M E M O

To: CSL, Learning Research Group

Date: June 6, 1974

From: Ben Wegbreit

Location: Palo Alto

Subject: Alto Virtual Memory Proposal

Organization: PARC/CSL

Archive category: Alto

1) Summary

I propose implementing a virtual address space of 2↑24 words at the instruction level. All language systems can use this virtual address space. Mapping virtual addresses into physical addresses is done, in the most frequent case, with a small set of non-associative mapping registers.

2) Addressing

Programs run in a 2↑24 bit virtual address space. I'll discuss addressing this space in terms of a Nova-like instruction set not because this is optimal but because its a concrete place to start.

2.1 Instruction Format:

An instruction which references memory has:

I - an indirect bit
X - a base register field (2 bits)
D - a displacement (8 bits)

2.2 Registers:

All registers dealing with addresses (e.g. PC and index registers) are at least 24 bits wide (32 if that proves useful for other reasons).

2.3 Indirection:

Indirected instructions are pre-indexed, not post indexed and the address obtained at the 1st level is interpreted as the first word of a 2 word block containing a 24 bit virtual address. Hence, any word in memory can be addressed.

2.4 Base Register Field Interpretation:

00 - PC 01,10,11 - AC1,AC2,AC3

2.5 Comment:

The proposed scheme works somewhat better if more base registers are used, e.g. 8 or 16. A modified instruction format is then necessary.

3) Mapping Virtual Addresses onto Physical Addresses

3.1 Observation:

The only addresses that can be accessed are those obtained by indirection or thru the base registers. Locality considerations argue that the latter will be the most prevelant mode.

3.2 The Idea:

When a program is making repeated references thru base register X to a virtual page VP, it will get into some core page CP. We introduce hardware which for each base register X gives the current core page CP[X]. Instructions which use base register X are (generally) mapped into core page CP[X].

3.3 Details:

Each of the 4 base registers has associated with it two other registers: last virtual page number (LVP) and core page (CP). Whenever base register X is used to form an address, the following occurs:

virtual address = D + contents of X
virtual page # = high-order 15 bits of virtual address
displacement = low-order 9 bits of virtual address

if virtual page # = LVP[X]
then memory address = <CP[X], displacement>
else base register fault.

In the successful case, mapping requires one comparison and no associative hardware. Indirection is treated as an implicit base register; the 24 bit virtual address is compared against LVP[I], etc.

3.4 Base Register Faults:

A base register fault is caused either because (D + contents of X) is on a different page than contents of X or because the contents of X has been changed since the last memory reference thru X. When a base register fault occurs, a hashing scheme like Peter's ("The Lisp Alto Map", 5/30/70) is used to find the

page if it is in core. (If all languages use the hash mechanism, there is economic incentive to shift more of the work into hardware, e.g. put the hash table into fast memory - around 2↑13 bits for a half-full table.) If the page is not in core, the processor delivers an interrupt; it's up to the language to decide what page(s) should be shoved out. ~ Passed ont to U. prog

3.5 Page Size:

Displacement addressing (2↑8) and page size need not be identical but should be comparable. I suggest 2↑9.

3.6. Generalization

The pair LVP,CP provides a fast way of mapping a single page. If we are willing to complicate everything somewhat, this can be generalized to a set of pages. Define a page group to be a contiguous set of virtual pages residing in a contiguous set of core pages. Associate with each base register CP as above and lower-upper virtual bounds registers LVP and UVP which span a page group. CP[X] serves as a base for the core address if the LVP[X] $\leq$ virtual page # $\leq$ UVP[X].

This has the advantage of allowing chunks of memory to be mapped directly, up to all of core. For example, a program which is preplanned to run in no more than 64K of the address space can set up a single page group of that size.

This has the following disadvantages:

(a) additional hardware, since testing virtual page # against LVP,UVP requires 2 subtractions (in parallel) instead of one comparison and requires forming the core page # by addition instead of concatination.

(b) a more complicated core manager, since it is necessary to allocate variable size core chunks.

