FAIRCHILD

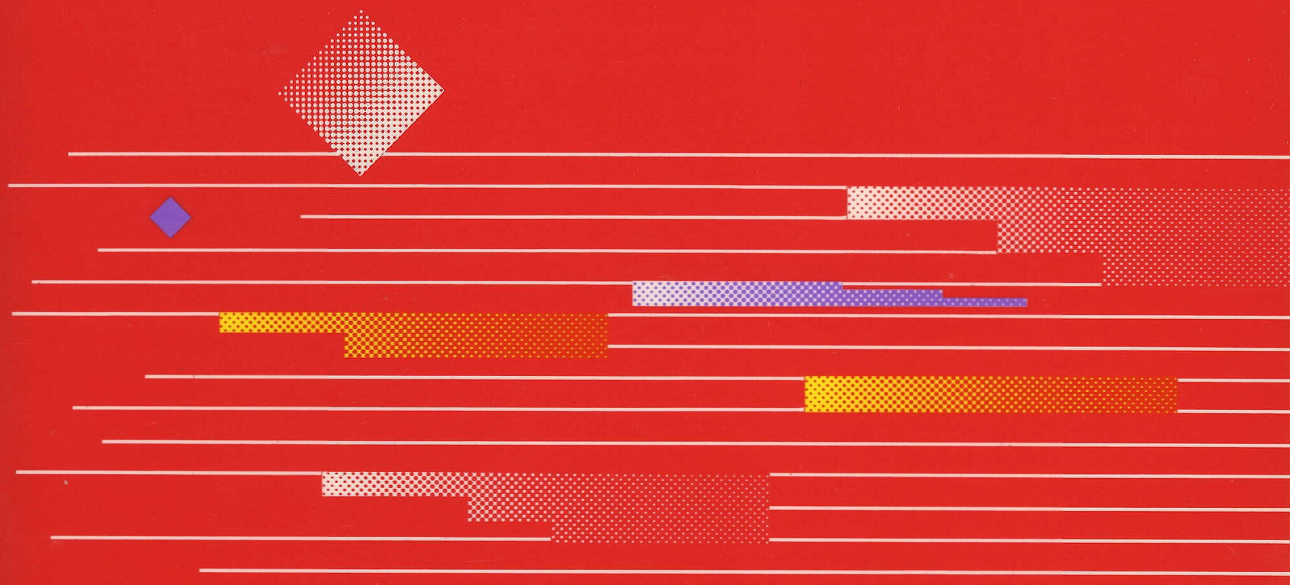
A Schlumberger Company

CLIPPER™

32-Bit Microprocessor

May 1986

Introduction to the CLIPPER Architecture



FAIRCHILD

A Schlumberger Company

CLIPPER™ 32-Bit Microprocessor

INTRODUCTION TO THE CLIPPER ARCHITECTURE

CLIPPER™ 32-BIT MICROPROCESSOR

INTRODUCTION TO THE CLIPPER ARCHITECTURE

TABLE OF CONTENTS

PREFACE

This document provides an overview of the architecture of the CLIPPER 32-bit microprocessor for software engineers, hardware engineers, and managers.

Managers should particularly consult Chapter 1, which summarizes some of the key features and benefits of CLIPPER. The remaining four chapters describe particular architectural topics. Chapter 2 addresses the internal architecture (caches, pipeline); Chapter 3 covers the instruction set, registers, and addressing modes; Chapter 4 describes the memory architecture, including the support for virtual memory; and Chapter 5 covers software and support.

CHAPTER 1
OVERVIEW

The CLIPPER is the fastest microprocessor in the world. It brings supercomputer and mainframe technology to a 32-bit microprocessor, while maintaining compatibility with existing software.

The advanced architecture of the CLIPPER permits it to achieve a maximum execution rate of 33 million instructions per second (MIPS), with an average rate in excess of the performance of a VAX 8600. Its built-in floating point unit performs at greater than 1 million floating-point operations per second (MFLOPS). The CLIPPER has a 4 gigabyte (2^{32}) physical address space, and it includes hardware support for demand paged virtual memory.

Clearly the CLIPPER was designed for applications that demand high performance — engineering workstations, graphics subsystems, and the latest industrial automation products. In fact, the CLIPPER makes possible for the first time many computation and memory intensive applications, such as speech recognition and robot vision systems, in a cost-effective package. Until the CLIPPER became available, these projects required superminicomputers, mainframes, or even supercomputers.

Despite its many advanced features, the CLIPPER was designed to be compatible with existing software. By providing optimizing compilers targeted to the CLIPPER for several standard programming languages, and a CLIPPER implementation of the UNIX™ standard operating system, Fairchild has ensured that most existing programs can easily be moved to the CLIPPER. Fairchild also offers a complete set of software development utilities for the CLIPPER, including interactive debuggers and simulators.

The following sections of this chapter discuss the innovative features of the CLIPPER architecture, including

— Supercomputer and mainframe-derived features, such as dual buses, advanced cache and pipeline designs, and concurrent processing units.

— The CLIPPER's streamlined instruction set (the most frequently used instructions execute in one clock cycle — 30 ns).

— Hardware support for virtual memory and other operating system functions.

— The architecture of CLIPPER-based systems, including the CLIPPER module itself, the I/O subsystem, and the memory subsystem.

— The high-level language and operating system "architecture" of the CLIPPER.

But first, we will explain exactly what we mean by computer architecture, and why the world needs another one.

1.1 WHY A NEW ARCHITECTURE?

The term **computer architecture** is often used as if it meant simply "the organization and design of a computer". In fact, however, there are several distinct architectures within a computer system, each defined by the boundary between different levels of the system. Figure 1-1 shows an abstract picture of a computer system, with the simplest operations and functions on the bottom and the most complex user-dependent operations on the top. Each level makes use of the functions provided by the level below. For simplicity, we have shown only four levels, and thus three architectures.

