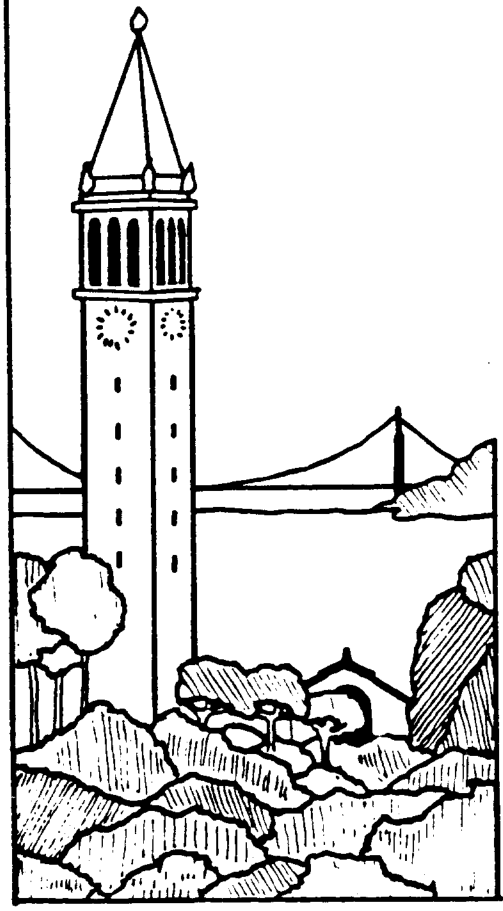
The Memory Architecture and the Cache and Memory Management Unit for the Fairchild CLIPPER Processor

James Cho, Alan Jay Smith, and Howard Sachs



Report No. UCB/CSD 86/289

April 1986

Computer Science Division (EECS)
University of California
Berkeley, California 94720

The Memory Architecture and the Cache and Memory Management Unit for the Fairchild CLIPPER™ Processor

James Cho*, Alan Jay Smith†# and Howard Sachs*

Abstract

The Fairchild CLIPPER is a new high performance three chip module consisting of a microprocessor chip and two cache and memory management (CAMMU) chips, mounted on a small PC board. CLIPPER implements a new instruction set architecture which has been designed for high performance, convenient programmability, broad functionality and sufficient architectural "openness" to permit future evolution and a variety of implementations.

The CLIPPER memory architecture is a separate 32 bit logical address space for each of the user and supervisor, with facilities for transferring information from one to the other. Virtual memory support is provided by a memory management unit on each CAMMU, each of which includes a translator that maps 32 bit virtual addresses through a two level page table to 4096 byte pages, and a 2-way set associative 128 entry TLB. There is a 4096 byte cache for each of instructions and data; it is organized as two way set associative with 16-byte lines and with LRU replacement within each set. The caching policy (write-through, copy back, non-cacheable) may be specified on a page basis, as may the protection modes (read, write, execute, by user and supervisor). The bus protocol and interface provides a mechanism to maintain cache consistency when the bus is shared by multiple processors and I/O devices with overlapping physical address spaces. There is a translator, cache and TLB implemented on each of the two CAMMU (cache and memory management unit) chips.

In this paper, we discuss, in some detail, the memory architecture and the cache and memory management units of the Fairchild CLIPPER. Timing for operations and performance estimates are provided. There is some discussion as well for the various implementation decisions and the tradeoffs involved.

---

*Fairchild Semiconductor Corporation, 4001 Miranda Avenue, Palo Alto, Ca., 94304.

†Computer Science Division, EECS Dept., University of California, Berkeley, Ca. 94720

#Partial support for the research results and performance studies presented here was provided by the National Science Foundation under grant DCR-8202591, and by the Defense Advance Research Projects Agency (DoD), under Arpa Order No. 4871, Monitored by Naval Electronic Systems Command under Contract No N00039-84-C-0089. Some of the computer time has been provided by the Stanford Linear Accelerator Center under Department of Energy contract DE-AC03-76SF-00515.

1. Introduction

Rapid advances in VLSI circuit density and speed have made it possible to obtain super-mini and mainframe levels of performance in a microprocessor. The Fairchild CLIPPER reaches this performance class by using an efficient, simple, load/store architecture and a 2 micron CMOS process. The CLIPPER consists primarily of three chips, a processor chip and two (identical) cache and memory management unit chips (CAMMU), one for instructions and one for data. These three chips, along with (33MHz - 30ns cycle) clock generation circuitry, are packaged on a 3.5 x 4.5 inch printed circuit board, and interface to systems through a 96 pin DIN connector; the overall system architecture is shown in figure 1. The CPU chip has 132,000 transistors and each CAMMU chip has 357,000; each chip (CAMMU and CPU) is 156,000 square mils. The data paths between the processor and the CAMMU units and to the system bus are all 32 bits, and the processor provides a 32 bit virtual memory address space. More detail about the system architecture and implementation beyond that here are provided in [Sach85a, Sach85b, Fair85a,b, Neff86]. Some additional information on the physical implementation of the CAMMU appears in [Cho86].

The instruction set architecture has been designed to reflect the latest thinking of much of the computer architecture research community, which is that simplified, load-store architectures permit high performance while minimizing design costs and complexity [Henn84, Patt85, Radi82]. We refer to this type of architecture as "RISC-like", in reference to Patterson's "Reduced Instruction Set Computer", although the original RISC incorporated both features and restrictions not found in CLIPPER. The CLIPPER instruction set architecture, described in detail in [Fair85a,b, Sach86], contains the usual set of basic instructions, including loads, stores, moves (including a string move), integer arithmetic, floating point arithmetic (including on-chip floating multiply and divide, single and double precision), conversion, shifts and rotates, logical operations, compares, stack instructions, and conditional and unconditional branches; a variety of addressing modes are provided. Data types include word (4 byte), half word and byte operands, and single and double precision floating point, with arithmetic executing according to the IEEE 754 Floating Point standard. There are 16 (32 bit) user registers, 16 (32-bit) supervisor registers and 8 (64 bit) floating point registers.

This set of instructions and addressing modes permits entirely hardwired decoding and sequencing logic, and a three stage main pipeline. The processor chip contains both hardwired integer and floating point arithmetic, and also a small ROM, which is used to handle traps and interrupts and implement a useful set of "complex instructions." Unlike some of the academic research projects

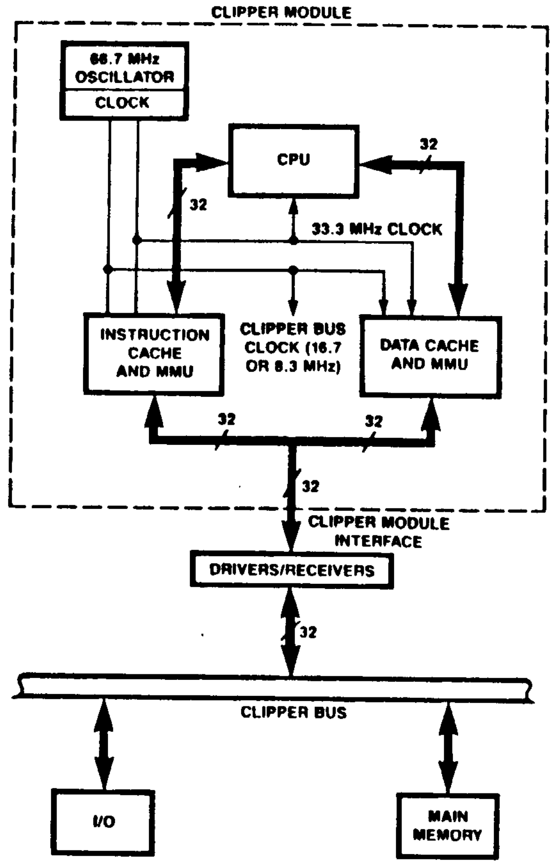
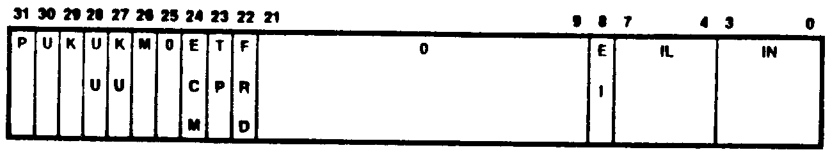


Figure 1: CLIPPER Module Block Diagram



FIELD	MEANING
IN	INTERRUPT NUMBER
IL	INTERRUPT LEVEL
EI	ENABLE INTERRUPTS
FRD	FLOATING REGISTERS DIRTY
TP	TRACE TRAP PENDING
ECM	ENABLE CORRECTED MEMORY ERROR

FIELD	MEANING
M	MAPPED MODE
KU	USER PROTECT KEY
UU	USER DATA MODE
K	PROTECT KEY
U	USER MODE
P	PREVIOUS MODE

Figure 2: System Status Word

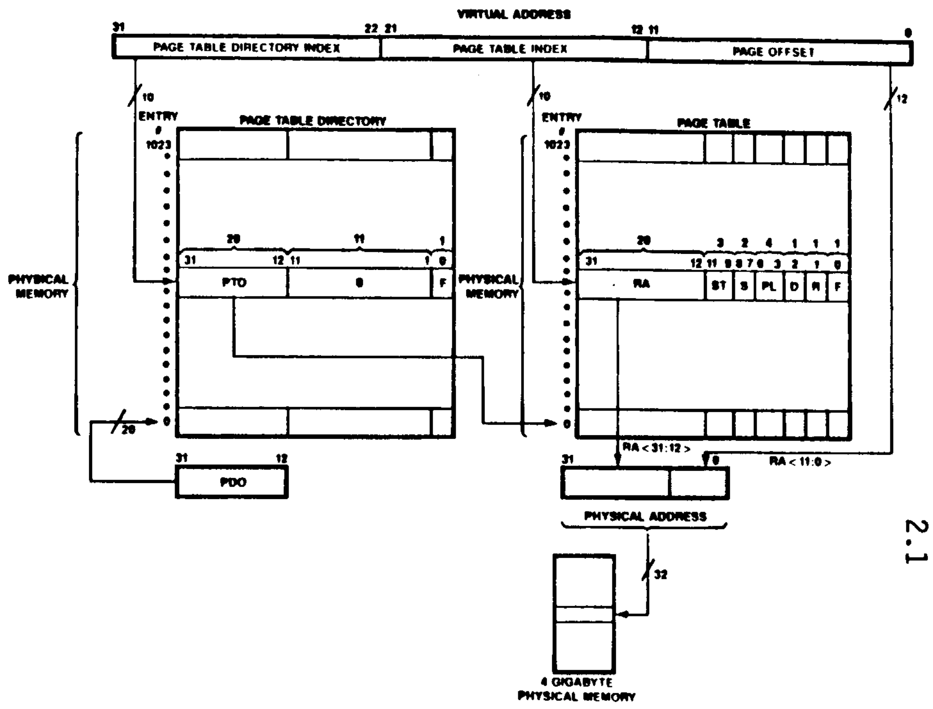


Figure 3: Virtual to Real Address Translation

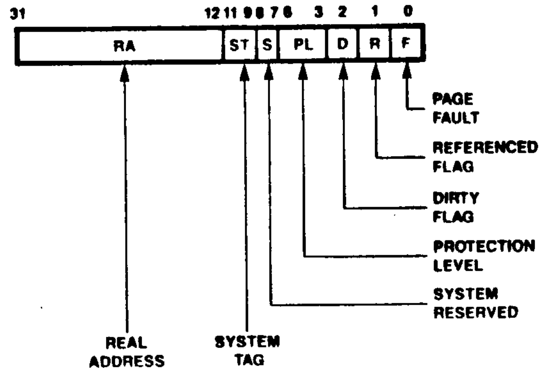


Figure 4: Page Table Entry Format

referenced above, sufficient logic is provided on-chip to resolve register and functional unit access conflicts, so that the burden of pipeline sequencing is not placed on the compiler or programmer.

The most important and in fact fatal limitation of most early computer architectures has been limited address space addressing; the non-extensibility of the 16-bit architecture of the PDP-11^{TM} forced DEC to develop the VAX^{TM} [Levy80], which uses 32 bits for addressing, and the 24-bit limitation of the IBM 360/370 architecture [IBM76] forced IBM to go to the 31-bit XA architecture. Keeping this lesson in mind, CLIPPER has 32 bits of physical addressing and separate 32 bit address spaces for the user and supervisor when in mapped (virtual memory) mode. Since the high order address bits are not used to partition the address space, there is the potential for evolution to larger address spaces as technology advances. A mechanism is provided (see section 3.3) to move information between the user and supervisor address spaces.

The single most important problem in obtaining high performance from a bus based system is memory access time and bandwidth [Mate84], and this problem is further magnified by the bandwidth limitations of the pins on a single chip processor. CLIPPER has therefore been designed as an integral three chip set, with a processor chip and two cache and memory management (CAMMU) chips. The CAMMU contains the translator, TLB (translation look-aside buffer), cache, and the logic to interface to the system (CLIPPER) bus. The latter has facilities and mechanisms for maintaining consistency when the bus is shared by multiple processors and I/O devices.

In this paper we explain in some detail the memory architecture of CLIPPER and its implementation via the cache and memory management unit. In the next section, we provide a somewhat more detailed, but still brief, overview of the CLIPPER functional architecture. The memory architecture is described in detail in section 3. The CAMMU chip and its components are considered in detail in section 4 and in section 5 we discuss the processor / CAMMU interface. Section 6 considers the system bus interface, and in section 7 we provide timing information and performance estimates. Some discussion of design tradeoffs appears in section 8.

2. Architectural Overview

In this section, we provide an additional overview of the CLIPPER architecture. We restrict our discussion primarily to those features that affect the memory architecture and/or the CAMMU. Further information on the instruction set architecture and implementation of the processor chip can be found in [Fair85a,b, Sach85a,b,86].

In figure 1 is a diagram of the **CLIPPER module**, which is a small PC board with a 96 pin connector. It contains a processor chip (the CPU), two cache and memory management chips (Instruction CAMMU and Data CAMMU), a clock, and circuitry to interface to the system (CLIPPER) bus. The clock drives the processor and CAMMU chips and the busses between the processor and CAMMUs at 33.3 MHz, i.e. with a 30ns cycle time; the main system bus runs at half that speed. The interface between the CAMMUs and the processor consists of 46 lines, including a 32 bit bi-directional address and data path.

