, the following fields of the microinstruction are interpreted specially:

```
0:3     C-field
4,9     mode - D, E, F, or G
10:11   usage class - R, W, E, or T
```

That is, the normal functions of the R and BUS fields are superseded. The output of the ALU is taken as the first word of address being supplied to the memory. (Note that since the R and BUS fields are given special interpretation, the BUS-input to the ALU is specified accidentally. Hence, only those ALU operations which are independent of BUS are well-defined in a memref. Don't use the others. In F and G mode, the output of the ALU on the next cycle will be taken as the second word of address to be supplied to the memory.

7.2 Fault Testing

There are three reasons why a memory request may fail to complete:

(1) protection violation

(2) bounds violation in D or E mode

(3) translation failure in F or G mode

We distinguish (2) from (3) since in case (3) it is assumed that the base register(s) in question was simply a wrong guess and that a hash table lookup is required. When (3) occurs, a protection violation on the page group is meaningless, i.e., (3) supersedes (1) should they both occur on the same memory request. For consistency, we adopt the convention that (2) also overrides (1). A 2-bit fault status register, FS, is set to one of four states for each use of the memref function:

00 - protection violation

01 - bounds violation (D or E mode)

10 - translation failure in F or G mode

11 - no fault

When testfault is selected, the contents of FS is read into the NEXT field of the following microinstruction. This provides a 4-way dispatch. No fault is coded as 11. Bounds violation in D and E mode is obvious. Protection violation signals only that some protection fault has occurred. To determine which of Read, Write, or Execute occurred, it is necessary to examine the usage-class field of the H register, which may be read out using miscmem, of

Section 7.3. In the case of a translation failure in F or G mode, the initial hash probe will be computed in the memory interface, placed in the MAR, and a memory access to the table entry initiated. The resulting word and the word following will appear as MD exactly one minor cycle later than data would have appeared had a non-faulting read been requested in the initiating memory request.

7.3 Miscellaneous Memory Operations

When misfunc is selected, the R and BUS fields are interpreted specially, as follows:

R	BUS	Meaning
0	2	BUS←MAR (Recover)
1	2	[illegible]←H (Read-out)
0	0	Initiate base register reloading sequence
0	1	Hash and probe
C-field	1	[illegible] output
"	3	MAR←H(C)
"	4	MAR←CL(C)+2↑8
"	5	MAR←Q(C)

These are intended to be used as follows:

(1) BUS=MAR - recovers MAR, for (4) and (5)

(2) BUS=H - reads out memory interface board data which is otherwise unavailable: (CL, R, S, L, Y, S, usage class) of the previous memory request, of Section 5.6.

(3) Initiate base register reloading sequence - Once initiated, the sequence of operations proceeds autonomously on the memory interface side, as specified in Section 6.4.e. Note that this constrains the processor side to follow the specified fetch sequence, since there is no other control of the memory interface until the sequence completes.

(4) Hash and probe - This computes the hash probe corresponding to a page number supplied by the ALU, as follows:
MIR←ALU output, then MAR←hash of MIR as in Section 5.7, then MAR←MIR, then the memory read is started. This feature is primarily intended to be used by the paging memory manager in controlling the hash table. To be compatible with normal usage of the MAR, the 16 bit page number is supplied by the ALU with its halves swapped.

(5) P(B)←ALU output - reloads P(B), e.g., for moving a base pointer within a page group.

(6) MAR←H(B), MAR←H(B)+2↑8, MAR←Q(B) - places P, CL, or Q into MAR when it can be recovered by [illegible] for hardware debugging.

Note that operations (2), (3), (5) and (6), deal with data for some B whereas microinstructions specify C. In the case where C and B may differ and some specific B is desired, it is necessary to first read H(C) which is guaranteed to be identical to P(C) and then, using S, select a C which will produce the desired B.

8. INTENDED USAGE

This section is a collection of suggestions and intentions as to how the virtual memory hardware should be used by language implementers.

8.1 Frame Pointers

A frame, e.g. for procedure activation, will reside somewhere in a page group for that frame type. A base register BF will control the page group and P(BF) will point to the core address of the head of the frame. Local variables will be expressed as positive displacements from P(BF) using B-mode addressing. Often, these displacements will only be 4 bits. Mesa statistics indicate that 4 bits suffice in about 95% of the cases (measured statically).

Consider transferring control to a new procedure activation in a co-routine environment such as Mesa. The storage manager for the page group produces the address of a new frame as a 16-bit displacement relative to the start of the page group (blocks are linked that way). The address is supplied to the memory interface board as an E-mode reference. The core address E_c corresponding to E is computed as E_c = E+CL*2↑8 (Section 4.3) then E_c is put into P(BF) using Recover (Section 7.3) followed by Load P(BF).