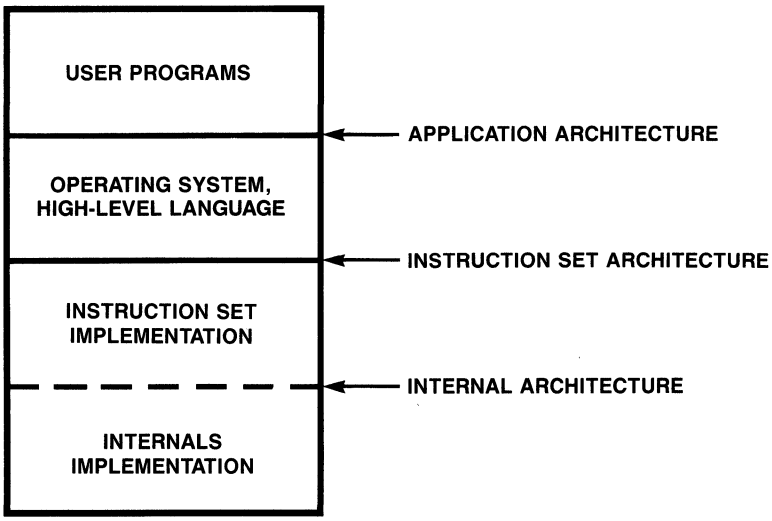


Figure 1-1 Computer Architectures

The **application architecture** is the interface between the application program and the high-level programming language plus operating system. Most application programs are written in standard programming languages, such as C, Pascal, or FORTRAN, and most are designed to execute on standard operating systems, such as UNIX. The language specification and operating system calls define the application architecture.

The **instruction set architecture** is the interface between assembly language programs and the computer hardware. It is defined by the registers, instruction formats, and addressing modes of the processor.

The **internal architecture** is defined by the gates, buses, caches, and execution units of the processor. This architecture is normally invisible to user-supplied software, but it has a great bearing on performance, as we shall see.

The CLIPPER has new internal and instruction set architectures, although it preserves a standard application architecture. These new low-level architectures incorporate technologies derived from supercomputers and mainframes, technologies that have never been applied to microprocessors before. Because of its new architecture, the CLIPPER has achieved a quantum leap in performance over other microprocessors — and that is the answer to "why another architecture?"

1.1.1 The Architectural Performance Gap

The advanced features of the CLIPPER architecture are not new to the world of computers in general, they are only new to the world of microprocessors. This fact explains why there has been a performance gap between microprocessors and large computers, such as mainframes and superminicomputers, even when the microprocessors ran at the same clock rate as the big machines and had the same word size.

Figure 1-2 shows a schematic representation of computer evolution. The lower, ascending line represents the increasing power of microprocessors from generation to generation, with the latest 32-bit microprocessors at the high-end. The upper, descending line represents the migration of supercomputer technology down from machines such as the Cray to mainframes and superminicomputers. The **architectural performance gap** in

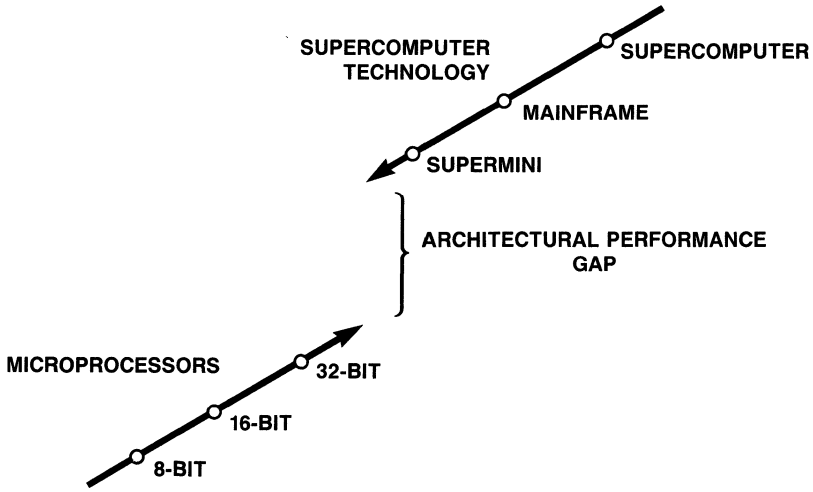


Figure 1-2 The Architectural Performance Gap

the middle is the result of microcomputer designers failing to incorporate advanced architectural features. It is this gap that the CLIPPER has jumped. In fact, in many ways the CLIPPER is a low-end supercomputer rather than a high-end microprocessor.

Now when microprocessors were first developed, the sheer feat of getting a computer on a chip was so impressive that it almost seemed uncharitable to ask for more. And the severe size limitations of these early processors precluded many fancy techniques. But the latest generation chips contain almost half a million transistors, and there are no longer any excuses for not employing the latest architectural technology as well. However, some 32-bit microprocessors seem to have been designed to make marginal improvements on existing 16-bit designs (that were in turn marginal improvements on 8-bit designs, and so on back to the original 4-bit microprocessor). The CLIPPER was built for the future instead of the past.

1.1.2 Compatibility—Preserving Your Software Investment

Since an increasing portion of the value of computer-based systems is in the software investment, preserving this investment has become a central concern of system implementers. The CLIPPER was designed to insure that our customers could get the best of two worlds — they can reap the performance benefits of supercomputer technology while preserving their existing software investment.

Until recently, this was not possible. Designers were basically faced with the choice of keeping their software, with only marginal performance gains possible, or moving to a new generation in performance, but having to scrap the existing software. The reason for this painful choice was that truly compatible computers had to be **instruction set compatible,** that is, their instruction set architectures had to be the same.

Instruction set compatibility (often called **binary compatibility** because the binary versions of application programs will run on a compatible instruction set) was important for two reasons: (1) most programs then were written in assembly language, and (2) no standard operating system existed. In recent years this situation has changed drastically. Now most application programs (in fact, most system programs) are written in a *high-level language,* and *UNIX System V* has emerged as the *de facto* standard operating system, available now on dozens of different computer architectures. Figure 1-3 lists some of the manufacturers committed to UNIX System V.

AT&T Information Systems
Altos Computer Systems
Alpha Micro
Amdahl Corporation
Apollo Computers Inc.
Arete
BBN Computer
Bull
Callan Data
Celerity Computing
Centurion
Charles River Data Systems
Control Data Corporation
Convergent Technologies
Convex Computer Corporation
Cray Research Inc.
Cromemco
Data General
Daisy Systems
Digital Equipment Corp.
Durango Systems
Elxsi
Encore Computer Corporation
Flexible Computer Corporation
Gould Inc.
Harris Corporation
Heurikon Corp.
Hewlett-Packard
Honeywell Information Systems, Inc.
IBM Corp.
ICL
Intel
Intergraph Corporation
Lanier
Motorola/Four Phase Systems
NCR Corporation
National Semiconductor Corporation
Nixdorf
Olivetti
Perkin-Elmer Corporation
Plexus Computers Inc.
Prime Computer, Inc.
Pyramid Technology Corp.
Ridge Computers
Sequent Computer Systems, Inc.
Siemens AG
Silicon Graphics
Sperry Corporation
Sun Microsystems, Inc.
Texas Instruments
Tandy Corporation
Tektronix, Inc.
Three Rivers
Tolerant Systems, Inc.
Toshiba
Valid Logic
Wicat
Zilog, Inc.