The **data paths** in CLIPPER are all **32 bits**, which is the **word size**. Control flow through the pipeline is in units of 16- bit parcels. **Data types** include 8 bit bytes, 16 bit half words, 32 bit words, and 32 and 64 bit floating point numbers. All data items must be stored on a boundary divisible by their size; instructions must start on a 2 byte boundary; this alignment restriction considerably simplifies loads, stores and instruction fetches. **Addressing** is consistent, with bits numbered from 0 from the least significant, bytes from 0 from the least significant, and words in such a way that the high order bit of word n adjoins the low order bit of word n+1. Bytes, halfwords and words can be treated as signed or unsigned integers. All integer arithmetic and logical operations are register to register and operate on full words. Stack instructions push and pop single words, and multiple words by saving and restoring a range of either fixed or floating registers. Data transfers to/from the processor chip must be done via stack instructions or loads and stores (byte, half word, word, double word). Three string operations are available: compare characters, initialize characters and move characters. There is also a Test and Set instruction which references memory directly.

The CPU contains three sets of **registers**: 16 (32 bit) user registers, 16 (32 bit) supervisor registers, and 8 (64 bit) floating point registers. All data transfers to/from the processor chip are effectively loads and stores; unlike many other machines, there are no other memory to register operations. There is also a program counter (PC) which may be used in address computations, and two status and control registers, the **PSW (program status word)** and the **SSW (system status word)**. The PSW holds flags for condition codes, exceptions, and various trap enables. The SSW (figure 2) includes fields for interrupt number, interrupt level, interrupt enables, a dirty bit for the floating point registers (to reduce context switch time), a bit (ECM) to trap on corrected memory errors, a bit (M) to specify whether mapped (virtual memory) mode is in use, a bit (U) to specify user/supervisor state, a protect key bit (K), a user protect key bit (KU) and a user data mode (UU) bit. Several of these bits in the SSW relate to the memory architecture and their use is described further in section 3.4. The SSW can be written from supervisor state only.

There are 9 **instruction addressing modes**, which in all cases generate 32 bit addresses. These modes may use as part of the address computation one or two registers, the PC, and 16 and 32 bit displacements. There are unused address modes (i.e. there are bits left to specify additional address modes), and some of the address modes have data fields which are not used; thus there is sufficient flexibility in the CLIPPER architecture to permit the number of address bits to be increased at some future time.

As noted above, almost all instructions and the processor control logic are hardwired, in order to obtain high performance. There are some operations, however, that are too complex to hardwire within the constraints of the silicon area and process available, but that are frequent enough and useful enough to include in the instruction set. For that reason, a small, on-chip ROM, called the **Macro Instruction ROM, (MIROM)** implements a few instructions, including context switch operations, floating point conversions and the string operations (initialize, compare, move). This is not, it is important to note, microprogramming; the operations in the MIROM use the standard hardwired instruction set. Use of the MIROM has several advantages: delays due to instruction fetch are eliminated, since the instructions are already on-chip; code density is increased, which saves memory, cuts memory traffic and lessens the frequency of cache misses; and programmability improves since neither the compiler nor programmer has to generate long code sequences for frequent operations. The key point here is that the MIROM instructions are only those frequent enough to require such implementation; CLIPPER does not attempt to compute polynomials or manipulate queues in one instruction. The amount of MIROM, therefore, is only 47 bits x 1K, and thus has only a small impact on the allocation of silicon area; there is no need to dynamically load microcode or have a large control store.

The system has provision for 256 **interrupts** at 16 priority levels, 128 **system calls**, and there are 18 **traps**. A trap, interrupt or supervisor call causes a context save, a switch to supervisor state, and an indirect branch through a low area of memory to the appropriate routine. Further discussion of that low area of memory is provided in section 3.2.

In this paper, we are concerned primarily with the CLIPPER memory functional architecture and implementation, and in section 3, we discuss that in more detail.

3. Memory Architecture

3.1. Addressing

As described above, all memory addresses in CLIPPER are 32 bits, which permits 4 gigabytes (4,294,967,295 bytes) to be addressed. It is possible for this address to be interpreted as a real address or as a virtual address. When the M (mapped) bit in the System Status Word (SSW) is unset, addresses are considered to be real, and are not translated. Some of the implications of operating in unmapped mode are discussed in section 4.2.2.

The normal mode of operation for CLIPPER is expected to be that in which memory mapping is employed; the generated addresses are virtual addresses. The CLIPPER mapping mechanism is **paging, using 4Kbyte (4096 byte) pages**, and almost all memory related functions (protection, caching policy, data sharing) are on a page basis. Segments are not supported by the hardware, although they may be simulated by the software, provided the segments are an integral number of pages in length; thus no artificial limitations are placed on the use of the address space as with some other machines such as the Z8000 [Peut79].

Virtual addresses in CLIPPER are translated, via a two level page table, into real addresses; the virtual to real address translation process is diagramed in figure 3. A **PDO (page directory origin)** register (located, as described below, in the CAMMU) contains the address of the **page table directory (PTD)**; since the PTD is 4Kbytes long and starts on a page boundary, the low order 12 bits of its address are always zero and the PDO is only 20 bits long. Each PTD entry is one word long, and contains a (possibly invalid) pointer to a page table for that portion of the address space; the low order (F - fault) bit of the PTD entry is used to specify whether that entry contains a valid pointer (see figure 3). If the bit indicates a fault, then there is no valid page table resident in memory for that portion of the address space. The high order 10 bits of the virtual address are used to select the PTD entry.

Bits 12-21 of the virtual address are used to select an entry in the **page table (PT)** that has been located via the PTD. Each PT contains 1024 entries, each consisting of the fields diagramed in figure 4. The 20 bit real address field in the page table entry gives the real memory address of the 4Kbyte page. (There is no inherent reason why larger physical address spaces couldn't be used, but the lack of extra bits in the PT entry implies either a change in the page table layout or some means to get extra bits, such as from the page directory. Another approach to a larger address space is to increase the page size, which is a possibility for future CLIPPER implementations.) There are also the system tag (ST, section

4.4.3), the protection level (PL, section 3.4), the **dirty flag** (D), the **referenced flag (R)** and the **fault (F) bit**. The D bit is set when the page is first written to, to indicate that the only valid copy of the page is in memory, not on disk. The R bit is set when the page is referenced, and is used by the paging algorithm to replace recently unreferenced pages. The F bit is used to cause a page fault when the page is missing from main memory. As explained below (sect 4.2), most of the fields of the PT entry are cached in the TLB for fast access.

There are two **PDO (page directory origin)** registers. One contains a pointer to the base of the page directory for the supervisor, and the other contains a pointer to the base of the page directory for the current user process. Changing the contents of the user PDO is sufficient (along with partial clearing of the TLB, as explained in sections 4.2.1, 4.5) to change user process address spaces. (A process swap, of course, will normally also involve register saves and restores, etc.) The U bit (user mode flag) in the SSW specifies which PDO to use in calculating a real address from a virtual one. Because entirely separate page tables are used by each user and the supervisor, no address bits are used for this purpose, in contrast to the design of the DEC VAX^{TM} [Levy80], where the high order two address bits are used to partition the address space. In that machine, the high order half of the address space (bit 31 = 1) is reserved for the system, and the process space (user address space) is further partitioned into the "program portion" and the "control region", the latter being used principally for stacks.

3.2. Hardwired Memory Mapping

The low order 8 pages of the supervisor virtual address space are permanently mapped, via hardwired random logic in the CAMMU. A table of these translations is in table 1, and diagrams showing the same material are in figure 5. As can be seen, virtual pages 0...3 are mapped to physical pages 0...3. Pages 4 and 5 are used for I/O space, and are translated to physical pages 0 and 1, but with the fact that this is an I/O operation encoded in the system tag (ST); the ST field is placed on the system bus for bus accesses. Pages 6 and 7 are mapped to physical pages 0 and 1 also, but with the system tag bits (ST) specifying that the address refers to the "boot ROM", which is used for power up initialization. (The boot ROM is attached to the system bus.) The PL (protection level) bit values are also provided by the fixed logic; the other page table entry bits are not relevant, and are not provided.

There are a number of reasons why the low order part of the supervisor address space is permanently mapped. First, a mapping for boot ROM is essential to starting the system, and the CLIPPER initialization logic starts by fetching

Virtual Page Number	Virtual Address	Real Address	Real Address Space	System Tag (ST)	Function
0	0xxx	0xxx	main memory	1	interrupt and trap vectors
1	1xxx	1xxx	main memory	2	general purpose
2	2xxx	2xxx	main memory	3	general purpose
3	3xxx	3xxx	main memory	3	general purpose
4	4xxx	0xxx	I/O	4	I/O, CAMMUs, reserved
5	5xxx	1xxx	I/O	4	I/O
6	6xxx	0xxx	Boot ROM	5	Boot ROM
7	7xxx	1xxx	Boot ROM	5	Boot ROM

Table 1 - Hardwired TLB Address Translation

	CPU SSW U, UU, K and KU Values			
ICAMMU	0x1x	0x0x	1x1x	1x0x
DCAMMU	001x	000x	1x1x	1x0x
PL			01x1	01x0
0	RW	–	–	–
1	RW	RW	–	–
2	RW	RW	RW	–
3	RW	RW	RW	RW
4	RW	RW	RW	R
5	RW	RW	R	R
6	RW	R	R	R
7	RWE	RWE	RWE	RWE
8	RE	–	–	–
9	R	RE	–	–
A	R	R	RE	–
B	R	R	RE	RE
C	–	RE	–	RE
D	–	–	RE	–
E	–	–	–	RE
F	–	–	–	–

R=read permitted, W=write permitted,
E=executed permitted, –=no access,
X=don't care

Table 2- Protection Codes

ST<2:0> or TG<2:0> Code	Meaning
0	private, write through, main memory space
1	shared, write through, main memory space
2	private, copy-back, main memory space
3	noncacheable, main memory space
4	noncacheable, I/O space
5	noncacheable, Boot ROM space
6	reserved
7	reserved

System Tag and Bus Tag Codes

Table 3

Figure 5: Hardwired Mapping of First Eight Pages in Supervisor Address Space

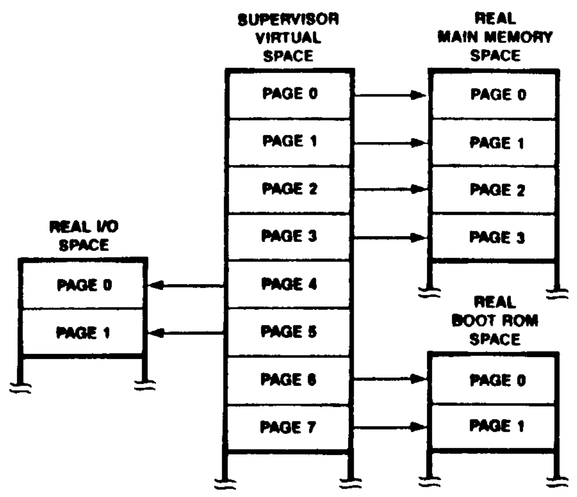
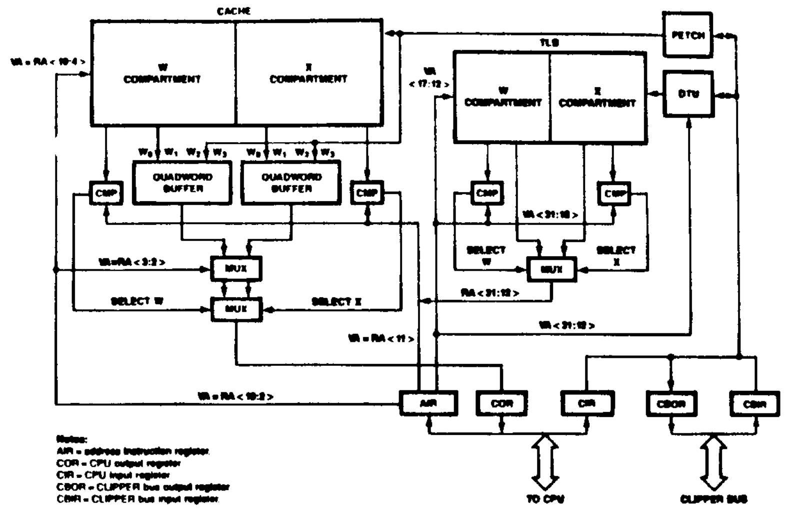


Figure 7: CAMMU Block Diagram



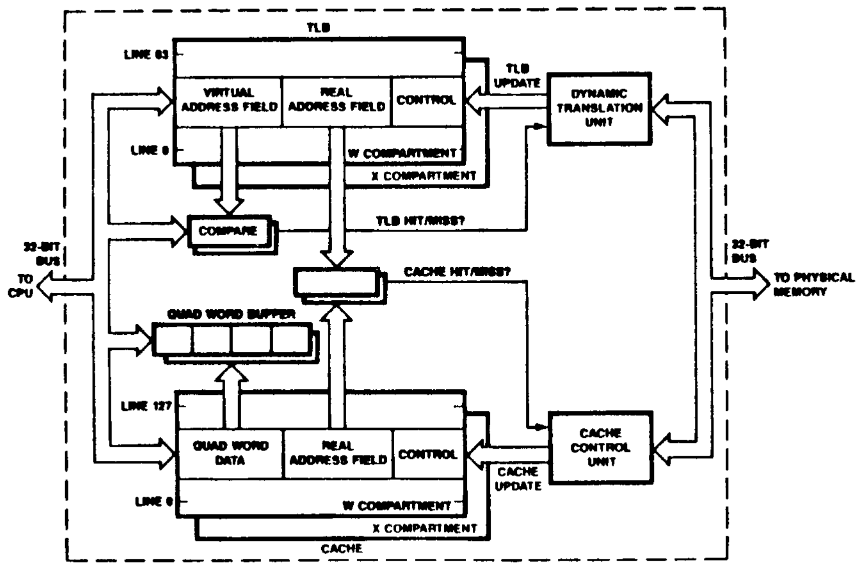


Figure 6: Simplified CAMMU Block Diagram

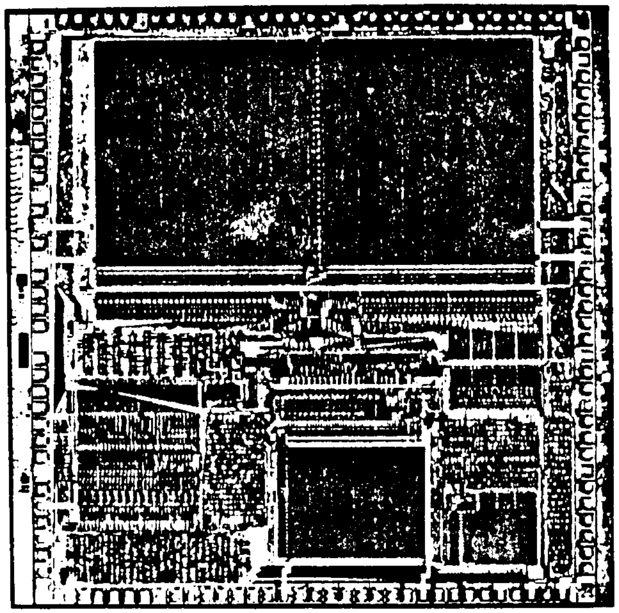


Figure 8: Photograph of CAMMU Chip

from location 6000 (hex) in the supervisor address space. It is likewise necessary to be able to do some input/output on initialization, and the permanent mapping of I/O space (IOS) facilitates this; IOS is also used to control the CAMMU chip (section 4.5). The low order regions of supervisor memory in most operating systems are also in constant use, and providing a hardwired mapping for those entries improves the performance of the TLB more than proportionately. Indirect addresses for traps, supervisor calls and interrupt vectors are in the first page.