(c) somewhat more complex communication between the program and the swapper, since the program must be able to specify segments."

4) Some Usage Modes

4.1 Within a Module:

Code and data can be accessed as D + contents of PC and will generally be mapped directly. — Page boundary probl

4.2 Cross-Module Code Linkage:

Indirect - requiring 48 bits in all.

4.3 Local Variables:

Using one of the base registers as a frame pointer, local variables can be accessed as D + contents of frame register and will generally be mapped directly.

4.4 Lisp CAR and CDR:

Use Danny's hash-linking scheme. Local pointers (non-linked) are converted into virtual addresses when put into a base register by adding to them the contents of the base register which bases their page . Linked pointers are full virtual addresses as in Danny's scheme.

4.5 Array processing:

Sequential accesses to anything, arrays in particular, are generally mapped directly.

5) Drawbacks

5.1. Indirection takes 2 words rather than 1 on the present address structure.

5.2. Unless the page group mechanism is implemented and employed, addresses generated in a random pattern within a core working set (e.g. tree-sorting a 32K word array) must go through the hash lookups since the base register mapping hardware will do little good.

5.3. Since the display hardware runs in the physical address space and the program runs in a virtual address space, it is necessary to be able to establish a correspondence between them. This is an understood problem (Tenex) with a reasonable solution (locked pages). Page groups (c.f. 3.6) help here. However, implementing locked pages is still a complication.

5.4. BCPL is complicated since full addresses become 24 bits, while integers probably should remain 16 bits. Making the distinction would, at the least, introduce some complication with BCPL.

6) Operating System

6.1. Since the virtual memory is large, the operating system can live in the same address space.

6.2. Protection of the operating system from the program or of the program from regions of itself is a separate issue. A write-protect bit for each of the core page registers (c.f., 3.3) can be included, if memory protection seems needed.

M E M O

To: CSL, Learning Research Group, SSL — Date: June 18, 1974

From: Ben Wegbreit — Location: Palo Alto

Subject: Alto Virtual Memory Proposal - Version 2 — Organization: PARC/CSL

Archive category: Alto

1) Summary

I propose that a virtual address space of 2^24 words be implemented at the instruction and micro-instruction level. All language systems can use this virtual address space. Mapping virtual addresses into physical addresses is done, in the most frequent case, with a small set of non-associative mapping registers.

2) Addressing

Programs run in a 2^24 bit virtual address space. I'll discuss addressing this space in terms of a Nova-like instruction set because its a concrete place to start.

2.1 Instruction Format:

An instruction which references memory has:

I - an indirect bit
X - a base register field (2 bits)
D - a displacement (8 bits)

2.2 Registers:

All registers dealing with addresses (e.g. PC and index registers) are at least 24 bits wide.

2.3 Indirection:

Indirected instructions are pre-indexed, not post indexed and the address obtained at the 1st level is interpreted as the first word of a 2 word block containing a 24 bit virtual address. Hence, any word in memory can be addressed.

2.4 Base Register Field Interpretation:

00 - a special 24-bit register GR which can be loaded by special instructions.
01 - PC
10,11 - AC2,AC3

2.5. Data width:

Since addresses are 24 bits wide, this is the standard width of full pointers. Integers should probably remain 16 bits wide. Short pointers (e.g. relative to the first word of their page) should also be supported. Registers dealing with addressing are 24 bits wide. Hence it is necessary to be able to load and store both 16 and 24 bit quantities into and from these registers.

2.6: Non-Nova Instructions:

Mesa and Altolisp will be programmed mainly by micro-interpreters for their specialized instruction set. Hence, the virtual address space must be available at the microinstruction level. In that mode, it may be possible to use more base registers, e.g. 8 or 16, which would make the scheme work somewhat better. The number of registers is limited by bits to address them in micro-instructions and card capacity.