Figure 1-3 Partial List of Manufacturers Committed to UNIX System V

As a result of this change computer manufacturers have been freed from the crippling constraint of instruction set compatibility with antiquated architectures. The compatibility that counts now is **application architecture** compatibility. And the customer gets the benefit, because much larger performance gains can be achieved that preserve compatibility at this level. Appendix A discusses compatibility issues in more detail.

1.2 SUPER PERFORMANCE

The high performance of the CLIPPER, as with mainframes and supercomputers, is the result of a balance among several factors and of careful tuning; it is not the result of one all-powerful feature. The most obvious starting point is the clock speed: the CLIPPER runs at a clock rate of 33 MHz, twice the standard speed of other 32-bit microprocessors. But in order to utilize that clock rate, the architects of the CLIPPER had to carefully balance and tune four other key elements of the internal architecture:

1. Caches.
2. Internal dual buses.
3. Pipelines.
4. Integrated execution units.

In addition, the instruction set is designed to use simple, register-to-register instructions, the most common of which execute in one clock cycle, and special hardware support is provided for critical operating system functions, such as virtual memory. The next four subsections will cover the four elements of the CLIPPER's internal architecture that contribute to performance, while the remaining sections will discuss features of the instruction set and of the OS support mechanisms.

Figure 1-4 shows a block diagram of the CLIPPER module. The module contains a CPU chip and two Cache-MMU chips (called **CAMMUs**). One CAMMU is dedicated to instructions, the other to data, and each is connected to the CPU by its own dedicated bus. The CPU contains an on-chip Floating Point Unit (FPU). We will refer repeatedly to this diagram in the following subsections.

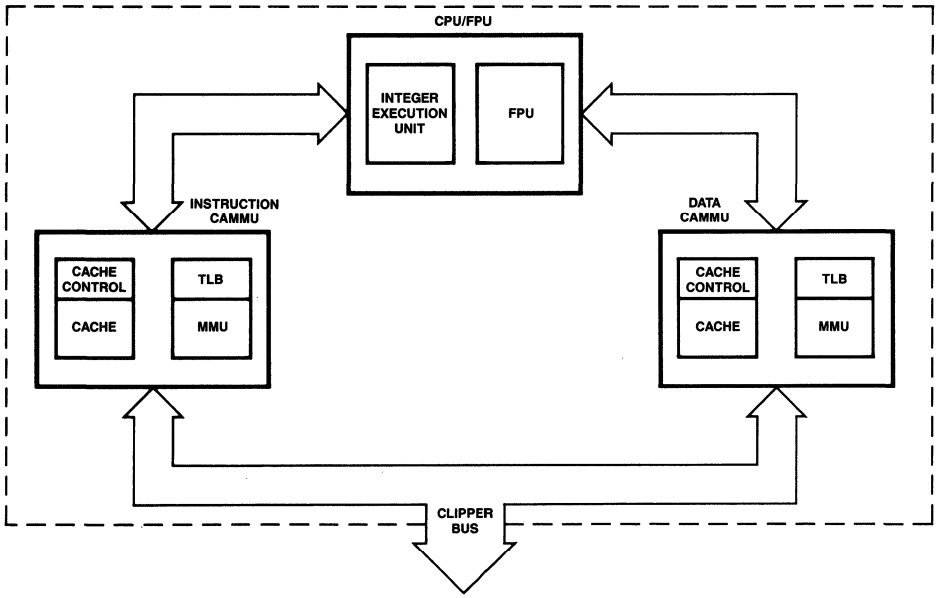


Figure 1-4 CLIPPER Module Block Diagram

1.2.1 Advanced Cache Management

Caching is basically a cost effective way of creating a computer system that looks like it has a very fast (and very expensive) main memory — but doesn't. The cache is a small, fast buffer with an access time that is intermediate (60-120 ns) between CPU registers (30 ns) and the low speed, inexpensive dynamic RAMS (150 ns) that are actually used to build the main memory. The caching hardware fills this buffer with recently accessed instructions and data; then at some later time, the processor often finds the data or instruction it wants in the cache and doesn't have to go all the way to main memory. (This occurs because programs often exhibit **locality**, that is, they tend to access the same address after a short time interval — temporal locality, or they access nearby addresses — spatial locality.) The **effective access time** is thus usually much lower than for main memory, although the actual number depends on the **cache hit rate**, which in turn depends on the cache design and the actual instruction mix.

If cost is no obstacle, main memory can be simply built out of 30 ns chips, and this is exactly how supercomputers are designed. Consequently, many supercomputers do not use caches. Mainframe computers, on the

other hand, make extensive use of caching, and the CLIPPER's caching mechanisms are largely derived from the experience of mainframes. The details of the cache architecture are covered in Chapter 2; we will simply summarize the facts here.

Each CAMMU contains three caches:

— a 4K byte, 2-way associative data or instruction cache, with a 16 byte line length (these terms are covered in detail in Chapter 2)

— a quadword buffer, that acts as a superfast cache-within-the-cache

— a Translation Lookaside Buffer (TLB) that caches 128 frequently used virtual-to-real address translations

When evaluating cache performance, one key metric is **hit rate**, the probability that the requested item is found in the cache instead of slower speed memory. Another key figure is **CPU access time**, the time the CPU requires to access an item. CPU access time includes the raw access time of the memory or cache chip plus the transfer time over the bus. For example, as we said above, the raw access time of a DRAM chip is about 150 ns, but the actual CPU access time to main memory is around 500 ns, because of bus transfer times and the resulting wait states.