3.3. System / User Address Space Communication

The fact that the user and supervisor address spaces are distinct means that special steps must be taken to allow the transmission of information between the user and supervisor. This can be done in three ways. First, the *Move Supervisor to User* and *Move User to Supervisor* instructions (available only in privileged mode) allow the transfer of information between the user and supervisor register sets. Second, the supervisor can address the user address space by setting the **UU (User Data Mode)** bit in the SSW, which means that memory operand addresses are interpreted via the user PDO; register references are to the supervisor register set. The third mechanism is to map a page of the user address space into the supervisor address space as well, although not necessarily at the same virtual address location.

3.4. Protection

A reference to a page can be a read, write or an instruction fetch, and thus an exhaustive protection scheme would allow each of these to be specified independently for each domain of execution. Such an exhaustive set of possibilities would be unlikely to be useful, (like independently steerable front wheels on a car) and for CLIPPER, a useful subset of these is available. Table 2 specifies the type of access allowed as a function of the PL (protection level) bits (from the page table entry) and the domain of operation, as defined by the U, UU, K and KU bits of the SSW (figure 2).

As we noted before, the **U bit** specifies the state - user/supervisor. When in supervisor state, the **UU** bit can be used to specify that memory operand addresses are to be taken from the user address space. **K** is used to set a protection level within either supervisor or user state, and **KU** is used in place of the K bit when the UU bit is set. This set of access possibilities is sufficient to cover all interesting cases, and permits the implementation of an operating system with several protection domains.

We note that because the protection level field is specified in the page table entry, access may be limited or permitted on a page basis, thus permitting a reasonably fine grain of access control. In particular, in CLIX, the version of UNIX^{TM} System V [Ritc74] that has been ported to CLIPPER, the protection access codes are used to implement a copy-on-write facility [Neff86], by which a process receives its own copy of a shared data page only when it attempts to write to it.

4. The Cache and Memory Management Unit (CAMMU) Chip

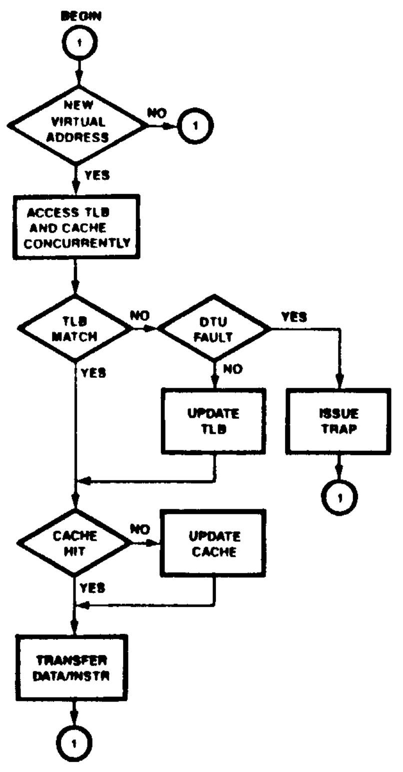
4.1. Overview

The responsibility for realizing the memory management architecture described above largely lies with the Cache and Memory Management Unit (CAMMU) chip. The CAMMU consists of three parts, the **Cache**, the **TLB (Translation Lookaside Buffer)** and the **Translator (DTU - Dynamic Translation Unit)**. Figure 1 shows the CAMMUs, as they are located in the CLIPPER module. A basic block diagram of the CAMMU appears in figure 6 and a more detailed diagram of some aspects of the CAMMU is in figure 7; a picture of the chip is in figure 8. A tutorial and thorough discussion of the functions of the parts of CAMMUs and their performance optimization appears in [Smit82,84] and the reader may wish to refer to those articles for a more general discussion than is provided here.

The basic operation of the CAMMU is diagramed in figure 9 and is summarized here; a more detailed discussion of some aspects appears in the following subsections. Translation of virtual to real addresses is normally performed by the **Translation Lookaside Buffer (TLB)**, which retains copies of recently used virtual to real translation pairs, along with the other fields of each page table entry. When the TLB fails to have a needed translation entry, the **DTU** accesses the page table directory and page table for the appropriate address space and loads the translation entry into the TLB; the translation then proceeds as before. In parallel with the translation, the protection fields and system tag field becomes available, and the validity of the operation is checked; if the operation is invalid (e.g. protection fault, page fault), a trap code is returned to the CPU.

Because access to main memory is normally quite slow compared to the cycle time of the processor, the CAMMU also contains a **cache** which retains the contents of recently referenced portions of the main memory physical address space. After the real address is available, it is presented to the cache and if the corresponding memory contents are found, they are then returned to the CPU via the CAMMU / Processor bus (section 5). If the information is not in the cache, the

Figure 9: Simplified Flow Chart of CAMMU Operation



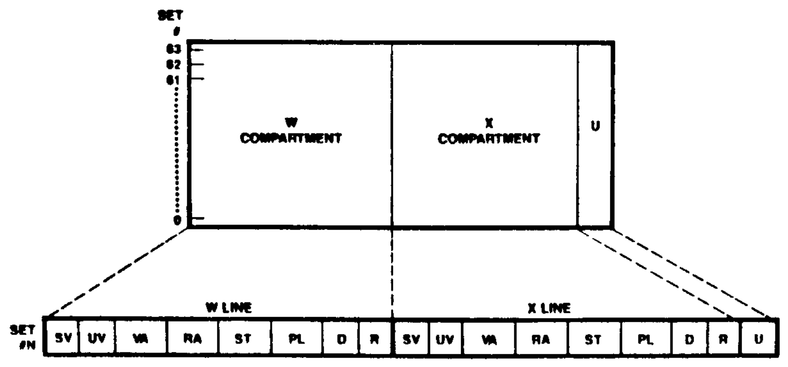
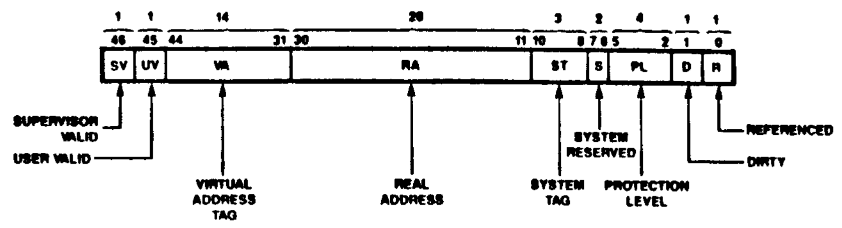


Figure 10: TLB Organization

Figure 11: TLB Entry Format



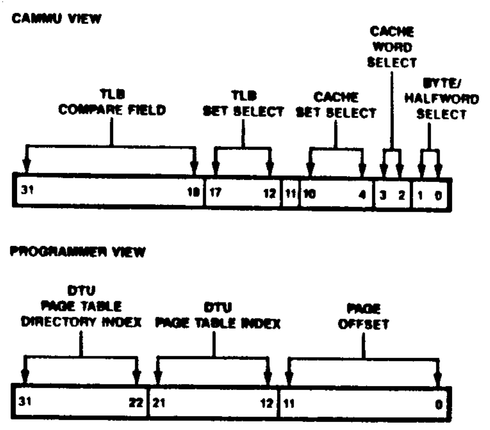


Figure 12: Interpretation of 32 Bit Virtual Address by CAMMU and Programmer

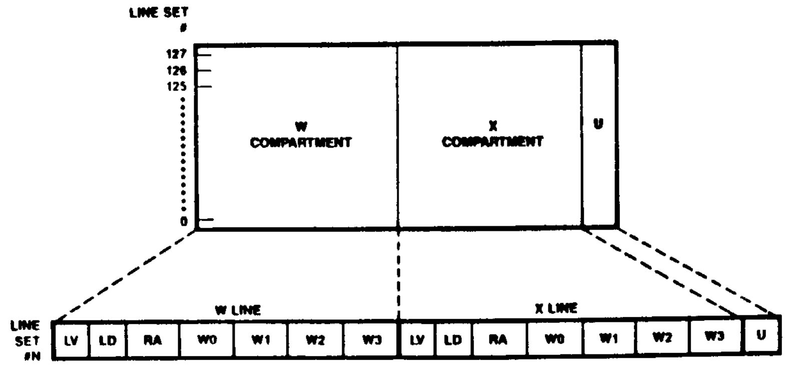


Figure 13: Cache Organization

cache loads the necessary material from main memory (copying modified information back to main memory if appropriate), and then proceeds with the reference. Aspects of cache management (e.g. copy back, write-through, etc.) are determined by the system tag, as explained below (section 4.4.3).

4.2. The Translation Lookaside Buffer

4.2.1. Mapped Mode

Each Translation Lookaside Buffer (TLB), sometimes referred to as the Translation Buffer (as with the DEC VAX), or the Directory Lookaside Table, is functionally an associative memory which provides rapid virtual to real address translation; it is implemented in CLIPPER as a set associative memory with two elements per set (2 way set associative) and 64 sets; i.e. 128 entries. A diagram of the contents of the TLB appears in figure 10 and the contents of a TLB entry is in figure 11.

Bits 12-17 of the virtual address are used to select a set of the TLB (figure 12), and then bits 18-31 of the virtual address are compared with the virtual address (VA) field of each TLB entry in that set. If a match is found, and if the appropriate valid bit is also on (**supervisor valid** (SV) or **user valid** (UV), depending on the SSW U bit), then the remaining TLB entry fields are selected and made available; the real address field is passed to the cache (see section 4.4 below).

The protection level field (PL) is used with bits U, UU, K and KU from the SSW (from the CPU) as per table 2 to validate the memory reference, in parallel with the translation. If the access is not valid, then the reference is halted, and a trap code is returned to the CPU.

One of the outputs of the translation process is the **system tag** (ST) field. That field specifies how the cache is managed, and whether the reference is to the cache at all; the ST codes are shown in table 3. If the code is 3, then the reference is non-cacheable, and proceeds immediately to main memory via the system (CLIPPER) bus. Codes of 4 (noncacheable, I/O space) and 5 (noncacheable, boot ROM space) are produced only in response to a reference to virtual pages 4-5 and 6-7 of the supervisor address space respectively. Both codes 6 and 7 are reserved for future use. Codes 0, 1, and 2 specify the type of cache operation, as described below in section 4.4.3.

The **dirty** (D) bit of the TLB entry is set when a write to that page takes place. When the D bit first changes from 0 to 1, the D bit is also written back to main memory immediately to the page table entry; further writes to the page while the dirty bit is on do not require additional writes to the D bit in the page table.

The **referenced (R)** bit of the TLB entry is set whenever that TLB entry is used for a translation and is also written back to the page table entry only when it first changes from 0 to 1.

Care must be taken that when a referenced or dirty bit is reset in the page table in main memory, the corresponding TLB referenced or dirty bits are similarly reset (by software).

Each pair of TLB entries have a **U (used) bit** associated with them. The Used bit specifies which of the two TLB entries in that set was least recently used. When an entry in the TLB set must be replaced, the LRU entry is selected.

As noted earlier, pages 0-7 of the supervisor address space are translated by special random logic, and are not mapped through the TLB; thus those translations never appear in the TLB and never displace other TLB entries.

It is important to note that the TLB does not contain a field for an address space ID; each entry is tagged only with a bit indicating if it is valid in the supervisor or user address space (or both). This implies that when a new user process is started, (i.e. one different than that most recently running), all TLB entries with the User Valid bit set must be purged. This is accomplished by the appropriate CAMMU control command, issued by the operating system, as described further in section 4.5. The savings in TLB miss ratio (see sections 7 and 8) to be gained by adding an address space ID were not considered sufficient given the additional silicon area needed in the CAMMU and on the processor chip. Future implementations of CLIPPER can be extended to include such a feature if it appears to be a good tradeoff for the then-existing technology.

4.2.2. Unmapped Mode

When operating in Unmapped Mode, the addresses generated by the CPU are not translated, i.e. neither the TLB nor the translator are used, but instead the address is used directly as a real memory address. There is, however, the need for some of the bits available in the page table, and normally cached in the TLB entry. The necessary system tag bits come from the CAMMU control register (section 4.5). There is no protection when operating in unmapped mode, and no protection bits are generated. Likewise, the referenced and dirty bits are not used when in unmapped mode.

The use of unmapped mode only disables translation; the cache is still available. However, the default settings for the CAMMU control register, i.e. those it is set to on reset (section 4.7), mark all references in unmapped mode as referring to uncacheable pages.

The expectation is that mapped mode will be used exclusively or almost exclusively; unmapped mode is provided mostly for flexibility and completeness. Its primary use is to build the page tables on power-up; the boot ROM code also runs in unmapped mode.