3) Mapping Virtual Addresses onto Physical Addresses

3.1. Observation

Addresses typically are not generated at random but rather are obtained by relatively small changes to a prior address. Examples:

(a) small change to current PC for next instruction and local jumps,

(b) accessing local variables as a small offset from a frame pointer,

(c) incrementing an index register in fetching consecutive words of an array,

(d) in Lisp, car and cdr pointers local to a segment

These prior addresses are often obtained from the registers: PC, and AC's.

3.2. The Idea

Supply with every memory request an indication (access code) of which register was used in generating the address. If the new address is close enough to the prior address, then it will probably be in core already. We

introduce hardware which for each base register X gives the current core page CL[X]. Addresses generated from base register X are generally mapped by CL[X].

Generalize this as follows. Define a page group to be a continuous set of virtual pages residing in a contiguous set of core pages. The program specifies a page group to the page manager by a JSYS-like command. Whenever any page in the group is subsequently referenced, the entire group is obtained. CL maps an entire page group.

3.3. Details:

Each of the base registers has associated with it three other registers

LVP - low virtual page
HVP - high virtual page
CL - low core page

LVP[X] and HVP[X] hold the page number of the low and high pages in the last page group accessed thru base register X. Whenever base register X is used to form an address, the following occurs:

virtual address = D + contents of X
virtual page # = high order 15 bits of virtual address

if LVP[X] ≤ virtual page # ≤ HVP[X]
then memory address = CL[X] + virtual address
else base register fault

Actually, (CL[X] + virtual address) is formed first, then memory fetch is initialized, then the comparison is carried out to see if the word which will be fetched is the right one. Hence, in the successful case, mapping delays the memory access by one add time.

Indirection is treated as an implicit base register; the 24 bit virtual address is compared against LVP[I], etc.

3.4 Base Register Faults:

A base register fault is caused either because (D + contents of X) is in a different page group than contents of X or because the contents of X has been changed since the last memory reference thru X. When a base register fault occurs, a hashing scheme like Peter's ("The Lisp Alto Map", 5/30/70) is used to find the page if it is in core. (If all languages use the hash mechanism, there is economic incentive to shift more of the work into hardware, e.g. put the hash table into fast memory. If the page is not in core, the processor delivers an interrupt; it's up to the language to decide what page(s) should be shoved out.

3.5 Page Size:

Displacement addressing (2¹8) and page size need not be identical but should be comparable. I suggest 2¹9.

3.6. Non-Nova Instructions:

In non-Nova mode, the correspondence between base register and 24-bit virtual address must be done at the micro-instruction level, e.g. via some bits of the micro-instruction, or via the contents of some machine register.

3.7 Disk management:

It is desirable to have page groups be identical with partitions. That is, given the address of the first page in a group on the disk, the addresses of the other pages be determined by a simple computation and pages should be placed so as to minimize time to read them all in.

4) Some Usage Modes

4.1 Within a Module:

Code and data can be accessed as D + contents of PC and will generally be mapped directly.

4.2 Cross-Module Code Linkage:

Indirect - requiring 48 bits in all.

4.3 Local Variables:

Using one of the base registers as a frame pointer, local variables can be accessed as D + contents of frame register and will generally be mapped directly.

4.4 Lisp CAR and CDR:

Use Danny's hash-linking scheme. Local pointers (non-linked) are converted into virtual addresses when put into a base register by adding to them the contents of the base register which bases their page . Linked pointers are full virtual addresses as in Danny's scheme.

4.5 Array processing:

Sequential accesses to anything, arrays in particular, are generally mapped directly.

4.6. Small programs:

A program which is preplanned to run in no more than 64K of the address

space can set up a single page group of that size and have all memory accesses mapped directly.

4.7. Bit map for the display:

A page group.

5) Drawbacks

5.1. Indirection takes 2 words rather than 1 on the present address structure.

5.2. BCPL is complicated since full addresses become 24 bits, while integers probably should remain 16 bits. Making the distinction would introduce substantial complication with BCPL. (c.f. 2.4)

5.3. Additional hardware is required to make the mapping fast enough to be acceptable.

5.4. When a base register fault occurs, the mapping registers must be loaded after hash-table lookup. Hence, in the worst case, if every reference faulted, the proposed scheme would run slower than using hashing alone.