The cache has a hit rate of greater than 90% and a CPU access time on a cache hit of 120 ns. The quadword buffer has a hit rate of 58% (instructions) or 41% (data) and a CPU access time of 60 ns (hit). The TLB has a hit rate of over 99%. The result of the complex interplay of the three caches is an effective CPU access time to memory of less than 100 ns (instructions) or 150 ns (data). This means that the CLIPPER behaves as if its main memory were made up of high-speed static RAMs and the bus had no wait states, even though it really uses inexpensive DRAMs and a less complex bus.

In order to implement this sophisticated caching system, the CLIPPER employs many advanced mechanisms, including:

— Bus Watch: a mechanism for guaranteeing cache consistency in a copy back caching strategy (see Chapter 2).

— Simultaneous access to the TLB for the physical address and to the main cache for instructions or data at that address; thus, there is no extra delay for address translation.

— Burst memory transfers (see below).

1.2.2 Increasing Bus Bandwidth

The factor that limits the performance of most microprocessors is **bus bandwidth**, the amount of data per unit of time that can be transferred over the key buses in the system. One of the most distinctive features of the CLIPPER internal architecture is the use of two buses between the CPU and the CAMMUs; one bus is used for data, the other for instructions. The use of two buses effectively doubles the bus bandwidth in this critical region. The use of two buses and two CAMMUS makes the caches more efficient as well, because locality is much stronger for pure instructions and pure data than for a mixture of the two.

The CLIPPER also maximizes the bandwidth of the CLIPPER bus between the Module and main memory. Through the use of burst mode transfers, the CLIPPER is able to send 16 data bytes (4 words) for each address word. Conventional buses transfer one word of data for each address word.

1.2.3 Sophisticated Pipelining

The CLIPPER has a pipelining system that is very similar to that of supercomputers. It consists of a pipeline with three phases:

1. fetch
2. decode
3. execute

(Since the CLIPPER is a load/store machine, it doesn't need a separate address calculation stage or a separate address pipeline. The address in a load or store is simply calculated during the execute phase.) The CLIPPER pipeline also has some significant novelties:

— The execution unit itself is partitioned into **three stages**, which execute in parallel and can overlap for successive instructions. Each stage takes only one clock cycle to complete, so the effective throughput is typically one instruction per clock.

— The CLIPPER includes a hardware resource manager that handles resource contention automatically. (In supercomputer terminology, this is called **scoreboarding**.)

— The execution unit includes a feedback mechanism so that intermediate results from one calculation are immediately available as input for the next calculation.

— The instruction CAMMU prefetches instructions into the CAMMU from main memory. It is able to do this because a copy of the program counter is maintained on the CAMMU.

1.2.4 Integrated Processing Units

One reason for the CLIPPER's spectacular performance is its high level of integration. The CLIPPER Module contains just three chips, instead of the dozens of chips that would be required to duplicate its functionality in more conventional microprocessor systems. The CAMMU alone replaces more than 30 chips in conventional cache designs. But beyond simply increasing the level of integration, the CLIPPER architecture was carefully partitioned to separate functions that should be integrated onto one chip from functions that should be on separate chips (given the current state of the art in VLSI).

An example of this careful partitioning is in the relationship among the CLIPPER's CPU, FPU, Cache, and MMU. Of course eventually all these functions may be integrated onto one chip; but that would require almost a million transistors, and VLSI technology is still a few years away from that level. For the present, more than one chip is required; so the question is, which functions should be grouped together and which should be separated?

One traditional approach is to combine the CPU and MMU, and leave the FPU and cache as separate chips (actually, in the traditional approach the user implements the cache). Figure 1-5 shows a diagram of this approach. This strategy has several disadvantages:

— Since the process of generating an effective address and translating it are inherently sequential, the CPU + MMU chip needs a two clock-cycle execution unit instead of a one cycle unit.

— Even though the MMU stands logically *between* the FPU and the cache (as it does between the CPU and the cache), the FPU has no direct access to the MMU, and the CPU is burdened with an overhead for every FPU address translation.

— Since the cache is left out of the integration picture altogether, no benefit can be derived from overlapping memory translations with cache accesses.

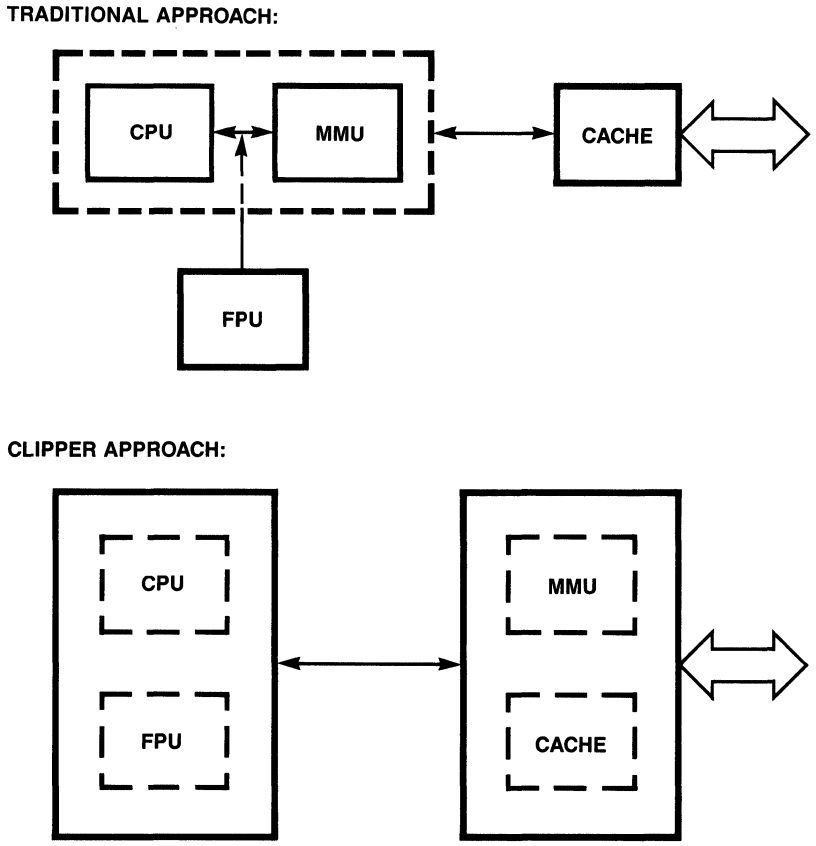


Figure 1-5 Functional Integration Strategies

The CLIPPER architecture, on the other hand, integrates all computational functions onto one chip (the CPU/FPU) and all memory access functions onto another (the CAMMU). The CPU/FPU chip contains both an integer execution unit and an on-board floating point unit, and the CAMMU contains both an MMU and a cache memory. On each chip the units can execute concurrently. Floating-point calculations can be done at the same time integer instructions are executing, and address translation occurs concurrently with cache lookup.