4.3. Dynamic Translation Unit

The Dynamic Translation Unit (DTU - the Translator) is responsible for loading new entries into the TLB, when an attempt to perform a translation is unsuccessful due to a missing entry. It operates (figure 3) as follows: (a) The TLB signals that a translation has failed due to a missing entry. (b) The DTU concatenates the 10 high order bits of the virtual address to the low end of either the user or supervisor PDO (page directory origin) register, and from the resulting address fetches a page directory entry (PDE) from main memory. (c) The DTU concatenates bits 12-21 from the virtual address to the high order bits of the PDE, and fetches a page table entry (PTE) from main memory at the resulting address. (d) The {virtual address, page table entry} pair is loaded into the TLB, displacing the least recently used entry in that set. Because the DTU fetches PDEs and PTEs from main memory only, all Page Tables and Page Directories must be located in non-cacheable pages (see section 4.4.3).

If either the page table directory, the page table or the page is found to be missing from main memory, the translator fails to complete the translation, and instead signals to the processor a page fault trap; the virtual address which caused the translation fault is held in the CAMMU fault register (see section 4.5.1.2), and may then be read by the CPU.

The processor, on the occurrence of a page fault, finds the virtual address causing the fault, and determines the physical disk address of the page, if any. The operating system finds a vacant page frame (removing another page, if necessary), and issues a disk I/O command to transfer the referenced page from disk to main memory. On the completion of that transfer, the disk controller will signal to the processor with an I/O interrupt, and the operating system will update the page table and restart the faulting process. (It is possible that the page table or even the page directory is missing, in which case the page table or page directory must also be fetched. The CLIX operating system does not permit a page to be in memory without its corresponding page table. The supervisor address space is paged as well, and the supervisor can handle page faults to its own address space (with the exception of the paging code itself, which must never be paged out)).

Once the page fault has been handled, and the target page is in memory, the process is restarted and the access reattempted. This time, the DTU will succeed in loading the translation entry into the TLB. The translation entry will consist of the virtual and real addresses, and the other fields including the system tag, protection level and supervisor and user valid bits.

4.4. The Cache

Cache memories are small and fast associative memories placed between the processor and main memory to reduce average memory access time by retaining the contents of recently referenced portions of the processor address space [Smit82,84]. As we discuss below in section 7, the performance of CLIPPER is quite sensitive to the average memory access time, and a cache is necessary for it to operate at a satisfactory rate. A cache is provided on each CAMMU chip. In this section, we discuss the design and operation of the cache, concentrating initially on the "normal" case of a read to a cacheable line. Writes and non-cacheable references are considered subsequently, in the section on caching modes (section 4.4.3).

4.4.1. Cache Organization

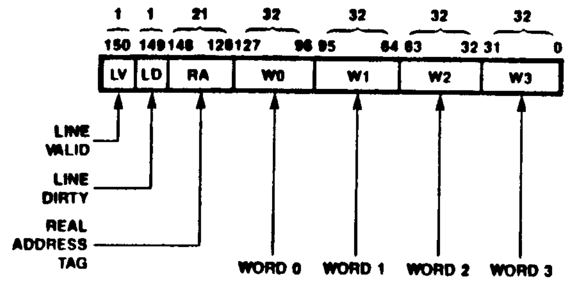
The cache on each CAMMU is 4Kbytes, organized as a 2-way set associative memory of 128 sets of two 16-byte lines (blocks). A diagram of the cache organization appears in figure 13 and the cache data flow paths are shown in figures 6 and 7. Each cache line (see figure 14) consists of 4 words of information, a line valid bit (LV), a line dirty bit (LD) and a real address tag field (RA); the use of each is described below. There is also a U (used) bit to specify the least recently used line of the pair in the set.

4.4.2. Cache Operation

Bits 0-11 of the virtual address specify the byte within page, and thus are not translated by the virtual memory mapping mechanism; they are the same for both real and virtual addresses. Those bits are thus available prior to the translation by the TLB, and bits 4-10 are used immediately to select a set of the cache (figure 12). As seen in figure 7, both data entries in that set are immediately gated into quadword buffers, and the real address tags are each gated into comparators. Bits 2-3 of the virtual address are further used to select from each of the quadword buffers the individual word that is addressed.

Simultaneously with the selection and read out of the two lines in the cache set, the two TLB entries specified by bits 12-17 of the virtual address are gated

Figure 14: Cache Line Format



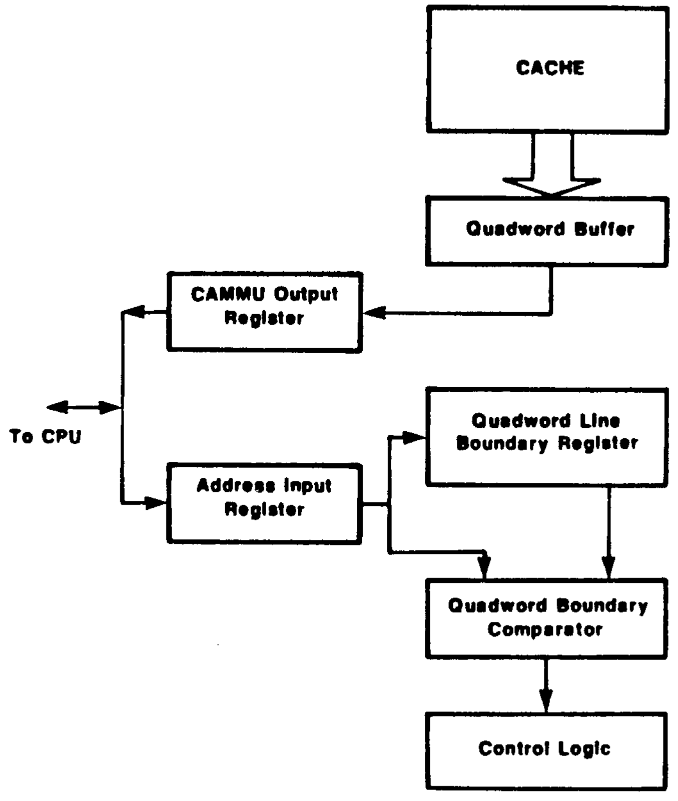
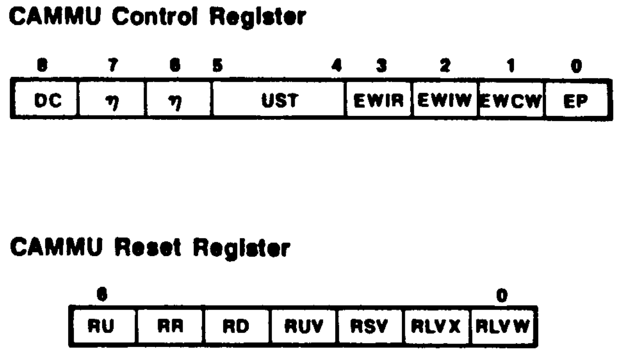


Figure 15: Quadword Buffer in CAMMU

Figure 16: CAMMU Control Register and Reset Register



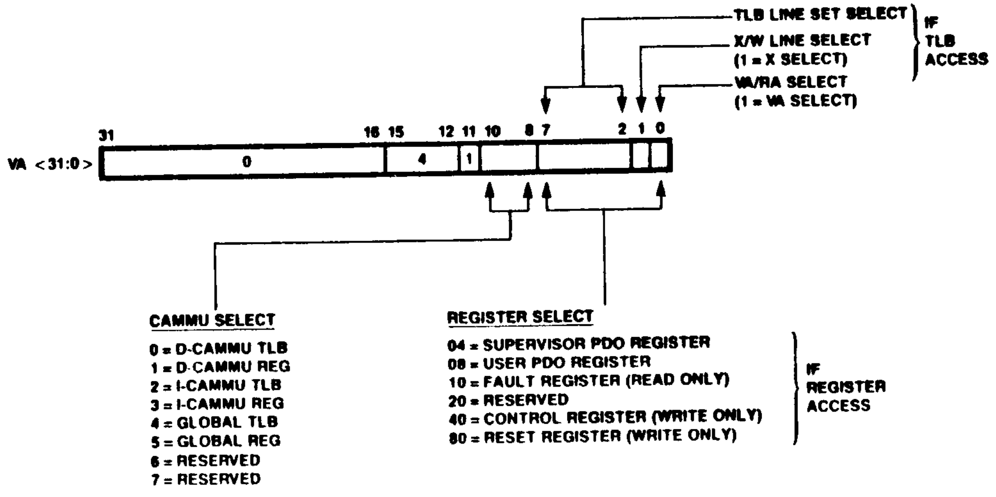


Figure 17: Interpretation of Address for CAMMU Control Command

into comparators, and the virtual address fields of the TLB entries are compared with bits 18-31 of the virtual address. A match by one of the comparators causes the real address associated with that entry to be gated into both of the comparators for the real address tags of the cache lines. If one of those comparators finds a match, then the associated word buffer is selected, as holding the target word. (If the LV (line valid) bit for a line is unset, then the address tag for that line will never match the one from the TLB.) In the case of a read, the target word is then transferred over the bus to the processor chip. The U bit is set to indicate the least recently used element of the cache set.

If no match is found between one of the real address tags in the cache line and the real address obtained from the TLB, then a **cache miss** occurs. (The fraction of cache references causing a miss is called the **miss ratio**.) In that case, a line of the cache must be selected to receive the target line when it is fetched. If one (or both) of the lines has the LV bit unset, then that entry is selected for the line to be loaded; otherwise, the least recently referenced element of the set is selected, as determined by the U bit.

It is possible that the location into which a line is about to be fetched contains data which has been modified, and is not valid in main memory. This case is detected by observing that the **LD (line dirty)** bit in that line is set. If so, the dirty line is written to main memory; the main memory address of the line to be pushed is available from its real address tag.

After the line in the cache has been made available (by pushing a dirty line to main memory, if necessary), the real address of the target line (from the TLB) is sent to main memory, the 16 byte line is returned over the system (CLIPPER) bus, 4 bytes at a time, and it is loaded into the cache.

4.4.3. Caching Policy

There are a number possible cache management policies, and CLIPPER provides that a cache management policy may be selected on a page by page basis. Four policies are provided for main memory references, and these are specified by the System Tag (ST) bits of the page table entry (and are buffered in the TLB). The three bits of the System Tag can specify up to 8 possible modes, and two of the four remaining modes apply only to references to the lower 8 page pages of supervisor virtual memory; see the discussion above in section 3.2. The last two remaining modes are not used, and are reserved.

The four policies provided (see table 3 for the system tag codes) for normal main memory references are: (0) private, write-through, (1) shared, write-through, (2) private, copy-back, and (3) noncacheable. The terms shared and private refer to the bus watch modes, and are discussed both here and further below in section 4.6. The difference between write-through and copy-back refers primarily to the behavior of the cache when a write occurs.

On a write, and using the **write-through policy**, there are two possibilities: if the line is in the cache, it is updated in the cache, and main memory is also updated. If the line is not in the cache (a write miss), the line is not fetched; only main memory is updated. For write-through, therefore, *no fetch on write* is used.

When using **copy back**, the cache is always updated. If the line is not in the cache (a write miss), the line is fetched and then updated; for copy back, the policy is *fetch on write*. Main memory is only updated when the dirty (modified) line in the cache is replaced, not when the write occurs.

We observe that when write-through is used, main memory is always current; when copy-back is used, the only valid copy of some data may be in a cache memory. Write through, as is discussed below in the section on bus watch modes, facilitates the sharing of data between processors, but increases bus traffic. Copy back reduces bus traffic, but requires more careful data management. System tags should be assigned with these considerations in mind.

The write-through policy may further be designated as shared write-through (see table 3) or private write-through. Any page that is to be shared and modified by more than one CAMMU, including the I and D CAMMUs of the same processor, must be marked as **shared, write-through**. Consistency between the CAMMUs will only be maintained for a page if the caching policy is shared, write-through. I/O is considered as a special case (see section 4.6) and regions to be read or written by I/O devices need not be marked as shared.

The remaining policy is **noncacheable**. As noted above (section 4.3), page tables and page directories must be noncacheable. Areas used alternately by different processors, areas used by interprocessor communications variables and regions, and areas used only as output buffers may be usefully designated noncacheable. Read access times to noncacheable regions, however, are quite high, and the noncacheable mode should be used sparingly for that reason.

4.4.4. Prefetching

Typically, instructions are executed sequentially, and therefore there is a high probability that after line i is referenced, line $i+1$ will also be referenced. It was observed in [Smit78a, Smit85a] that sequential prefetching significantly lowered the miss ratio for instruction caches of all sizes, and reliably lowered the miss ratio also for data caches of 8Kbytes or over. **Sequential prefetching** means that whenever line *i* is referenced in the cache, the cache prefetches line *i+1* if it is not already cache resident.

Two features of the cache are provided to take advantage of the utility of prefetching and thereby optimize system performance. First, the quadword most recently referenced in the cache is retained in the **quadword buffer** (figure 15; see sections 4.4.2 and 5), until a different quadword is referenced. A reference to the same quadword as was most recently referenced is detected and the target bytes are immediately returned to the CPU on a read, without the need (and delay) for a translation and cache access. This feature is available on both the instruction CAMMU and data CAMMU. It works quite well in the instruction CAMMU (see section 7) because of instruction sequentiality. Good results are also obtained in the data CAMMU because of sequential data accesses due to string operations, double precision floating point loads and stores, context saves and restores on procedure calls, such as are common in UNIX, and sequential data references frequently encountered when processing arrays.

The instruction CAMMU also optionally provides prefetch. When prefetch is enabled (via a bit in the CAMMU control register; see section 4.5), a reference to line *i* causes the CAMMU to check to see if line *i+1* is in the cache. (Actually, only when a new line is loaded into the quadword buffer does the check take place.) If the next line is absent, the CAMMU initiates a fetch from main memory for that line. If access continues sequentially, then the prefetch transfer takes place in parallel while the CPU satisfies its fetch requests from the quadword buffer. (The CPU is locked out of the rest of the cache, however, during a prefetch transfer.) Prefetches are not initiated when the subsequent line is beyond a page boundary, so no extra page faults are generated due to prefetching.

It is known [Smit78a, Smit85a] that sequential prefetching significantly lowers the instruction miss ratio, but increases the bus traffic somewhat; prefetching may thus be turned on or off depending on the system or application, so as to maximize performance, given the advantages and disadvantages to prefetching. From results in [Smit85a], it can be seen while prefetching cuts the miss ratio for a 4Kbyte data cache on the average, it does not always do so, and it also increases the bus traffic significantly. Prefetching is therefore not always advantageous for a cache like that on the data CAMMU, and is therefore not provided there; a non-prefetch policy is called a *demand* policy.