6) Operating System

6.1. Since the virtual memory is large, the operating system can live in the same address space.

6.2. Protection of the operating system from the program or of the program from regions of itself is desirable, but a separable issue. Using this proposal, it suffices to add Tenex-style bits for read, write, and execute access to the mapping registers (c.f. 3.3) and maintain them in the hash table (c.f. 3.4) to be loaded on base register fault.

7) Separation

There are roughly six ideas combined here, pulling them apart may be helpful:

1) large address space (greater than 2↑16)

2) access code to specify which mapping register will probably map the address.

3) Peter's hash table when (2) gets a fault

4) page groups to get: each mapping register to map several pages, preloading and escape back to the bare addressing structure.

5) correspondence between physical disk allocation and page groups to speed up access to all of group

6) access protection

MEMO

To: CSL, SSL Date: June 25, 1974

From: Ben Wegbreit Location: Palo Alto

Subject: Alto Virtual Memory Proposal - Version 3 Organization: PARC/CSL

File: BWALTO
Archive category: Alto

1) Summary

It is proposed that a virtual address space of 2¹24 words be implemented at the micro-instruction level. All language systems can use this virtual address space. Mapping virtual addresses into physical addresses is done, in the most frequent case, with a small set of non-associative mapping registers.

2) Addressing

Programs run in a 2¹24 word virtual address space. I'll discuss addressing this space at the micro-instruction level.

2.1. Base Registers

Some number (8 or 16) of 24-bit base registers are added to the Alto (on the memory interface board).

2.2. Address Formation

Addresses are of two sorts:

(a) normal mode: 16-bit displacement from the contents of one of the base registers.

(b) full address: a 24-bit full address

In normal mode, a micro-instruction referencing memory supplies a 16-bit displacement as the output of the ALU and a 3 or 4 bit field (specifying base register number) in the micro-instruction. The micro-instruction field is ORed with the contents of a special loadable S register to get the base register number. The displacement is added to the contents of the base register to get the virtual address.

2.3. Manipulating Base Registers

Micro-instructions are provided for loading base registers and their associated mapping registers (c.f. Section 3.2). Additionally, the following may be useful: incrementing, adding 16-bit displacements, and storing.

3) Mapping Virtual Addresses onto Physical Addresses

3.1 The Idea When an address is formed as the displacement from a base register, if the new address is close enough to the prior contents of the base register then it will probably be in core already. We introduce hardware which for each base register X gives the current core page = CL[X]. Addresses generated from base register X are generally mapped by CL[X].

Generalize this as follows. Define a page group to be a continuous set of virtual pages residing in a contiguous set of core pages. The program specifies a page group to the page manager by a JSYS-like command. Whenever any page in the group is subsequently referenced, the entire group is obtained. CL maps an entire page group.

3.2. Details:

Each of the base registers has associated with it a block of three other registers, mapping registers.

LVP - low virtual page
HVP - high virtual page
CL - low core page

LVP[X] and HVP[X] hold the page number of the low and high pages in the last page group accessed thru base register X. Whenever base register X is used to form an address, the following occurs:

virtual address = D + contents of X
virtual page # = high order 15 bits of virtual address

if LVP[X] ≤ virtual page # ≤ HVP[X]
then memory address = CL[X] + virtual address
else base register fault

Actually, (CL[X] + virtual address) is formed first, then memory fetch is initiated. Hence, in the successful case, mapping delays the memory access by one add time. The comparison is done last and is actually performed more simply - Chuck Thacker worked out a method that requires only one comparison for bounds.

3.3. Base Register Faults

A base register fault is caused either because (D + contents of X) is in a different page group than contents of X or because the contents of X has been

changed since the last memory reference thru X. When a base register fault occurs, a hashing scheme like Peter's ("The Lisp Alto Map", 5/30/70) is used to find the page if it is in core. Call this a hashed map. (If all languages use the hash mechanism, there is economic incentive to shift more of the work into hardware, e.g. hardware hash. If the page is not in core, the processor delivers an interrupt; it's up to the language to decide what page(s) should be shoved out.