In fact, if you add the three phases of the pipeline, the three stages of the integer execution unit, the FPU, the cache prefetch, the address translation, and the cache lookup, as many as ten operations can be happening simultaneously on the CLIPPER with every clock cycle. And the clock runs at 33.3 MHz!

1.3 STREAMLINED INSTRUCTION SET

The CLIPPER's instruction set was designed for rapid execution of compiled code, especially code produced by compilers that optimize register usage. All unnecessary operations that are hard for compilers to use have been eliminated, the operations that remain are implemented with hard-wired logic, not microcode, and many execute in one clock cycle. In many ways, the CLIPPER's instruction set resembles the scalar instruction set of a supercomputer.

All arithmetic and logical instructions manipulate data in registers; only loads, stores, calls, and branches access memory directly. Load and store instructions are supplied with a set of nine addressing modes for efficient access to the elements of typical high-level language data structures, such as arrays, records, and arrays of records. All addresses are unsegmented 32-bit quantities; thus the logical address space and the physical address space are both 4 gigabytes.

The CLIPPER contains two sets of sixteen 32-bit registers, with one set provided for user programs and one set for the operating system. The registers are general purpose and may be used to contain operands or addresses. In addition, the on-chip FPU has eight 64-bit floating-point registers. The CLIPPER has instructions for manipulating signed and unsigned bytes, halfwords (16 bits), words, and single- and double-precision floating-point numbers.

A Macro Instruction Unit on the CPU contains sequences of instructions that implement more complex operations, such as string manipulation and operating system support. The Macro instructions access their own private registers and need not use general purpose registers, so their execution places no additional burden on programs. The combination of the load/store instruction set plus the Macro Instruction Unit gives the CLIPPER many of the advantages of both a reduced instruction set computer (RISC) and a complex instruction set computer (CISC).

1.4 VIRTUAL MEMORY AND OPERATING SYSTEM MECHANISMS

The CLIPPER provides hardware support for demand paged virtual memory, plus support for fast operating system calls and context swaps.

In many computer systems the logical address space is far larger than the actual main memory hardware. **Virtual Memory** is a mechanism implemented by the operating system (usually with hardware support) for circumventing the limits on physical memory size. Under a virtual memory system, it appears to users as if the entire logical address space were available for storage. Programmers can thus leave management of memory to the OS and never have to write overlays or worry about running out of memory.

The CLIPPER's virtual memory mechanisms are based on 4K byte pages of memory. At any given moment, only a few pages of the logical address space need be mapped into physical memory by the CAMMUs. The other pages are stored on disk, whose cost-per-bit is more economical. The CLIPPER keeps track of several items of information about each page, including the traditional Valid, Dirty, and Referenced bits, as well as protection data and caching specifications. The CAMMUs automatically trap to the operating system when an instruction attempts to reference a page that is not present in memory; the OS can then swap in the missing page, and the processor will restart the instruction. This whole operation is completely invisible to the end user.

The TLBs mentioned above are caches (one on each CAMMU) that each contain mapping information for 128 frequently used pages. Thus the CLIPPER has immediate access to 256 pages or one million bytes of memory and therefore can translate most addresses without ever having to consult a memory-based page table (see Chapter 4 for more details on mapping). The typical hit rate for the TLB is greater than 99% for a single process, which may seem like overkill, but the access time for a TLB miss (having to look up an address in a page table) is at least 10 times the access time for a TLB hit. So going from a 98% hit rate to a 99% rate yields an 8% performance improvement. It's all part of the CLIPPER's performance tuning.

The other OS support mechanisms are designed to make possible fast system calls. The separate register files for system and user mean that no context loading and storing is required when a system call is made, thus significantly reducing the system overhead. Exception processing is handled in the Macro Instruction Unit, which makes use of its own private registers instead of the system's registers. System calls can be vectored directly to the appropriate call handler instead of having to go through a common routine that calls the

proper handler. Finally, the interrupt offset is passed on a separate interrupt bus, so no bus arbitration is required for external devices to send an interrupt vector.

1.5 SOFTWARE AND SUPPORT

Fairchild supplies a complete development environment to go with the CLIPPER. In addition, a high-performance implementation of UNIX System V is available. The UNIX port makes use of the CLIPPER's demand paging and caching hardware to implement a high performance virtual memory system.

The development environment includes optimizing compilers for C, Pascal, and FORTRAN, all of which can either be hosted on VAX/ULTRIX systems or on the CLIPPER running System V. Along with the compilers are an assembler and linker, plus a VAX-hosted CLIPPER simulator, and an interactive debugger. Subsequent products will include a real-time operating system, source level symbolic debugger, additional compilers and additional hosts for the development environment.

There are Fairtec design centers across the country, staffed by application engineers and system designers. In addition, Fairchild has an active third party software program. The result of it all is complete support — hardware and software — for our customers.

1.6 SYSTEM ARCHITECTURE

The CLIPPER Module is really an 846,000 transistor compute engine that happens to be implemented on three chips instead of one. But the module is supplied as a single unit, complete with clock logic, all on a small circuit board. The module plugs into the main motherboard.

A complete CLIPPER-based system will include more than the Module, however. As can be seen in Figure 1-6, the Module interfaces to the CLIPPER bus — a 32-bit, fully synchronous data/address bus — which acts as the main system bus. Also attached to the CLIPPER bus will be the two other main subsystems:

— the I/O subsystem, usually with its own I/O processor

— the memory subsystem, with a burst transfer memory controller

The I/O subsystem typically has its own local bus, with standard logic forming the bridge between the CLIPPER bus and the I/O system bus. This standard logic permits drivers executing on the I/O processor to communicate with the main OS running on the CLIPPER module.

The memory subsystem interface is designed for high-speed burst transfers, since the CLIPPER module requests 16 bytes at a time with one starting address (to fill one line of the CAMMU cache). This burst mode transfer mechanism permits the CLIPPER bus to transfer data roughly 50% faster than other buses at the same clock Rate. In combination with the other architectural features, this mechanism makes possible the construction of systems with unparalleled performance.