4.5. CAMMU Control and CAMMU Registers

4.5.1. CAMMU Internal Registers

There are five software accessible internal registers on the CAMMU chip; these are used for initialization and control. It is possible to read and write these registers using I/O commands. It is also possible to read and write TLB entries using I/O commands, and that is discussed later in this section.

4.5.1.1. PDO Registers

The supervisor and user PDO (page directory origin) registers are 20 bit read/write registers that hold respectively the high order bits of the real memory address of the page directory for the supervisor and user. The contents of these registers are used by the Translator (see section 4.3) to translate virtual to real addresses.

4.5.1.2. Fault Register

The fault register is a 32 bit read only register which holds the virtual address associated with a page or protection fault. It is intended for use by trap handling routines for fault recovery.

4.5.1.3. Control Register

The control register is a 9 bit register used to control aspects of the CAMMU operation. Seven of the bits are programmer visible and those bits are shown in figure 16; the remaining two bits (6&7) are used for hardware testing and debugging.

When the system is running in the unmapped mode (M bit in SSW is clear), all CPU addresses are treated by the CAMMU as real addresses not requiring translation, and therefore they cannot obtain system tag values from the page table or TLB. The two bit UST field is used to specify the default caching policy (see table 4), for all unmapped references. UST is set to 3 when the CLIPPER module is reset.

Bits EWIR, EWIW and EWCW mean respectively Enable Watch I/O Reads, Enable Watch I/O Writes and Enable Watch CPU Writes. These bus watch policies are explained in section 4.6. All three bits are set by CLIPPER reset (section 4.7).

Bit EP, when set, enables prefetching in the ICAMMU; it is set by CLIPPER reset.

UST	Meaning
0	private, write through, main memory space
1	shared, write through, main memory space
2	private, copy-back, main memory space
3	noncacheable, main memory space

Codes for UST Field of CAMMU Control Register

Table 4

Bit	Bit Name	Operation
6	RU	Reset All U Flags in Cache
5	RR	Reset All R Flags in TLB
4	RD	Reset All D Flags in TLB
3	RUV	Reset All UV Flags in TLB
2	RSV	Reset All SV Flags in TLB
1	RLVX	Reset All X Line LV Flags in Cache
0	RLVW	Reset All W Line LV Flags in Cache

Table 5 - Reset Register Command Bits

Bits 10-8	CAMMU	Operation
0 0 0	Data	R/W TLB
0 0 1	Data	R/W Registers, Reset TLB/Cache
0 1 0	Inst.	R/W TLB
0 1 1	Inst.	R/W Registers, Reset TLB/Cache
1 0 0	Global	Write TLB
1 0 1	Global	Write Registers, Reset TLB/Cache

Bits <7:0>	Register Addressed
0000 0100	Supervisor PDO
0000 1000	User PDO
0001 0000	Fault (read only)
0010 0000	reserved
0100 0000	Control (write only)
1000 0000	Reset (write only)

Interpretation of Addressing Bits for CAMMU Commands
Table 6

Signal	State	Operation or Meaning
CT5	0	CPU Master
	1	I/O Master
CT4	0	Write Operation
	1	Read Operation
CT<3:2>	00	Single Word Transfer
	01	Quadword Transfer
	10	reserved
	11	Global CAMMU Write
CT<1:0>	00	Whole Word Transfer
	01	reserved
	10	Byte Transfer
	11	Halfword Transfer

Cycle Type Codes on System Bus

Table 7

Bit DC, when set, causes the dirty bit on a (copy-back) cache line to be reset when the data contained within that line is supplied for an I/O Read. This bit is to be set only when using a memory controller that is capable of capturing from the bus the dirty line (as it is transferred to the IO Controller) and simultaneously updating itself.

4.5.1.4. Reset Register

The reset register (see figure 16), which when written to, causes certain bit flags to be reset in the cache or the TLB. The Reset Register contains 7 bits; a write operation to the reset register with bit #i set will cause the event described in table 5. We describe the more interesting cases in the remainder of this section.

Setting the bits RLVX and RLVW have the effect of completely clearing the cache, since all lines are now marked as invalid. There is no reason to set only one of these bits (except for hardware debugging purposes), since whether a line is in one element of a set or the other is generally unpredictable. This operation also has the effect of discarding all lines in the cache even when they are dirty and are the only valid copy, and thus this operation should be used carefully.

Writing to the RUV bit (reset all UV flags in TLB) has the effect of purging all user address space translations from the TLB. Since the TLB does not contain an address space ID for each translation, it is necessary to set RUV when starting a user address space different than was running previously. There exists another mechanism (see section 4.5.2) to selectively invalidate TLB entries when a page table entry is changed, so it isn't necessary to clear the entire TLB when a page fault occurs.

4.5.2. Reading and Writing CAMMU Registers and TLB Entries

Commands are given to the CAMMU and the CAMMU registers via reads and writes to the I/O space. The D-CAMMU is controlled directly, since all writes by the CPU go directly to/through the D-CAMMU. The I-CAMMU must be controlled indirectly, since there is no provision for issuing writes over the instruction bus; this is done by having the D-CAMMU pass the commands along the interface between the two CAMMUs (see figure 1). The D-CAMMU is controlled by reads and writes to virtual addresses 4800-49FF (Hex), and the I-CAMMU by reads and writes to 4A00-4BFF.

It is also possible to read and write entries in the TLB using IO to the CAMMU, with the target of the operation specified by the specific address read or written. We note that since commands to the CAMMU consist of reads and writes through the IO portion of the supervisor address space, such commands can only be issued when in supervisor state.

In figure 17 and table 6, we show the interpretation of the addressing bits when reading or writing to that portion of the IO space which references or controls the CAMMU. As can be seen, bits 11-31 are fixed, and specify the upper half of page 5 of the supervisor virtual address space. Bits 8-10 select either the I or D CAMMU, or all CAMMUs using the same system bus (global space), and also specify whether a CAMMU register or TLB is being referenced. If a register is selected, bits 0-7 specify the register, as shown in table 6. If a TLB entry is selected, bits 2-7 specify the set, bit 1 indicates which of the two entries in the set, and bit 0 whether the real address (RA) portion of the entry or the virtual address (VA) portion of the entry is to be read or written (figure 18).

The IO address is used to select the register or TLB entry to be read or written. The data word or words read or written follow a standard format. That format appears in figure 18, where we show the formats for the TLB RA field, TLB VA field, and PTE contents; the control and reset register contents were previously defined in figure 16.

It is worth noting the principal use of the TLB read/write commands, and that is to be able to implement changes in the address map without purging the entire TLB or all TLBs in the system. Specifically, consider the effect of changing a virtual to real address translation, as one would do under the following circumstances: (a) page fault, where a page map entry is added, (b) page out, where a page map entry is deleted, and (c) address space sharing where a portion of one address space (e.g. a user address space) is also mapped into the address space of another process or the supervisor.

When a new translation is added, i.e. it replaces a previously invalid one (page table entry invalid), then no special action need be taken; the TLB will be loaded by the translator automatically.

The problem comes when a previously valid translation is either deleted, (the page is paged out or otherwise removed from the address space) or when it is changed, as when a region of one address space is mapped into another. Given a virtual address, the set in the TLB is determined by bits 12-17 of the virtual address, and therefore if a translation for that page is in the TLB, it is in one of the two entries in that set. By invalidating both of those entries, it is guaranteed that an invalid entry has been deleted. Similarly, if the translation may exist elsewhere in the system, a global TLB write has the same effect; a global TLB write permits entries in a given set to be invalidated in every TLB in the system (i.e. sharing the same bus). Since the TLB set is selected via the virtual address

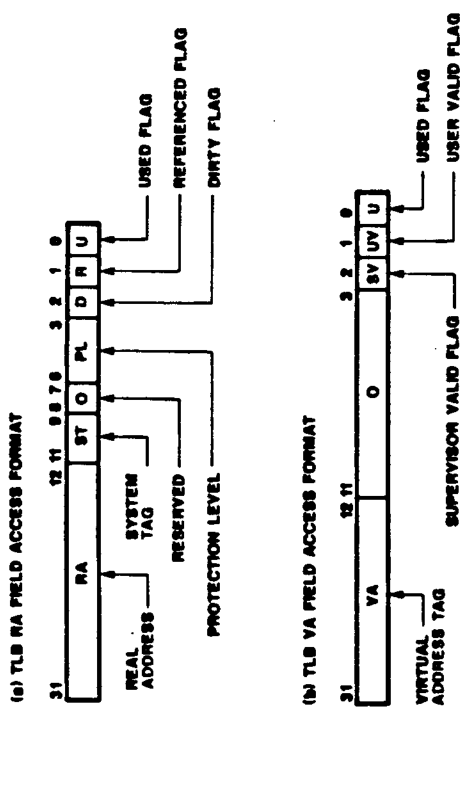


Figure 18: Interpretation of Data Field for CAMMU Control Command

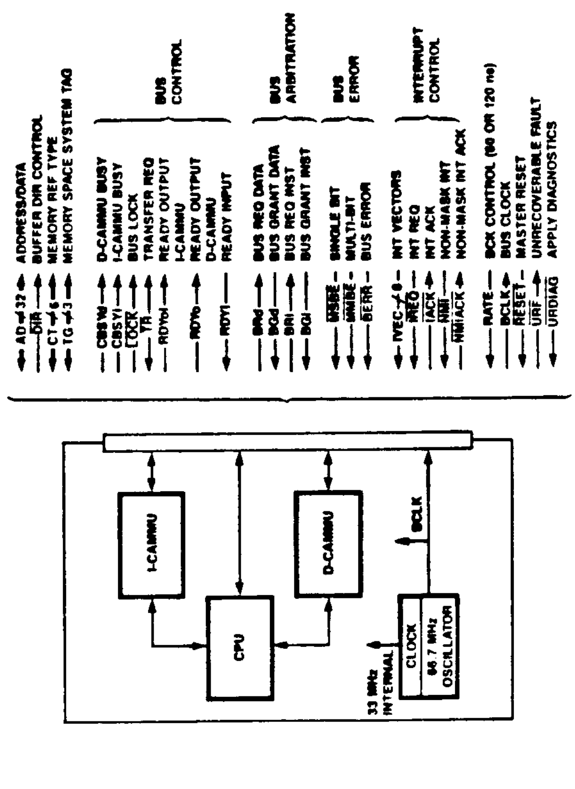


Figure 19: System Bus Interface

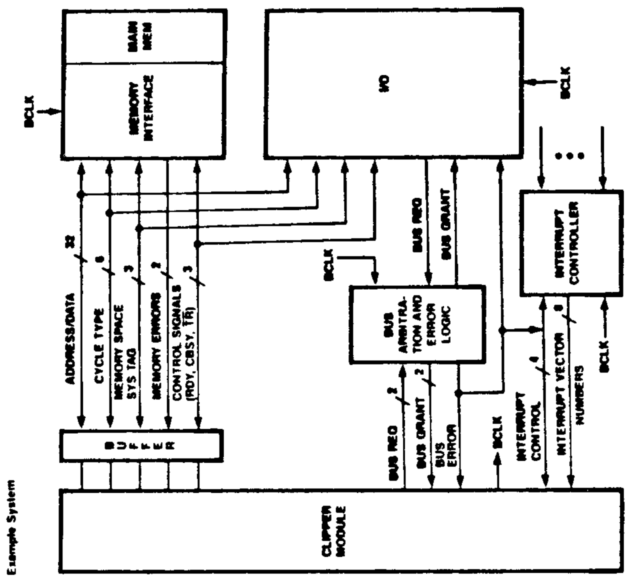


Figure 20: Example CLIPPER Module System

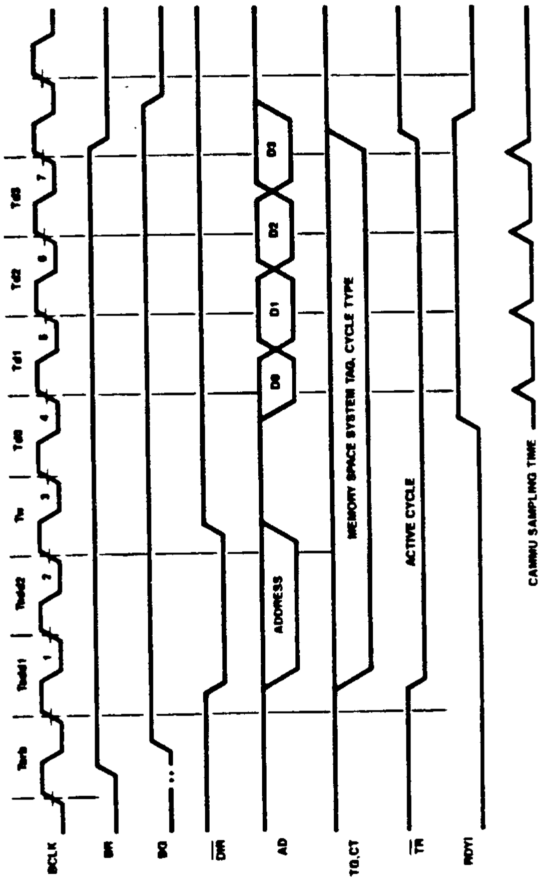


Figure 21: Bus Timing for Quadword Read

and since if a translation is changed, it is the virtual address that has become invalid, such an invalidation has the desired effect.

The principal use for the ability to read TLB entries is for debugging of either the software or the hardware; it is not necessary for normal operation of the operating system.

4.6. Cache Consistency and Bus Watch Modes

CLIPPER is designed to operate with a global system bus (called the CLIPPER bus) to which are attached one or more CLIPPER modules, one or more I/O interface processors, and main memory. For high performance, the CLIPPER CAMMU provides a copy-back caching mode, whereby data may be written to the cache and only later recopied back to main memory. The difficulty is that some region of memory may be referenced by a processor or I/O device while the only valid copy of that data resides in a cache memory; this is called the *cache consistency problem*. The performance advantages of a copy back design are so compelling in terms of reduced bus traffic and decreased processor idle due to delays on writes (see section 7 for a further discussion), that a means must be found to ensure cache consistency.

The advantages of a shared bus design in which the processors have cache memories and use copy back have long been recognized, but the first solution to the resulting cache consistency problem was proposed only in 1983 by Goodman [Good83]. Goodman's scheme required that every cache monitor the bus, and when it observed a read to an item for which it held the only valid copy, it would preempt the read from memory, and would supply the cache line itself. (There is considerably more to his algorithm than we give here; we are noting only the essential concepts.) Other bus watch schemes were proposed by Papamarcos and Patel [Papa84] and by Katz et al. [Katz85]; all are similar in holding and passing around the only valid copy of some portions of main memory. (A more detailed discussion of this material is provided in [Smit85c] and [Arch85], and a new consistency scheme is also proposed in [Swea86].)