8.2 Code Pointers

A page group holds all code belonging to a module. Let PC be the base register pair for code. P(PC) points to the head of a local procedure. Instructions are addressed using B-mode as B+P(PC). If a transfer vector for the module is constructed, it can be placed in a fixed place and reference local procedures in E-mode. Assuming less than 2↑8 local procedures in a module, this allows 8-bit addressing of procedures. The new value of P(PC) for a local procedure called in this way is computed as for frame pointers. (See above.)

8.3 User-Computed 24-bit Addresses

Consider reads, writes are analogous. Use G-mode addressing and assign a pair of base registers to the logical function of user-computed reads. When such an address is issued, then the virtual memory system tries the current read register; if it maps successfully, we're done. If it does not map successfully, then the alternate is tried. If it succeeds, then it becomes the new current read register; this realizes MRU trial of the registers in the base pair. If the alternate fails, then a fault occurs and, in processing the fault, the (old) alternate is reloaded; this implements LRU replacement.

Trying the alternate base register takes 2 minor cycles. Reloading a base register takes about 25. Hence trying the alternate is cost effective in time if the chance of success on the alternate (given that the current has failed) is better than 2/25 = .08. Measurements for Mesa give probabilities of .47, .86, and .44 (three different programs); BCPL gives .63. Trying the alternate, i.e. using G instead of E-mode, is a clear winner.

8.4 Control Transfer

Consider a 24-bit control transfer from procedure P to procedure Q, i.e., any transfer on which it is not guaranteed during compilation that P and Q are in the same page group. Suppose a G-mode T usage class memory reference is used to map the address of Q. There are three mutually exclusive outcomes:

(a) succeed on current register of the pair

(b) fail on current but succeed on alternate register of the pair

(c) fail on both

The statistics in Section 3 give the relative frequencies of outcomes for the five Mesa programs simulated:

	Current	Alternate	Fault
compiler	.3%	66.4%	33.3%
analyzer	22.6%	69.1%	8.3%
formatter	.3%	95.9%	3.8%
sequentializer	0%	95.8%	4.2%
debugging session	3.9%	66.6%	29.5%

So why bother with E mode?

Note that the current register is seldom correct: this can occur only for inter-module calls made through a procedure parameter. However, the alternate register is correct roughly 2/3 of the time or better. In either case, the contents of MAR after translation is the new value of P for the (possibly new) current base register for control.

8.5. Other Applications of Using Base Registers in Pairs

The above two examples both involve translation of full 24-bit virtual addresses using G-mode addressing. Some advantages of using registers in pairs apply in D and E mode, in language implementations other than Mesa. In Mesa, address translation faults are assumed to occur only for user-computed addresses and module changes because all frames for activations are assumed to fit into a page group. In Smalltalk this will not be the case for instances, since these effectively form the free storage pool. Consider transferring control back and forth between two instances P and Q. The code bases for P and Q can be handled as discussed in (8.4). The instance pointers for P and Q can be handled analogously.

Let I be a pair of base registers used for instances. A micro-instruction simply specifies D-mode references relative to I. Such references are mapped by I-current. Trying I-alternate is meaningless in this case. However, when changing to a new instance, it is necessary to change the interpretation of I. Proceeding as in (8.4), consider using a G-mode, T-usage-class request to test I-alternate to see if it does, in fact, map the new instance base. As with all such G-mode requests, this changes the state of the I-pair so that the former I-alternate becomes the new I-current. In this way, the micro-instruction specifies only I, while state information associated with the I pair distinguishes which of the two I-registers is intended.

8.6 Protection with Multiple Processes

Consider switching from process P to process Q having a different protection domain. Consider a primitive protection model on the order of that found in the IBM 360/370 series, i.e., data private to a process can only be accessed by that process. This may be obtained as follows: As part of process switching, the base registers are block reloaded to establish Q's state. Only the contents of page groups addressed by the new base register contents can be directly accessed. When a translation fault occurs in I or G mode, the base register reloading sequence is augmented with an additional check: to insure that the page group being loaded is within the protection domain of the current process.

8.7 Statically Nested Scope Rules

In languages with a statically nested scope rule (e.g. Algol, PL/I, and their derivatives) the set of immediately accessible named variables is distributed between the procedure being executed, the procedure which statically encloses it, etc. recursively. The number of statically enclosing procedure levels is, in practice, typically 1 or 2. The set of pointers to the various base points is usually termed the display. Four base registers allocated to serve as the display should suffice for almost all programs. The compiler can determine the static level number and ordinal number within level for each variable. Static level number is reflected in the op-code, selecting a microinstruction containing the correct base register number; ordinal variable number is used as a displacement for D-mode addressing.