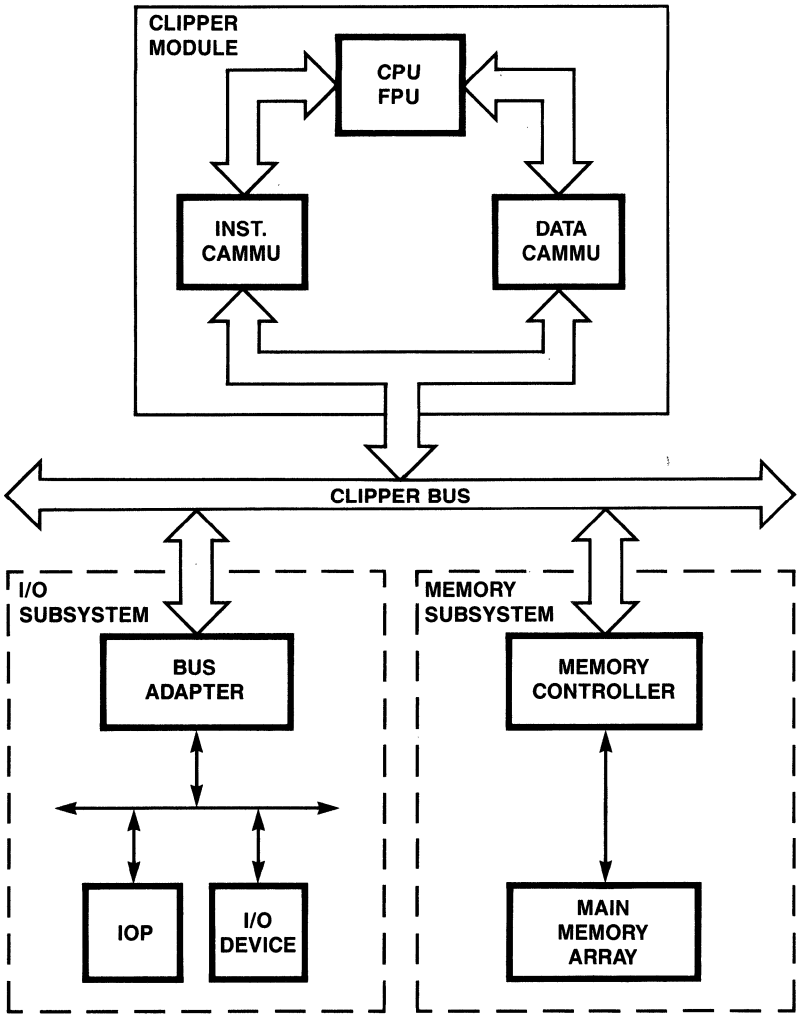


Figure 1-6 CLIPPER Module

CHAPTER 2
INTERNAL ARCHITECTURE

The CLIPPER's exceptional performance is primarily due to its advanced internal architecture. This architecture is comprised of a number of features, which for explanatory purposes we will group into two categories:

1. caching features:
   a. the dual 4K byte caches (instructions and data)
   b. the quadword buffers
   c. multiple caching strategies
   d. the bus watch mechanism for insuring cache consistency

2. pipelining features:
   a. the three phase pipeline itself
   b. the dual execution units (integer and floating-point)
   c. the three stage integer execution unit
   d. the resource management mechanism for avoiding resource contention

The overall result of the caching features is to reduce the effective memory access time by a factor of five, thus drastically increasing the speed of instruction fetches and data reads and writes. The result of the pipelining features is to greatly improve the number of instructions that can be executed in a given period of time, by overlapping the suboperations of successive instructions.

Those features of the CLIPPER instruction-set architecture that also contribute to performance (especially the instruction set itself, the registers, and the memory management system) are discussed in subsequent chapters.

2.1 CACHING

Caching is a technique pioneered by mainframe computer designers for matching high-speed CPUs with lower speed, lower cost memory systems. The technique basically employs a small, high-speed buffer that is positioned between the CPU and the main memory. Frequently used data and instructions are loaded into this buffer where they can be accessed quickly; only when new items are fetched into the buffer does main memory have to be accessed. Thus the **effective memory access time** is greatly reduced. The name *cache* refers to the fact that the buffer mechanism is essentially hidden from the user, who is aware only of an apparently higher speed main memory.

Since the CLIPPER Clock runs at 33 MHz, the CLIPPER CPU is capable of initiating a memory access every 30 ns. It is of course possible to build a memory system using high-speed static RAMs (SRAMs), but most microprocessor systems use slower speed and much less expensive dynamic RAMs (DRAMs). The typical access time for currently available DRAMs is around 120 ns, with a conservative memory bus design bringing the total memory access time to around 500 ns. The CLIPPER uses a cache memory to bridge the gap between its 30 ns CPU and the 500 ns main memory. Caches are thus part of the **memory hierarchy** of the CLIPPER. (See Figure 2-1.)

At one end of the memory hierarchy are the CLIPPER's registers, which have an access time of one clock cycle, or 30 ns. At the opposite end of the hierarchy is the mass storage system — usually a hard disk — with an access time of around 30 ms. Bridging the gap of six orders of magnitude (a factor of a million) between the registers and the disk is the main memory and caches. The CLIPPER actually employs two levels of caches: an 8K byte main cache and a very high-speed cache-within-the-cache called a **quadword buffer.**

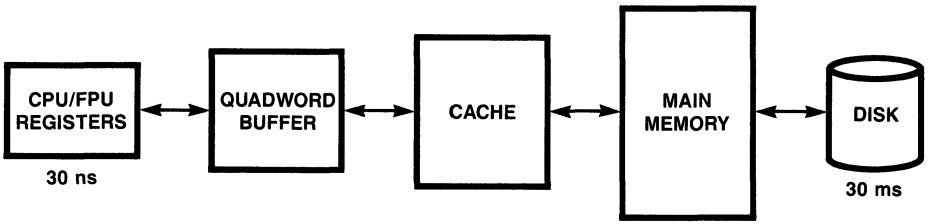


Figure 2-1 Memory Hierarchy

The effectiveness of a cache system depends on three factors:

1. cache access time
2. hit rate
3. replace time on a cache miss

The **cache access time** is the number of clock cycles required to access data or instructions contained in the cache. The **hit rate** refers to the frequency that the desired data or instruction is actually in the cache. If the data is found, the access is said to have produced a **cache hit;** but if the requested data is not in the cache, a **cache miss** has occurred. The hit rate is basically the percentage of cache hits to total memory accesses. Finally, the **replace time** is the number of clock cycles required to fetch an item from main memory into the cache when a cache miss occurs.