3.4 Page Size

Displacement addressing (2¹⁸) and page size need not be identical but should be comparable. I suggest 2¹⁹.

3.5. Disk management

It is desirable to have page groups be identical with partitions. That is, given the address of the first page in a group on the disk, the addresses of the other pages be determined by a simple computation and pages should be placed so as to minimize time to read them all in.

3.6 Full Addresses

A full 24-bit address can be delivered with a base register number X, to be interpreted as: map the address with the bounds registers of X. This is intended to provide a way of using a previously set up base register as a hint for mapping a full address without affecting its contents. Its utility is somewhat marginal.

4) Discussions of Mapping

The mapping hardware proposed here works well only if most memory references are issued as displacements from previously loaded base registers and most of these are mapped directly. The hash table lookup takes around 2.8 ms in the best case (with current hardware). A ratio of around 10:1 memory references to hashed map references is required if this is to be practical. Howard Sturgis, Peter Deutsch, and Jim Mitchell are collecting statistics for the dynamic behavior of BCPL, Bytelisp, and Mesa.

These statistics provide memory references more or less unambiguously, but the number of hashed map references depends on how the language processors use the base registers. In this regard, there are three classes of memory references:

(1) Easily reprsented as base register + displacement, since the base register contains a fixed, identifiable component of the pseudo-machine supporting the language, e.g. PC, stack pointer(s), global pointer(s).

(2) possibly represented as base register + displacement, depending on the compilation of the language, e.g. array access, accesses to several fields of a pointer-based structure, cars and cdrs on the same page.

(3) not represented as base register + displacement, e.g. address of a procedure in another segment, a newly constructed pointer.

Class (3) must be hash mapped, class (1) will be directly mapped in most implementations; class (2) presents an uncertainty. Use of base registers here is similar to the technical problems of using index registers well in normal compilation; the incentive for doing so is substantially greater. It appears that this will be easier to do for array processing (e.g. in Mesa) than for list processing (e.g. in Bytelisp) since more is done in-line (procedure calls are usually treated as destroying state information).

5) Some Usage Modes

5.1 Within a Module:

Code and data can be accessed as 0 + contents of PC and will be mapped directly.

5.2 Cross-Module Code Linkage:

Requires a full 24-bit pointer and a hashed map.

5.3 Local Variables:

Using one of the base registers as a frame pointer, local variables can be accessed as 0 + contents of frame register and will be mapped directly.

5.4 Lisp CAR and CDR:

Use Danny's hash-linking scheme. Local pointers (non-linked) are converted into virtual addresses when put into a base register by adding to them the contents of the base register which bases their page . Linked pointers are full virtual addresses as in Danny's scheme.

5.5 Array processing:

Sequential accesses to anything, arrays in particular, can be mapped directly, but this depends on the language processor (c.f. Section 4).

5.6. Small programs:

A program which is preplanned to run in no more than 64K of the address space can set up a single page group of that size and have all memory accesses mapped directly.

5.7. Bit map for the display:

A page group.

6) Nova Emulation Mode

It is desirable to let existing Nova machine language programs run unchanged and also to allow use of the 2↑24 word address if wanted. The following is a compromise.

6.1. Instruction Format

An instruction which references memory has:

I - an indirect bit
X - a base register field (2 bits)
D - a displacement (8 bits)

6.2 Registers:

All registers dealing with addresses (e.g. PC and index registers) are 24 bits wide.

6.3 Base Register Field Interpretation:

00 - a special 24-bit register GR which can be loaded by special instructions.
01 - PC
10,11 - AC2,AC3

6.4. Data width:

Since addresses are 24 bits wide, this is the standard width of full pointers. Integers should probably remain 16 bits wide. Short pointers (e.g. relative to the first word of their page) should also be supported. Registers dealing with addressing are 24 bits wide. Hence it is necessary to be able to load and store both 16 and 24 bit quantities into and from those registers.