CLIPPER features a new bus watching cache consistency protocol, which is different than any of those previously proposed. Here we describe that protocol, and then comment on why it was chosen rather than one of the other schemes mentioned.

When **Bus Watch** is enabled in a CAMMU, it monitors memory access by other bus masters. The CAMMU can be enabled separately to watch three different events, which are: (a) Watch CPU writes (CPU writes to shared cacheable pages), (b) Watch I/O Writes (I/O Writes to cacheable pages) and (c) Watch I/O

Reads (I/O reads from private copy-back pages). The type of operation on the bus is defined by the tag bits (table 3) (which are the same as the system tag bits) and the cycle type bits (read/write, byte, single word or quadword transfer, etc., table 7)); bus operation is described further in section 6. As noted earlier (section 4.5), these three bus watch modes are enabled/disabled by bits 0-2 of the CAMMU control register. Each of these is now described.

Watch CPU writes is enabled in a CAMMU to ensure that data in the local CAMMU is either updated or invalidated when that data is written to by another CAMMU. We refer to the CAMMU initiating the write as the *master* and the CAMMU monitoring the bus as the *slave*. When Watch CPU writes is enabled and the slave CAMMU observes a write to a shared page, the slave CAMMU searches its own cache (keying on the real address taken from the bus) to see if the line is in its cache. If the write is a single word or less, then the local copy is updated; if the write is for an entire quadword, the local copy is invalidated - i.e. LV is set to zero. That the write is to a "shared" page is determined by the TG bits on the bus, which with a code of 001 specify shared, write-through.

We note that bus watch on CPU writes is only invoked when the reference is marked as "shared;" there is an important efficiency reason for this choice. The state of silicon fabrication technology in 1985/6 does not make it feasible to provide the cache on the CAMMU with a duplicate directory, whereby a bus watch cache search can occur simultaneously with a processor access, without removing some other functionality from the CAMMU; this was not considered to be a desirable tradeoff. Each bus watch operation makes the CAMMU busy for a number of CPU clock cycles (6 cycles for any bus watch miss or a hit on an IO Write or CPU Write; 10 cycles for an IO Read of one word, and 16 cycles for an IO Read of a quadword). To the extent that searches are only performed for pages which might be shared, the performance impact is minimized.

The *test and set* operation is an atomic operation performed on a lock resident in main memory; test and set variables are not normally cached, and do not interact with the consistency protocol.

Watch I/O Writes functions identically to Watch CPU Writes, but applies to I/O writes only; that this is an I/O write may be determined by the CT (cycle type) lines of the bus. When the CAMMU with this function enabled observes an I/O write, it searches its local cache and updates for a one word (or shorter) write, and invalidates for a quadword write. Since I/O activity is likely to be only a very small fraction of the bus traffic, enabling a watch for I/O activity is not likely to have a significant performance impact.

The third bus watch mode is **Watch I/O Reads**. When this mode is enabled, the D-CAMMU monitors the system bus for I/O reads out of private, copy-back pages. Each such read causes the CAMMU to search the cache to see if that line is resident. Should there be a hit, and the local copy is dirty, the CAMMU preempts the bus before main memory can respond, and itself supplies the contents of the memory locations being read. This mode functions identically for both quadword and smaller transfers. Again, because I/O traffic is only a small fraction of the bus activity, enabling watch I/O reads is likely to have small performance impact, and may be used under most circumstances. We note that this bus watch mode only functions, and is only necessary, for private, copy back pages, since otherwise main memory must be up to date.

Although the published bus watching cache consistency protocols all lead to less bus traffic than the protocol used in CLIPPER, *there are two reasons why the CLIPPER protocol was chosen*. First, as noted above, CLIPPER does not have a dual ported cache, and therefore there would be a performance impact if all bus activity were monitored. Second, the susceptibility of these other designs to failure is substantially higher than for CLIPPER. With CLIPPER, the failure of a processor and/or its associated cache can at worst result in the loss of the contents of private pages which have dirty lines in that cache; any region that is shared is always up to date in main memory. This means that a failure should only crash a process whose data is held in a copy-back cache, and most of the time, only one process will have such data resident in a given cache. Conversely, with the other protocols, the only valid copy of any shared data could be in any cache; not only could the failure of a single element bring down the system, it would almost have to: there would be no way to know if the only valid copy of data was in the failing unit. The CLIPPER protocol, therefore, both takes advantage of the state of the art of semiconductor fabrication technology, and also provides reliability advantages over those other consistency schemes.

It is worth noting, however, that minor modifications to the CLIPPER consistency scheme could convert it into one of the other bus watching protocols. For example, if Watch I/O Reads is applied to all CPU reads, and if a read by another CPU also causes an invalidate, then a type of of "ownership" protocol [Swae86] has been created, whereby a given line could reside in at most one cache. Since the consistency protocol is not visible to the user mode programmer, and is only visible to a small part of the operating system (the memory management code), the protocol could be changed for future implementations without impairing the portability of code developed for the current machine.

4.7. System Reset

The CLIPPER system is reset either when power is first applied or when the system reset line is asserted. Many of the effects of reset apply to the memory system.

Reset causes all entries in both the cache and TLB to be marked invalid - i.e. all LV (line valid) flags in the cache are cleared and all UV (user valid) and SV (supervisor valid) flags in the TLB are cleared. Somewhat incidently, the U (used) flags in the cache, the D (dirty) flags in the TLB, and the R (referenced) flags in the TLB are also cleared, although since the entries are invalid, the setting of these flags doesn't really matter.

The CAMMU Control Register is set to (hex) 3F on reset, which means that unmapped accesses are treated as uncacheable, bus watch is enabled for I/O Reads, I/O Writes and CPU Writes, and prefetch is enabled for the I-CAMMU.

In the CPU, the PSW (program status word) and SSW (system status word) are cleared. This means that the CPU is placed in supervisor mode with all traps and conditional interrupts disabled.

5. The Interface Between the Processor and CAMMUs

The interface between the Processor and each of the CAMMUs is not visible either to the programmer or the hardware designer working with the full CLIPPER module, but we believe that it is useful to describe that interface; we do so briefly in this section.

The interface between the CPU and CAMMU consists of 45 lines into the D-CAMMU and 46 lines into the I-CAMMU. (See table 8.) 32 lines are **bi-directional address and data lines**. 4 bits transmit the **protection bits and supervisor/user state bit** from the SSW: K, UU, KU, and U, and a fifth bit specifies the **M (mapped) bit** from the SSW. There is a four bit bidirectional set of lines which from the CPU to the CAMMU give the **function code** (which specifies load, test and set, or store, and size: word, double word, byte, half word); in the reverse direction, the code specifies the **trap code**, if any. The trap strobe signal indicates that the operation has terminated abnormally. The **address strobe** is asserted by the CPU to indicate that the address lines and function code lines are valid; the **response** signal is asserted to indicate that the CAMMU is either responding with data (on a load or instruction fetch), or has completed a store and is ready to accept another command.

The one line used by the ICAMMU and not by the DCAMMU is the **instruction send (ISEND)** line; the CPU asserts this line when the CPU instruction

Signal	Mnemonic	In/Out
Address/Data Bus	IADF<31:0>, ADF<31:0>	I/O
User Mode	MPU0	O
Protection Key	MPK	O
UU Bit from SSW	MPUOU	O
KU Bit from SSW	MPKU	O
Mapped Bit	MPM	O
Function Code/Trap Code	IFC<3:0>, FC<3:0>	I/O
Address Strobe	IASF, ASF	O
Send Instruction (ICAMMU)	ISEND	O
Response Signal	IRSP, RSP	I
Trap Strobe	ITSTB, TSTB	I
Inst/Data	ID	O

Signals on Processor/CAMMU Bus
Table 8

Signal	Mnemonic	In/Out
Address/Data Bus	AD<31:0>	I/O
Buffer Direction Control	DIR	O
Memory Space System Tag	TG<2:0>	I/O
Cycle Type	CT<5:0>	I/O
Cache Busy	CBSYi,CBSYd	O
Bus Lock	LOCK	O
Transfer Request	TR	I/O
Ready	RDYi	I
	RDYo,RDYoi	O
Bus Request	BRi,BRd	O
Bus Grant	BGi,BGd	I
Memory Error	MSBE,MMBE	I
Bus Error	BERR	I
Interrupt Vector Bus	IVEC<7:0>	I
Interrupt Request	IRQ	I
Interrupt Acknowledge	IACK	O
Non-Maskable Interrupt	NMI	I
Non-Mask. Int. Ack.	NMIACK	O
BCLK Rate Select	RATE	I
Bus Clock	BCLK	O
Master Reset	RESET	I
Unrecoverable Fault	URF	O
Apply Diagnostics	URDIAG	I

Table 9 - System Bus Signal Lines

Ratio to Fully Associative Miss Ratio						
Degree of Assoc:	1	2	4	8	16	inf.
Cache Type:						
Unified	1.52	1.19	1.07	1.02	1.01	1.0
Instructions	1.43	1.19	1.10	1.06	1.02	1.0
Data	1.60	1.23	1.10	1.03	1.01	1.0

Increase in miss ratio due to set associative mapping as compared to fully associative.
Table 10

buffer is able to accept the next sequential instruction; this avoids the need to send instruction addresses to the ICAMMU except when a branch takes place. The ICAMMU contains a local program counter and an incrementer which computes the next address when the ISEND line signal arrives; this is done simultaneously with the incrementing of the program counter on the processor chip. Because the CAMMU program counter is normally several bytes ahead of the program counter on the processor chip, it is considerably more convenient to have the additional PC on the CAMMU, rather than maintaining that additional PC in the processor; it is also much more efficient and faster than using using the processor PC for the same purpose.

The remaining line is to tell the CAMMU whether it is a D-CAMMU or an I-CAMMU, since the chips are identical.

6. The System Bus Interface

The system bus (CLIPPER bus) interface is closely related to the memory system architecture and also has a major effect on the system performance. For that reason, we provide a brief summary in this section of the interface between the CLIPPER module and the system bus. Further information is available in [Fair85].

Figure 19 shows the signal lines that connect to the interface, and a diagram showing how a simple CLIPPER system might be configured is in figure 20. The abbreviations for the signal line names are given in table 9. Some of the more interesting signal lines are described here.

There are 32 bidirectional **address/data lines (AD<31:0>)** along which addresses and data are transmitted. The **DIR** line specifies the direction of the transfer and is asserted by the bus master.

The **TG lines (TG<2:0>)** are the memory space system tag, and their encoding (see table 3) is the same as the encoding for the system tag bits of the TLB (figure 11) and the page table (figure 4). This tag must be placed on the system bus so that CAMMUs employing bus watching can determine the type of reference and whether CAMMU activity is required.

The **CT<5:0> (cycle type)** lines specify the type of bus operation in progress - see table 7. CT5 specifies whether the master is a CPU or I/O device, and CT4 indicates whether it is a read or write operation to memory. The size of the transfer is indicated by lines CT<3:0>, with transfers of 4 words, 1 word, 1/2 word or 1 byte possible.

The **RDYo signal** is asserted by a D-CAMMU to signal that it has preempted a memory read by an I/O device, and will provide the data itself.

The **CBSYi, CBSYd** signals are used to indicate that a CAMMU is busy doing internal operations associated with a bus watch operation; main memory data may not be placed on the bus while these lines are asserted, since the cache may yet preempt the memory and provide the data itself.

The **LOCK** signal is asserted by the bus master when it requires the bus for more than one operation. It is used in four cases: (a) DTU (translator) page table directory and page tables accesses. (b) DTU R (referenced) or D (dirty) bit modifications in page tables. (c) Read-modify-write (test and set) operations. (d) Cache line replace and fetch on cache miss; i.e. a quadword write followed by a quadword read.

The **RDY signal** is asserted by a slave when the slave places data on the A/D lines. A slow slave can introduce an arbitrary number of wait states by delaying the assertion of RDY.

Lines **IVEC<7:0>** constitute effectively a *separate interrupt bus*. This bus allows interrupt levels and numbers to be transferred to the CPU without regard to normal system bus activity, thereby reducing CPU interrupt response time and increasing effective bus bandwidth.

7. Timing and Performance

The normal way to measure the performance of a computer at the instruction execution level is to compute the average instruction execution time as the sum of three components: the time for the instruction in the execute stage of the pipeline, the additional delay induced by pipeline interlocks (averaged over the various preceding and following instructions) and the time for storage delays due to cache and TLB misses, and other memory contention [Macd84, Peut77]. In this section, we provide the basic data by which one can compute the performance of the memory system of the CLIPPER. Following that, we provide estimates of expected cache and TLB hit ratios, and then estimate the achievable level of performance for a CLIPPER based system as a function of the constraints introduced by the memory system.

7.1. Timing Figures

The CLIPPER machine cycle time is normally 30ns (33MHz) and all timings are multiples of that cycle time. The bus cycle time is double that, or 60ns. Most accesses across the CPU / CAMMU bus are full word accesses, and all times here are based on a full word reference by the CPU.

The time to do a non-sequential read from either CAMMU or a copy-back write to the DCAMMU is 4 cycles or 120ns. If a read is from the same quadword as was the most recently read, then the time required is only 2 cycles, or 60ns, since the target quadword is still in the quadword buffer (figure 15). The high order (4-31) bits of the virtual address are observed to be the same as previously, and the TLB and cache cycles are bypassed. This speedup for adjacent access will not only be useful for instruction fetch, where execution is sequential, but will also be useful for data reads, where context restores and array accesses often causes sequential data reads (see below). This feature does not apply to write operations.

The time to handle operations to main memory is a function of the speed of main memory, which can vary. The timing figures here assume a zero wait state memory, as is commonly assumed when timing figures are published.

The additional time for a read miss (quadword fetch) is 570-600ns (excluding the 120ns normally required for a read); the bus is busy to other processors for 420ns. (See figure 21 for an illustration.) The time for a write-through with or without cache update; i.e. hit or miss, is 210-240ns, with the bus being busy for 240ns to other processors. (The CAMMU provides an early response signal, rather than waiting for the store to entirely complete). The time for a write to cache on a write miss using copy back is 600ns, with 420ns bus busy. The additional time to replace a dirty line in cache when servicing a miss is 480ns, with 480ns bus busy.