The **effective access time** is the average time to access data or instructions. It depends on the interaction of the three factors mentioned above and the character of the program itself. Most programs tend to re-access memory locations soon after they are accessed for the first time, or else to access nearby locations. This behavior is called *locality of reference.* The more locality a program exhibits, the more effective caching is at reducing effective access time. For typical programs, the CLIPPER caches reduce the effective access time from 500 ns, to around 100 ns, a 500% improvement in performance.

2.1.1 Cache Access Time

The CLIPPER's cache memory actually consists of two 4K byte caches: the instruction cache and the data cache, each located on a CAMMU. Each CAMMU also contains a 16 byte (quadword) line buffer. The data in the caches is organized as 256 **lines**, with each line containing 16 bytes. Whenever an access is made to the cache, the entire line containing the accessed item is loaded into the quadword buffer. Subsequent accesses to the same line will not require that the cache be accessed; instead, the request will be satisfied from the quadword buffer.

The quadword buffer can be accessed in one clock cycle (30 ns). If the quadword buffer misses, the cache can be accessed in an additional two clock cycles, for an access time of 3 clocks or 90 ns. The time to transfer the address over the bus to the CAMMU is 15 ns. (The CPU and CAMMU are designed to use one half of a clock cycle for this transfer.) Similarly, it takes only 15 ns to transfer the data back to the CPU. Thus the total access time for the quadword buffer is 60 ns and for the main cache, 120 ns. Figure 2-2 illustrates the CLIPPER cache access times.

2.1.2 Hit Rate

The hit rate is a function of four factors:

1. the size of the cache in bytes
2. the number of bytes in a line
3. the degree of **set associativity**
4. the program's locality

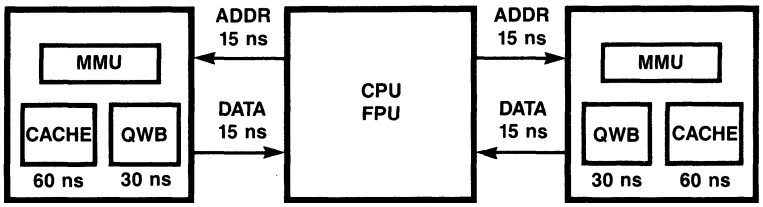


Figure 2-2 CLIPPER Cache Access Time

Since computer architects have no control over program locality, we will not discuss this factor further, except to observe that modern structured programming techniques are very beneficial in this regard. The other factors require more explanation.

Other things being the same, a larger cache is better than a smaller cache; but other things are seldom the same. Caches are also characterized by their line size and set associativity.

The **line size** is the number of bytes that are fetched into the cache when main memory is accessed. The more bytes that are fetched, the fewer main memory accesses have to be made, if programs have reasonable locality. In general, increasing the line size improves the effective memory access time. The CLIPPER uses a 16 byte line size, which means that 16 bytes are updated when a cache miss occurs, and 16 bytes are transferred whenever main memory is accessed. Each cache contains 256 lines.

If the cache is full and a cache miss occurs, the new line will have to overwrite a line of data already in the cache. The **set associativity** of the cache determines how much flexibility the cache has in determining which line to overwrite. The simplest kind of cache is a *direct mapped* cache, in which each main memory location can be written into only one cache location. A *two-way set associative* cache can write each memory location into two cache locations. Similarly, four cache locations are available in four way set associativity, and so on. The higher the degree of associativity, the more flexible the cache and the less likely useful data is overwritten. Figure 2-3 illustrates the CLIPPER's two-way set associativity.

In the CLIPPER cache, each set contains 2 lines of 16 bytes, and there are 128 sets. The collection of all the left hand lines is called the **W compartment**, the collection of right hand lines is called the **X compartment**. There are a total of 2K bytes in each compartment. Physical memory is divided into 4K byte pages, and within each page, a location is mapped into a unique location in both compartments. Consequently, several main memory locations are mapped into the same pair of cache locations. The first time main memory is accessed, 16 bytes are fetched and stored in a line of the W compartment. If a new main memory access would require that that line be overwritten, the cache will instead employ the corresponding line in the X compartment. If the corresponding lines of both compartments are full, the cache will overwrite the one that was least recently used.

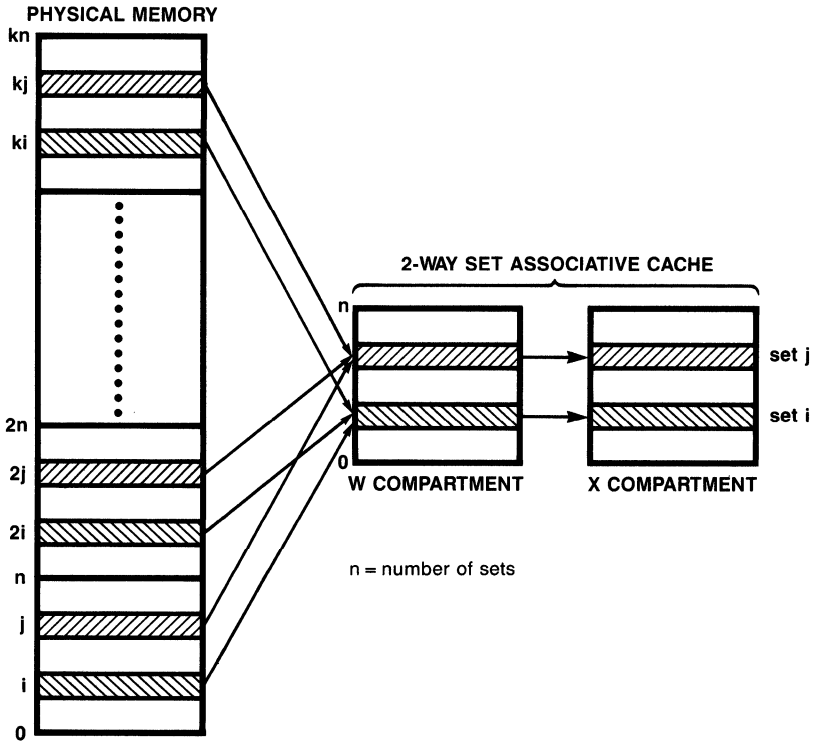


Figure 2-3 Set Associativity

Thus we can see that the efficiency of a cache depends on both the line size and set associativity, and not just the size of the cache. Figure 2-4 shows how hit rate varies with cache size, line size and associativity. Notice that the CLIPPER's 8K byte total cache size along with its 16 byte line size and two-way set associativity yield a hit rate that is about the same as a 128K byte, 4 byte line, direct mapped cache. In fact, the caches have a hit rate of greater than 90%. And when prefetch is enabled, the instruction cache can exceed a 96% hit rate.