6.5. Indirection

Indirected instructions are interpreted as currently on the Alto. Hence, only 2↑16 words can be accessed this way.

6.6. Register Overflow

Registers used in Nova emulation mode which overflow 16 bits will behave differently. Sorry.

6.7. BCPL BCPL is complicated since full addresses become 24 bits, while integers probably should remain 16 bits. Making the distinction would require changes to the BCPL compiler and would make the language into a non-standard dialect.

7) Operating System

7.1. Since the virtual memory is large, the operating system can live in the same address space.

7.2. Protection of the operating system from the program or of the program from regions of itself is desirable, for the usual reasons. Using this proposal, it suffices to add Tenex-style bits for read, write, and execute access to the mapping registers (c.f. 3.2) and maintain them in the hash table (c.f. 3.3) to be loaded on base register fault.

8) Separation

There are roughly six ideas combined here, pulling them apart may be helpful:

1) large address space (greater than 2↑16)

2) access code to specify which mapping register will probably map the address.

3) Peter's hash table when (2) gets a fault

4) page groups to get: each mapping register to map several pages, preloading and escape back to the bare addressing structure.

5) correspondence between physical disk allocation and page groups to speed up access to all of group

6) access protection

9) Unresolved Questions

9.1. Usage

(a) How effectively can the language processors use the base registers for class (2) references?

(b) What percent of memory references will be mapped directly?

9.2. Design Parameters

(a) Is the full 24-bit address worth providing?

(b) How extensive should the arithmetic operations on base registers be?

(c) How many base registers should there be?

(d) Should the 16-bit displacement be treated as a signed-integer (sign bit extended) to give relative addressing to either side of the base register?

M E M O

To: CSL/SSL

Date: October 4, 1974

From: Ben Wegbreit

Location: Palo Alto

Subject: Multiple Base Registers for the Alto Virtual Memory

Organization: PARC/CSL

File: <VANJEPMOND>BWMBR.PUB;10

1. SCOPE

This is an improvement to the design in "Alto Virtual Memory Update", Sept. 13, 1974. A subsequent memo will consolidate the two previous memos with this one to produce a final design.

2. SUMMARY

A proposed reorganization of the base registers, along with a small amount of additional hardware, can be used to reduce the address translation fault rate by approximately a factor of two.

3. THE IDEAS

In the previous design, a single base register was associated with each logical function. The new proposed design allows multiple (N=2) base registers to be associated with a logical function. Of the pair of registers, one is current at any given time while the other serves as an alternate to be tried, under appropriate circumstances, when the current one is incorrect. State information in the memory interface determines which one of a pair is deemed to be current and is kept updated so that successive references to the same page group through a pair try the current one first.

As an example, consider the planned design for Mesa ("The Implementation of Mesa on Alto", 8/21/74). Faults can be caused by

(1) user-computed addresses
(2) non-local procedure calls

The faults for each of these may be decreased as follows:

(3.1) User-computed Addresses

The previous design uses one base register for each of read and write. The new proposed design uses a pair of base registers for each of read and write: a current register and an alternate. Consider reads; writes are analogous. When a read request occurs in 24-bit addressing mode, then the current read register is tried first; 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 read pair. If the alternate fails, then a fault occurs and,

in processing the fault, the (old) alternate is reloaded; this implements LRU replacement.. To distinguish this mode of addressing from the 24-bit mode described in the previous memo, call this a G-mode memory reference.

Trying the alternate base register takes 2 minor cycles. Loading 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, .88, and .44 (three different programs); BCPL gives .63. That is, trying the alternate is a clear winner in time.

(3.2) Transfers of Control

Consider transferring control from procedure P to procedure Q. Three cases arise:

(1) Q is known to be in the same page group; example: Mesa intra-module call
(2) Q is known to be in a different page group; example: Mesa inter-module call.
(3) uncertainty; example: Mesa procedure-valued parameters; also: Bytelisp without block compilation.