8.8 Page Groups Smaller than a Logical Unit

Most of this description has presupposed that a page group constitutes a complete logical unit. It is possible, however, that constraints on physical memory will be such that complete logical units may not fit. The address translation hardware can be used under such constraints - at the cost of a higher translation fault rate.

Specifically, one can distinguish between a segment and a page group - the former being a relatively large logical unit while the latter is used as a smaller window of contiguous pages into the segment. The following changes are then necessary:

(1) In D-mode, it is possible for an address to fall outside the page group and yet be within the segment. That is, address translation faults can occur in D-mode. Hence, when a bounds violation occurs in D-mode, it is necessary to obtain the page number of the virtual address sought, from which the test for within-segment address can be made and a hash probe computed. To obtain the page number, it suffices to: recover MAR (which contains I+P(B)), shift it, get Q(B), and form the difference.

(2) E-mode addressing relative to the head of the segment is impossible. Hence, either E-mode addressing is page group relative, or E-mode addressing is unused.

9. EXTENSIONS

9.1 Larger Physical Address Space

Because the disk access time is more than four orders of magnitude greater than main memory, there is reason to worry that the system may be disk bound (e.g., c.f. the study of the Mesa compiler, Section V of "The Implementation of Mesa on Alto", 8/21/74). There are two cures: more main memory and a faster swapping device (e.g., a CCD drum). More main memory has some attraction for Lisp since the relatively large amount of fixed code it requires limits the available swapping space.

Consider implementing a physical address space of 18 bits, while keeping the virtual addressing mechanism unchanged. Thus, for example, four page groups of 2t16 words each could simultaneously reside in main memory. This would require changing the following quantities:

Quantity	From	To
CL(B)	8	10
CH(B)	8	10
P(B)	16	18
ADDR	16	18
MAR	16	18

Unchanged would be C, B, MIR, MSR, Q(B), S(B), R(B), W(B) and E(B).

Handling these longer quantities would slow down and/or introduce additional complexities into the base register loading sequence and the recover (R/S-MAR) operation. However, the two frequently used operations - memref and fwit test - would remain unchanged. Hence, it is possible to consider implementing a 16-bit physical address space, with the intention of retrofitting later if that proves necessary.