The number of CAMMU cycles required for a bus watch operation was given for all relevant cases in section 4.6. Since the fraction of bus watch operations that hit in the cache should be very small, we will assume all bus watch operations are misses, and therefore busy the CAMMU for 6 CPU cycles. The number of additional bus cycles used by a bus watch operation is 2 (write), 3 (IO Read miss), 4 (IO Read hit) or 8 (IO Read Quadword hit).

The time to service a TLB miss is 960ns; the bus is busy for 600ns. (Additional time is required if the TLB miss causes a change in the value of a reference bit or dirty bit for a page, in which case the bit must be immediately written back to memory. The times in that case are respectively 1230ns and 960ns.)

The time of the remaining operations, such as that to issue commands to the CAMMU, are expected to be sufficiently infrequent that they will have no significant effect on system performance.

7.2. Projected Miss Ratios

From the results in [Smit85a], we project that a 4KByte DCAMMU and also a 4KByte ICAMMU will each have about a 10% demand miss ratio. The miss ratio for instructions drops to about 3% when using prefetch. From results to appear in [Smit86], (see also table 10), the miss ratio for 2-way set associative mapping should be about 20% higher than for fully associative mapping, and thus we believe the instruction cache miss ratio will be 12% without prefetching or 3.6% with, and the data cache miss ratio will be about 12%.

From the same simulations as were used for [Smit85a], we were able to determine that over 60% of all instruction references were to the same quadword (16 byte line) as the previous one, and over 35% of all data references were to the same quadword as the previous data reference.

In [Smit82,84] we note that measurements at Amdahl, taken on their 470V/6, showed a TLB miss ratio of about .2% for user state and .5% for supervisor state; that machine uses 4Kbyte pages, a 256 entry 2-way set associative TLB which uses an address space ID, and runs with the IBM MVS operating system. The only other existing TLB measurement is provided by [Clar85], in which it is reported that the TLB miss ratio for the DEC VAX 11/780, which has a 512 byte page, a 128 entry 2-way set associative TLB, and which purges the user entries from the TLB when task switching occurs is 2% to 4%, with about half the misses resulting from the purging of the user translation entries. Simulations based on traces taken by a hardware monitor appear in [Alex85], where a 128 entry, 2-way set associative TLB is predicted to have about a .3% miss ratio.

We note that CLIPPER has two TLBs, one on each CAMMU, and each is 128 entries, 2-way set associative. In effect, CLIPPER has a 4-way set associative, 256 entry TLB.

Based on the data cited immediately above, we feel that a TLB miss ratio of about .25%, or one miss for every 400 memory references, is a reasonable estimate. The reasoning behind that estimate follows; we note that the scarcity of relevant data and the inability to use traces gathered by other than hardware monitors to obtain an estimate requires this kind of very approximate reasoning. Because of the large, 4K pages, the miss ratio should be similar to that for the Amdahl 470V/6, but with the following differences and consequent adjustments: (a) The operating system is expected to be CLIX, a Unix port [Neff86, Sach85a], which is smaller than MVS and should have better miss ratio behavior. (b) The TLB is 256 entries, 4-way set associative, instead of 2-way set associative, which should lower the miss ratio slightly. (c) The system will be multiprogrammed to a much lower extent, which should lower the miss ratio. (d) The user translation entries must be purged when a new user address space is initiated, which will increase the miss ratio. (e) The data from [Alex85] is very close to our own estimate.

Data for the VAX 11/780 and IBM 370 in [Smit85a] indicate about 3.5 to 4 bytes of instruction fetch and 4-5 bytes of data R/W per instruction. Similar data appears in [Emer84] and [Peut77]. Since CLIPPER instructions are individually less powerful than for those machines, and since individual CLIPPER instructions can only do single loads and stores, we assume 3.5 bytes of instruction fetch per instruction, and 2 bytes of data read/write per instruction. Since the miss ratios above are per 4-byte memory reference, we adjust the instruction fetch miss ratio with and without prefetch to 3.6%*(3.5/4) = 3.15% and 12% * (3.5/4) = 10.5%; the data reference miss ratio becomes 12% * (2 / 4)= 6%.

In [Smit85a], it is shown that when using copy back, about half of the data lines pushed are dirty; we assume that that estimate holds here. It is also observed in that paper that data writes are approximately 1/3 of the data references; i.e. reads outnumber writes by 2/1.

Using the figures above, we can compute the average delay per instruction for main memory activity. First, we assume that all references are for copy-back. The instruction miss ratio is estimated to be 3.15%, and with a delay of 585ns for such a miss, that adds 18.4ns per instruction. (Actually, in effect slightly higher, since not all prefetches will arrive in time.) The 6% data miss ratio yields a fetch delay of 35.1ns (585ns x 6%). An additional 14.4ns (480ns x 3%) is needed to account for copy-backs. The delay due to TLB misses is 3.3ns (960ns x .25% x ((3.5 + 2.0)/4)). Thus the main memory component of the mean instruction time is approximately 71ns. To get the mean time per instruction, we must add the weighted sum of the execution times for each instruction, the effect of pipeline conflicts, and any additional delays due to instruction fetch [Macd84]. (Instruction fetch time will normally be overlapped with the execution of other instructions, especially considering the high hit ratio to the quad word buffer in the ICAMMU.) A paper with detailed analysis of this type is in preparation [Holl86].

7.3. Bus Traffic Analysis

In a multimicroprocessor system with a single shared bus, the factor that limits overall system performance is the bus bandwidth. In this section, we evaluate the load on the bus.

Assuming the same miss ratios as above in section 7.2, the bus occupancy (busy time) per instruction is ((1.25 x 12% x (3.5/4)) x 420 + 420 x 6% + 480 x 3.% + 600 x .25% x ((3.5 + 2.0)/4))) = 97ns. (The instruction fetch bus traffic is about 25% higher when using prefetch than for demand fetch [Smit85a].) Thus, the bus capacity, neglecting I/O and bus watch operations, is 10.3 millions of CLIPPER instructions per second (11.6 millions if instructions are demand fetched). If in addition, we assume that 20% of all writes are write throughs to shared pages (in a multiprocessor system), this adds 8ns (20% x ((1/3 x 2 bytes/inst.)/4) x 240ns) per instruction.

We note that our calculations all ignore I/O traffic on the bus. We believe that the I/O traffic (i.e. traffic to I/O devices) on the bus will be less than 10% of the processor fetch/store traffic, and most likely well below 5%. Given the rough calculations above, that amount is not material and hasn't been included.

8. Tradeoffs and Alternative Designs

In the design of the memory system architecture for CLIPPER, there were numerous parameters which had to be specified, such as cache size, organization, line size, TLB design, page size, etc. The reasons for some of the design choices are given above, and we discuss many of the remainder in this section.

8.1. Why Two Caches?

There are two common approaches to providing adequate memory bandwidth into and off of a microprocessor chip: separate instruction and data busses, or separate address and data busses; there aren't enough pins to do both. For CLIPPER, we have chosen the former; this is the same choice as was made for the Z8000 [Peut79]. The primary reasoning for this choice is quite simple: it is relatively easy to make parallel use of the instruction and data busses simultaneously in a pipelined machine, since one instruction is being fetched while a previous instruction is being executed; in this case, execution is a load or store. In addition, the dual instruction and data busses conveniently permit the use of two caches, one for instructions and one for data, thus doubling the cache capacity available on one chip. Further, there is no need to switch arriving data to either the I-unit or the E-unit, as would be required in the address/data bus case, and time for arbitration and control is accordingly less.

8.2. Page Size

One of the first choices to be made was the page size, which we also assume here to be the fixed I/O block size. There are a number of factors favoring a large page size: (a) From the data presented in [Smit82, 84], [Alex85] and [Clar85], it is clear that the TLB miss ratio drops sharply with increasing page size. (Each entry covers much more of the address space.) (b) From the data in [Mcku84] and the material presented in [Smit81], it is clear that I/O performance improves with increasing page size, up at least to blocks in the range of 4K to 8K bytes. (c) The larger the page size, the larger the number of bits that are not translated, and therefore the larger number of bits available to select the set in the cache, and therefore the larger the possible cache size given the current real address, overlapped fetch and translation design. (d) The larger the page size, the fewer the levels of mapping required. With 4K pages, a two level page table is required. With a 512 byte page, a four level page table would have been needed.

There are, however, risks in using a large page and I/O block size. First, if there are very large numbers of small files, then disk allocation will suffer greatly from internal fragmentation will be poorly utilized. Second, if most virtual memory pages are only partially utilized, then main memory may also be internally fragmented and poorly used. Third, larger pages or blocks take longer to transfer between disk and main memory. Finally, even though disk blocks or main memory pages may only be partially used, the full physical block or page will usually have to be transferred during I/O operations, which would load down the system bus.

In our view, the advantages of a large page size, at least in the range of 4Kbytes, significantly outweigh the disadvantages, and we chose a 4Kbyte page for CLIPPER. We believe that the DEC VAX suffers significantly from its small page size [Clar85, Mcku84], and prefer to move in the other direction. Further, the trends toward cheaper disk and main memory and higher I/O transfer speeds further recommend the larger page size. We note the foresight of IBM in providing its System/360 with a 4K page.

8.3. TLB Organization

The primary question in TLB organization was whether to make it direct mapped, set associative (2 or 4 way) or fully associative. In table 10, we present research results [Smit86] showing the extent to which the miss ratio seems to increase over that for a fully associative cache when set associative mapping is used. It seems clear that direct mapping is appreciably worse than 2-way set associative, and that only minor improvement is obtained over 2-way by going to 4-way set associative. While those results were determined by studying cache designs, we feel comfortable using them to estimate TLB performance. (See also [Smit78b].)

The other factor influencing the TLB design was the fact that it was possible to accommodate 128 entries in each TLB when a set associative design is used, whereas the fully associative design, given both silicon area limitations and the feasible associative logic, would only have only had 64 entries per TLB. While there is no systematic study available to exactly predict the miss ratio of the two alternate designs, we feel comfortable in predicting that the miss ratio for two 128 entry 2-way set associative TLBs will be lower than that for two 64 entry fully associative TLBs. The use of hardwired translation entries (see section 3.2) minimizes much of the penalty that would occur from mapping conflicts in certain TLB sets. For those reasons, the design described above, that of 128 entries organized as 64 sets of 2 elements each, was chosen.

Another major advantage to the two way set associative design is the robustness that it provides with respect to manufacturing errors in the TLB storage array. If there is a bad bit in one TLB entry, then that entry can be marked permanently invalid (via laser burning of fuses), while still permitting the TLB to function at a slightly reduced level of performance. Each such non-functioning TLB entry provides an increase in TLB miss ratio of (1/64)*((miss ratio for 64 entry direct mapped TLB)- (miss ratio for 128 entry 2-way set associative TLB)), which is a very small amount. Considering the small effect of the TLB miss ratio on the mean instruction time, the effect of defective TLB entries on processor performance should be negligible.

8.4. Cache Line Size

Cache miss ratios (for reasonable size caches) generally drop with increasing line size, but larger line sizes also increase the bus traffic. During the period during which the parameters for CLIPPER were selected, we used data from [Smit82, 84] to select the 16 byte line size, based on the tradeoff between bus traffic and miss ratio.

The choice of a 16 byte line was confirmed in [Smit85b], where 16 byte lines are seen to give close to minimal mean memory access delays (multiplying miss ratio by memory latency), while not increasing bus traffic much beyond that for 4 and 8 byte lines.

We believe that 16 bytes is the best choice for the line size for current technology. Since line size is not a programmer visible parameter, the line size can be increased in future implementations as technology changes. We expect that as the cache size increases and bus transmission rates increase, the preferable line size will grow, first to 32 bytes and then to 64.

8.5. Cache Size

The cache size was essentially determined by the decision to create a system with two CAMMUs, and to include on each CAMMU the TLB and translator. 4Kbytes was the maximum that fit on the chip.

8.6. Cache Organization

From the data in table 10, we can see that two way set associative mapping yields a significantly lower miss ratio than direct mapped. The improvement seems to more than justify the additional logic required to implement the set associative mapping. Increases to 4-way set associativity seem to provide only minor improvements in hit ratio while requiring considerable extra logic and data paths on the CAMMU chip. For that reason, the existing 2-way set associative design was chosen.

We note that of the 12 bits that are not translated, and are thus immediately available to select a set in the cache, only 11 bits are used immediately to access cache memory: The low order 4 bits select byte within line, and 7 of the remaining bits are used to select the set. With an appropriate scaling of the CMOS process used for CLIPPER, the cache size could be doubled by doubling the number of sets and making no other changes. A further increase to 16KBytes would be possible by changing to 4 way set associativity, which is still a feasible degree of associative search.

As noted above for the TLB, the 2-way set associative design means that a chip processing error which disables a single cache entry only increases the miss ratio slightly, while still leaving the chip functional. The increase in the miss ratio per such entry would be (1/128)*((miss ratio for 2Kbyte direct mapped cache)-(miss ratio for 2-way set associative 4Kbyte cache)), which we estimate (from [Smit85a] and table 10) to be .075% for the instruction cache and .054% for the data cache.

8.7. Alternative Cache Locations

The choice of a cache chip is intermediate between two other possibilities also used for microprocessor systems: an on-chip cache [Hill84] or a full board cache. In this section, we briefly summarize the reasoning that led us to the choice of a cache in the CAMMU.

8.7.1. On-Chip Cache

There are several reasons why we feel that with current technology, an on-chip cache would not be a good tradeoff. First, given the other functionality on the processor chip (floating point unit, pipelined execution unit), very little space for a cache would have been available - perhaps 256 bytes. With that size cache, and a 16 byte line, the miss ratio would have been about 30% [Smit85a,b], and with a 4 byte line, about 65%. (With a 16 byte line, most of the hit ratio comes from sequential access to the first line in the buffer. We obtain this same effect via the quadword buffer in the CAMMU.) For an instruction cache alone, the figures would have been respectively 25% and 59%. In addition, an extra machine cycle would likely have been required to reach the external cache, given the need to check the internal one. The net result is that a local cache would likely not have yielded a performance advantage; this same conclusion was reached for the 32332 design [Mate85]. We believe that on-chip cache designs such as that used in the 68020 [Moto84, Macg84] will not prove to yield sufficient performance advantage to be worthwhile.