Consider imitating (3.1) and always trying a current base register for control and, failing that, trying an alternate. The results would be:

(1) succeeds on current
(2) fails on current; may succeed on alternate
(3) may succeed on either current or alternate

Some measurements of Mesa programs show that with this policy, for three different programs, the percent of control transfers mapped by current are: 40.9, 28.4, and 40.7 while the percent mapped by the alternate are: 40.0, 69.1, and 39.6 respectively. Note that the sums of current plus alternate are somewhat more consistent: 80.9, 97.5, and 80.3 respectively. (It is conjectured that the somewhat anomalous behavior of the second program - a text formatter - is caused by very frequent calls out of "current" module to the string-manipulation module).

This differs somewhat from G-mode in that to try the current base register and then the alternate requires address translation, but not actually going to the memory. If a fault does not occur, then the contents of MAR after translation is the new value of P for the (possibly new) current base register for control. This should be loaded into P(control-current) to produce the new PC base.

(3.3) Other Applications of Using Base Registers in Pairs

The above two examples both involve translation of full 24-bit virtual addresses. Some advantages of using registers in pairs with a current and alternate apply in D and E mode, in language implementations other than Mesa. In Mesa address translation faults are assumed to occur only for the two reasons given above 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 will be handled properly and efficiently, as discussed in (3.2). 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 ~~or E-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 (3.2), consider testing I-alternate to see if it does, in fact, map the new instance base and changing the state of the I-pair if it does, so that the former I-alternate becomes the new I-current. In this way, the micro-instruction still specifies only I, while state information associated with I-pair distinguishes which of the two I-registers is intended.

(3.4) Fallback to Non-Paired Organization

For maximum flexibility in base register usage, it seems desirable to allow optional usage of the 16 base registers individually.

4. IMPLEMENTATION

A memory request is specified by an 8-bit specifier, broken down as follows:

4-bit C field - giving partial specification of the base register
2-bit mode field - D-mode, E-mode, etc.
2-bit usage class - Read,Write,etc.

(4.1) Addressing Modes

The memory interface can be used in 4-modes:

D - displacement from P(B)
E - displacement from CL(B)
F - 24-bit virtual address to be found in the page group described by B
G - 24-bit virtual address to be found in the page group described by B or B's alternate.

(4.2) Specification of Base Register Number

In each case, B is specified in the following way. The micro-instruction supplies a 4-bit C 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 -- set the low order bit of C to zero when assembling the micro-instruction
(2) to obtain non-paired addressing - 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. In mode G, B is tried and if it fails then B's alternate is formed by complementing S(B[0:2]) and trying the (new) resultant B. Hence, in G-mode, the memory interface reports a fault only if B and B's alternate both fail. Further, in G-mode, if a fault does not occur then the new state of the memory interface is such that the current element of the pair is the one which succeeded. The state bits to keep track of the current element of each pair are stored in a special 1x8 memory.

(4.3) Usage Classes:

Orthogonal to the four modes are four usage classes:

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

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

(1) implementing (3.2) and (3.3) above without initiating and hence waiting for the unneeded memory cycle

(2) changing the value of a base pointer P within a page group, e.g. in moving the frame pointer down on the stack for a simple hierarchical procedure call.

5. PERFORMANCE

Simulations of Mesa have been run under the previous and the (new) proposed use of paired-base registers. In the paired simulations, pairs were used for read, write, and code. The resulting fault rates are as follows for three programs (several million instruction in each case):

	previous	new
compiler:	4.6%	2.1%
text formatter:	2.3%	0.1%
analyzer/compiler:	5.1%	2.3%

From statistics previously issued on BCPL, it is possible to compute the affect in one experiment of using paired base registers for user-computed addresses only, with a single base register for control.

	previous	new
experiment #1	4.9%	3.1%

(The effect of paired base registers for control and the effect on the other two experiments can't be computed from the previously issued data.)