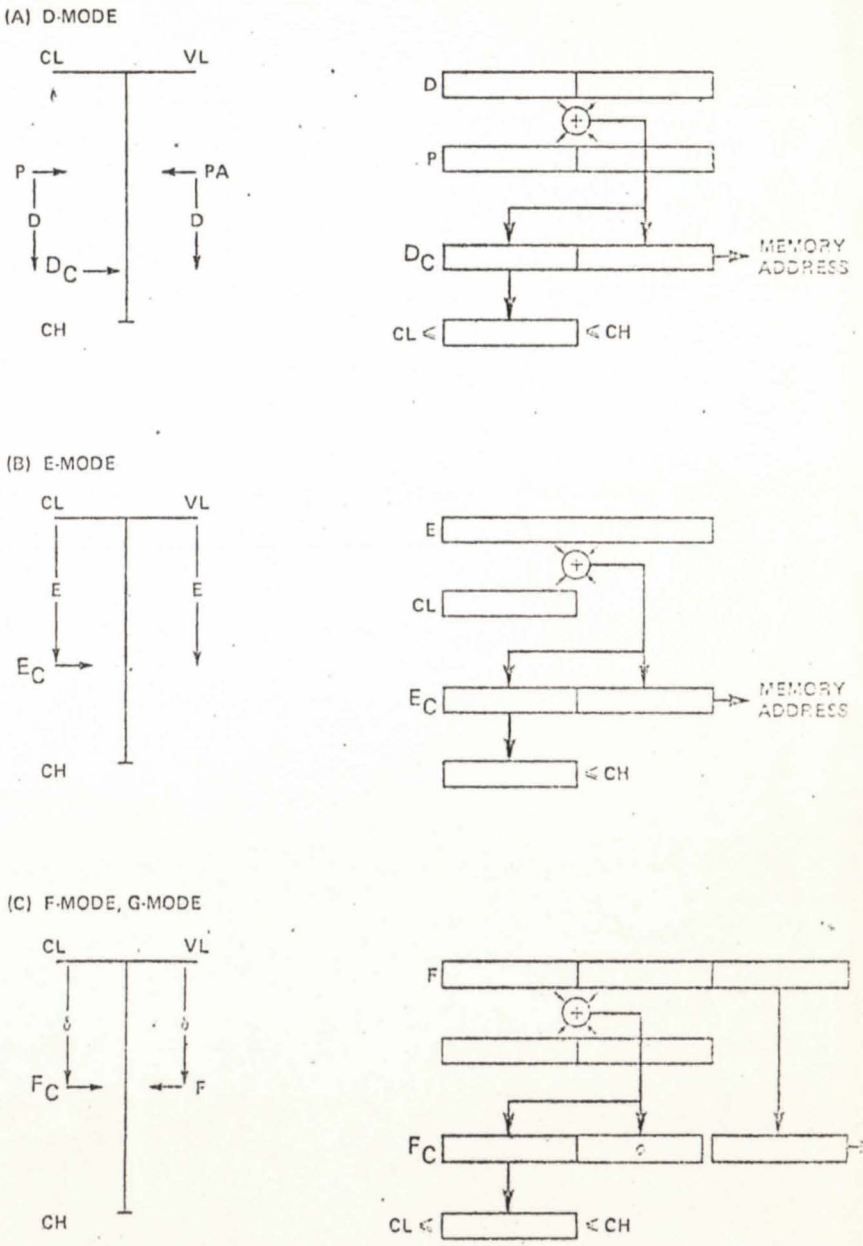

Figure 1. Addressing Modes

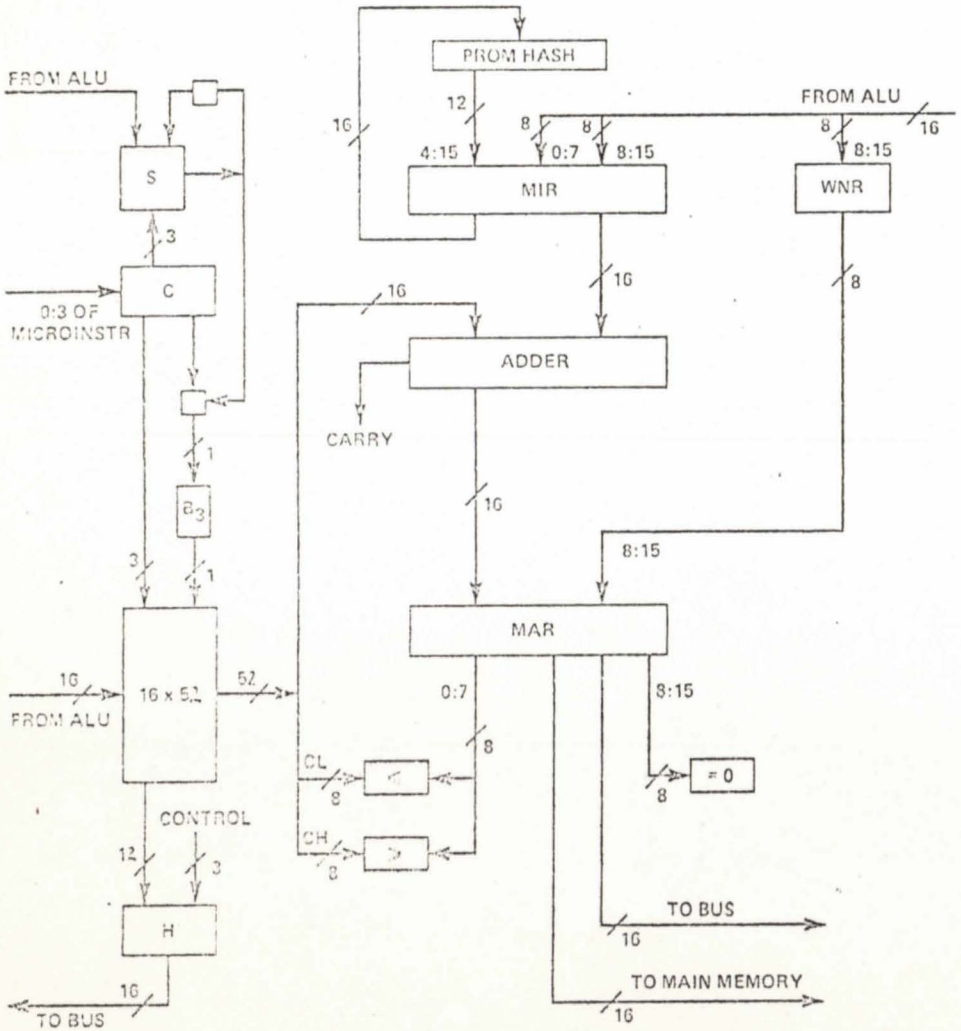

Figure 2. Address Translation Hardware Data Paths