There are other difficulties with the on-chip design. If it was to be a cache containing data, then steps would have to be taken to maintain consistency with the other caches in the system. If it was only an instruction cache, then data would still have been cached off chip. If the cache were a virtual address cache, it would have had to be purged when task switching, unless an address space identifier were used. If the cache were real address, then the memory management unit would have had to be on-chip as well.

The conclusion here is that an on-chip cache would not have yielded performance advantages to make up for the other functionality that would have had to be removed from the chip in order that there be room for the on-chip cache.

8.7.2. Board Size Cache

Board size caches have a number of advantages, including potentially a much higher hit ratio than is achieved by the CLIPPER CAMMU, but there are a number of reasons why we chose as we did. (1) It would have been/ will be very difficult to access a board level cache in the same time as the one chip CAMMU. (2) If the board cache were real address accessed, then the memory management unit would still have to go somewhere, and it (including the TLB and translator) didn't fit on the processor chip; thus a memory management unit chip would still have been required. In addition, the number of untranslated bits would be insufficient (see below) to address the cache, which would mean that the translation would have had to complete prior to cache access. If the cache were accessed on virtual access, then a complicated reverse translation buffer would be required [Smit82] to avoid synonyms and maintain consistency, and would also have to interface to the bus for the same purpose. To avoid purging a virtual address cache, a user address space ID would have had to been added to each virtual address tag. (3) The current CLIPPER module is an integrated solution to a customer need, whereas the board cache represents a "do it yourself" project for the customer. (4) Using 8 bit wide 64K static RAMs, 4 chips yields a 32Kbyte direct mapped unified cache, which yields a miss ratio of about 19% [Smit85b, table 10], which is significantly worse than we obtain.

Despite our argument above, the board level cache is potentially a good addition to a very high performance CLIPPER based system. The board level cache, as a cache level between the CAMMU and the bus, would have a number of advantages. It would be considerably faster than the main memory, and would significantly cut bus traffic, which would make systems with several CLIPPERs on the same bus feasible. It would be referenced via real addresses, and so wouldn't need a reverse translation buffer. A multilevel cache of this type is used in the Fujitsu 384 [Hatt83].

The conclusion here is that a board level cache is valuable, but in addition to the CLIPPER CAMMU, not instead of it.

8.8. Why Not a Virtual Address Cache

The CLIPPER cache is a "real address cache"; i.e. the virtual addresses are translated to real addresses before the cache line is selected. It has been suggested that there are some advantages to a virtual address cache [Hill85]: it is potentially faster, since no translation is required (although CLIPPER overlaps most of the translation time), and all of the bits are available for set selection (although CLIPPER has sufficient bits).

There are a number of disadvantages to a virtual address cache, however, which make it a poor choice for CLIPPER. First, a reverse translation buffer is needed to map real addresses back to virtual addresses, to avoid problems with "synonyms" - two virtual addresses that map to the same real address. Within the CPU, the synonym problem is difficult to avoid, especially as we are making use of the ability to have synonyms to (a) overlap user and system address spaces, (b) to share code among processes, and (c) to share data among processes using the copy-on-write facility [Neff86]. Even more difficult is the fact that addresses on the bus are real addresses, and in order to see if a reference is to something in the cache, that address would have to be reverse mapped. The translation process is used in CLIPPER to provide protection, tags, etc., and these other features would have to be implemented in some other way. Finally, the architecture would have had to be extended to use address space identifiers, so that multiple address spaces could share the cache.

The conclusion here is that while a virtual address cache is feasible, for CLIPPER it would have been a poor choice.

8.9. Address Space Identifier

Address space identifiers are used in some machines, such as the Amdahl 470 and IBM 370/168 [Smit82], to minimize the TLB miss ratio. If virtual addresses in the TLB are tagged with the address space ID, then it isn't necessary to purge the user TLB entries when a new user task is started.

We chose not to implement address space identifiers for several reasons: (1) They require space for storage, and signal lines and comparators to compare them. (2) The "true" address space identifier (ASID), the page directory origin, is too large to use as an ASID, so a shorter identifier would have to be created, and a mapping table from one to the other built. (3) We believe that with the large, 4K page that is used in CLIPPER, the potential performance gains from an ASID are small (see above). (4) Since the user TLB entries only need to be purged when starting a *different* user, i.e. one which is different than that running previously, the gains are limited in an environment where task switching is infrequent, as would be the case for high performance workstations. Results in [Alex85] suggest that an ASID provides negligible improvement under these circumstances.

The result here is that an ASID would be useful, but only slightly, and not enough to compensate for the silicon area and design complexity warranted. As a feature, it can be added to some future implementation of the CLIPPER architecture.

9. Conclusions

We've described in detail in this paper the Cache and Memory Management Unit, and the memory architecture, for the Fairchild CLIPPER processor. CLIPPER is designed to be a high performance microprocessor chip set, and as this paper demonstrates, the CAMMU design and implementation is appropriate to that goal. The memory architecture has been shown to be extremely versatile and powerful, and not likely to soon suffer from technological obsolescence. We've also presented an extensive discussion of the tradeoffs leading to our design choices. Overall, we believe that the CLIPPER CAMMU represents the best achievable technology/performance tradeoff, given the current state of technology, while permitting evolution in both the implementation and in the CLIPPER memory architecture.

Acknowledgements

We wish to thank the entire team responsible for designing and implementing CLIPPER. We especially note Walt Hollingsworth, David Neff, Laura Neff, John Kellum, Ray Ryan, Jean-Paul Meunier, Kevin Norman, Kevin Kissel, Vern Brethour and Rich Dickson, who had major roles throughout the project, and Jin Kaku who was responsible for much of the CAMMU circuit design.

Bibliography*

[Alex85] Cedell Alexander, William Keshlear, and Faye Briggs, "Translation Buffer Performance in a UNIX Environment", Computer Architecture News, 13, 5, December 1985, pp 2-14.

[Arch85] James Archibald and Jean-Loup Baer, "An Evaluation of Cache Coherence Solutions in Shared-Bus Multiprocessors", Tech. Rpt. 85-10-05, October, 1985, Computer Science Dept., University of Washington, Seattle, Wash.

[Cho86] James Cho and Jin Kaku, "A 40K Cache and Memory Management Unit", Technical Report, Fairchild Advanced Processor Division, to appear, ISSCC, 1986.

[Clar85] Douglas Clark and Joel Emer, "Performance of the VAX 11/780 Translation Buffer: Simulation and Measurement", ACM TOCS, 3, 1, February, 1985, pp. 31-62.

[Emer84] Joel S. Emer and Douglas W. Clark, "A Characterization of Processor Performance in the VAX-11/780", Proc. 11'th Ann. Symp. on Computer Architecture, June, 1984, pp. 301-309.

[Fair85a] Fairchild, "CLIPPER Module Product Description", Fairchild Camera and Instrument Corporation, October, 1985.

[Fair85b] Fairchild, CLIPPER 32-Bit Microprocessor Module Instruction Set", October, 1985.

[Fuji82] Fujitsu Corp., "FACOM M-382", third edition, September, 1982, Tokyo, Japan.

[Good83] James R. Goodman, "Using Cache Memory to Reduce Processor-Memory Traffic", Proc. 10'th Ann. Int. Symp. on Comp. Arch., June, 1983, Stockholm, Sweden, pp. 124-131.

[Hatt83] Akira Hattori, Minoru Koshino and Shigemi Kamimoto, "Three Level Hierarchical Storage System for Facom M380/382", Proc. IFIP Conf., September, 1983, pp. 693-697.

[Henn84] John Hennessy, "VLSI Processor Architecture", IEEETC, C-23, 12, December, 1984, pp. 1221-1246.

[Hill84] Mark Hill and Alan Jay Smith, "Experimental Evaluation of On-Chip Microprocessor Cache Memories", Proc. 11'th Annual Symposium on Computer Architecture, June, 1984, Ann Arbor, Michigan, pp. 158-166.

[Hill85] M. D. Hill, S. Eggers, J. R. Lazarus, G. S. Taylor, G. Adams, B. K. Bose, G. A. Gibson, P. M. Hanson, J. Keller, S. I. Kong, C. G. Lee, D. Lee, J. M. Pendleton, S. A. Ritchie, D. A. Wood, B. G. Zorn, P. N. Hilfinger, D. Hodges, R. Katz, J. Ousterhout, and D. A. Paterson, "SPUR: A VLSI Multiprocessor Workstation", December, 1985, University of California, Berkeley, Computer Science Division Technical Report UCB/CSD 86/273.

[Holl86] Walt Hollingsworth, Alan Jay Smith and Howard Sachs, "Analysis of Processor Performance in the Fairchild CLIPPER", in preparation.

*The various papers marked here as "in preparation" should be completed by the time this paper is finally published.

[IBM76] IBM System/370 Principles of Operation, Form Number GA22-7000-5, IBM Corporation, Poughkeepsie, New York, 1976.

[Katz85] R. H. Katz, S. J. Eggers, D. A. Wood, C. L. Perkins, and R. G. Sheldon, "Implementing a Cache Consistency Protocol", Proc. 12'th Ann. Int. Symp. on Comp. Arch., June, 1985, Boston, Mass, pp. 276-283.

[Levy80] Henry Levy and Richard Eckhouse, "Computer Programming and Architecture: The VAX-11", Digital Press, Bedford, Mass., 1980.

[Macd84] Myron H. MacDougall, "Instruction Level Program and Processor Modeling", IEEE Computer, July, 1984, pp. 14-24.

[Macg84] Doug MacGregor, Dave Mothersole and Bill Moyer, "The Motorola MC68020", IEEE Micro, August, 1984, pp. 101-118.

[Mate84] Richard Mateosian, "System Considerations in the NS32032 Design," Proc. NCC, 1984, pp. 77-81.

[Mate85] Richard Mateosian, "National's NS32332 CPU: A Graceful Extension of the Series 32000", Profession Program Session Record, Wescon/85, November, 1985, San Francisco, Ca., Session 1: 32-Bit Microprocessors - Part I.

[Mcku84] Marshall K. Mckusick, William Joy, Samuel Leffler, and Robert Fabry, "A Fast File System for UNIX", ACM TOCS, 2, 3, August, 1984, pp. 181-197.

[Moto84] Motorola Corporation, "MC68020 Technical Summary", 1984.

[Neff86] Laura Neff, "CLIPPER Microprocessor Architecture Overview", Proc. Compcon, March, 1986, San Francisco, Ca., pp. 191-195.

[Papa84] Mark Papamarcos and Janak Patel, "A Low-Overhead Coherence Solution for Multiprocessors with Private Cache Memories", Proc. 11'th Ann. Int. Symp. on Comp. Arch., June, 1984, Ann Arbor, Michigan, pp. 348-354.

[Pate85] Anil Patel, "Z80,000 32-bit Microprocessor", Proc. NCC 1985, pp. 225-231.

[Patt85] David Patterson, "Reduced Instruction Set Computers", CACM, 28, 1, January, 1985, 8-21.

[Peut77] Bernard Peuto and Leonard Shustek, "An Instruction Timing Model of CPU Performance", Proc. 4'th Ann. Symp. on Computer Arch., College Park, MD, March, 1977, pp. 165-178.

[Peut79] Bernard L. Peuto, "Architecture of a New Microprocessor", IEEE Computer, 12, 2, February, 1979, pp. 10-21.

[Radi82] George Radin, "The 801 Minicomputer", Proc. Sigarch/Sigplan Symp. on Arch. Support for Prog. Lang. and Op. Sys., March, 1982, Palo Alto, Ca. pp. 39-47.

[Ritc74] Dennis M. Ritchie and Ken Thompson, "The UNIX Timesharing System", CACM, 17, 7, July, 1974, pp. 365-375.

[Sach85a] Howard Sachs and Walt Hollingsworth, "A High Performance 846,000 Transistor Unix Engine: The Fairchild CLIPPER", Proc. ICCD, October, 1985, Port Chester, New York, pp. 342-346.

[Sach85b] Howard Sachs, "The Fairchild CLIPPER Microprocessor Family, A High Performance 32-Bit Processor", Proc. Professional Program Session Record, Wescon / 85, November, 1985, Session 6, 32-Bit Microprocessors, Part II.

[Sach86] Howard Sachs, Walt Hollingsworth, and Alan Jay Smith, "Instruction Set Architecture and Processor Chip Design for the Fairchild CLIPPER", in preparation.

[Smit78a] Alan Jay Smith, "Sequential Program Prefetching in Memory Hierarchies", IEEE Computer, 11, 12, December, 1978, pp. 7-21.

[Smit78b] Alan Jay Smith, "A Comparative Study of Set Associative Memory Mapping Algorithms and Their Use for Cache and Main Memory", IEEETSE, SE-4, 2, March, 1978, pp. 121-130.

[Smit81] Alan Jay Smith, "Input/Output Optimization and Disk Architecture: A Survey", Performance Evaluation, 1, 2, 1981, pp. 104-117.

[Smit82] Alan Jay Smith, "Cache Memories", Computing Surveys, 14, 3, September, 1982, pp. 473-530.

[Smit84] Alan Jay Smith, "CPU Cache Memories", to appear in *Handbook for Computer Designers*, ed. Flynn and Rossman.

[Smit85a] Alan Jay Smith, "Cache Evaluation and the Impact of Workload Choice", Report UCB/CSD85/229, March, 1985, Proc. 12'th International Symposium on Computer Architecture, June 17-19, 1985, Boston, Mass, pp. 64-75.

[Smit85b] Alan Jay Smith, "Line (Block) Size Selection in CPU Cache Memories", June, 1985. Available as UC Berkeley CS Report UCB/CSD85/239. To appear, IEEETC.

[Smit85c] Alan Jay Smith, "CPU Cache Consistency with Software Support and Using "One Time Identifiers"", Proc. Pacific Computer Communication Symposium, Seoul, Republic of Korea, October 22-24, 1985, 142-150. Also available as UC Berkeley Computer Science Report.

[Smit86] Alan Jay Smith, "Quantification of the Effect of Set Associative Mapping and Its Performance Relative to Fully Associative and Direct Mapping", in preparation.

[Swae86] Paul Sweazey and Alan Jay Smith, "A Class of Compatible Cache Consistency Protocols and Their Implementation on the IEEE Futurebus", to appear, Proc. 13'th Ann. Symp. on Computer Architecture, June, 1986, Tokyo, Japan.