Prefetching is a mechanism available on the instruction CAMMU for bringing into the cache the next 16 bytes of memory *before* they are accessed. The CAMMU maintains a copy of the Instruction Pointer register and uses it to fetch instructions before the CPU requests them. For in-line code sequences, the prefetch mechanism can insure a 100% hit rate, since prefetch is performed concurrently with other CPU and CAMMU operations. (See Section 2.2, Pipelining, for a discussion of prefetch.)

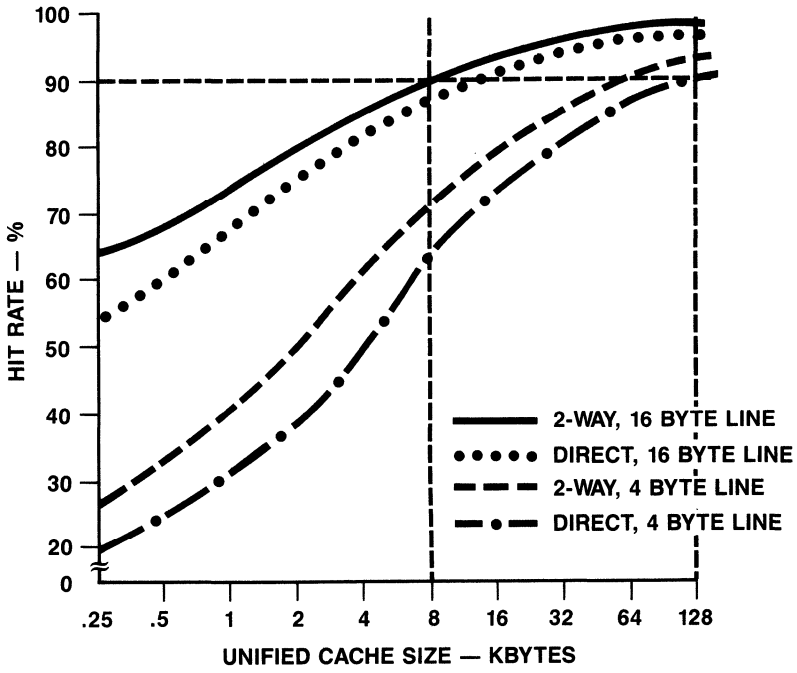


Figure 2-4 Cache Hit Rate

2.1.3 Replacement Time

Inevitably, a cache miss will eventually occur. The miss **replacement time** is the number of clock cycles that occur from the time the miss is detected by the CAMMU until the last byte of the line is loaded into the cache. Leaving aside the speed of memory, this time is determined by the bus clock rate, the line length, and the number of clock cycles required to transfer a line over the memory bus.

The CLIPPER's memory bus runs at a clock speed of 16.7 MHz (i.e., one bus cycle is equal to two CPU cycles). Since the line length is 16 bytes, the replacement time depends on how many clock cycles it takes to transfer 16 bytes over the bus.

Now, conventional 32-bit microprocessors transfer one word (32 bits or 4 bytes) of data over the bus with each memory access, and the slow speed of DRAMs introduces at least one wait state for each access. So these conventional machines require at least 12 clock cycles to transfer 16 bytes (see the lower half of Figure 2-5).

By contrast, the CLIPPER employs a special supercomputer-derived technique called **burst mode** to transfer all four words (16 bytes) in a single memory access. Since only one access is made, only one address is transferred to memory (the starting address of the line of 16 bytes) and only one wait state is needed. The next four clock cycles transfer the four words of the line. Thus burst mode transfers are twice as fast as conventional 4-word transfers. The top part of Figure 2-5 illustrates burst mode timing.

The concepts of hit rate and miss replacement are relatively straightforward for read accesses: when a cache hit occurs, the data is used, when a miss occurs, an access is made to main memory. Write accesses

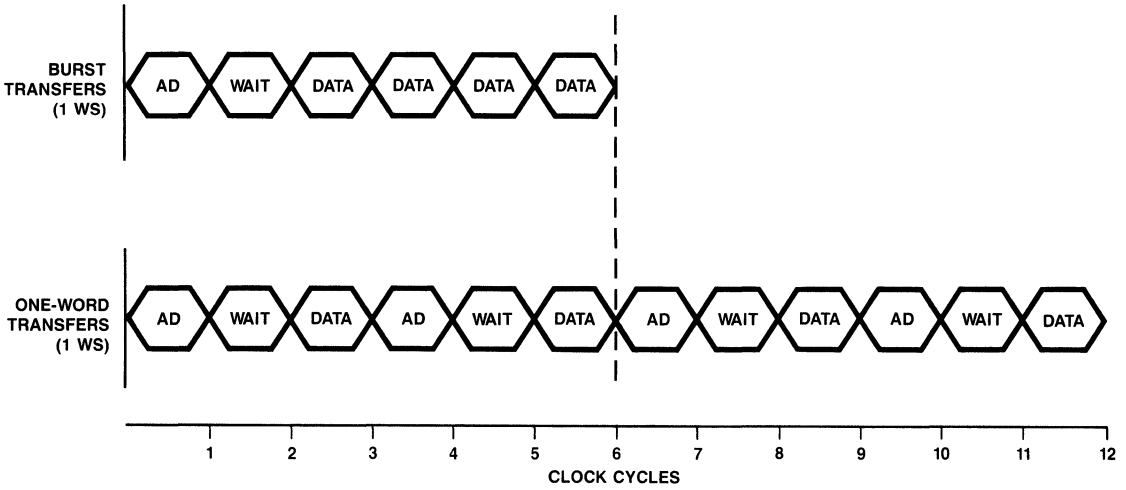


Figure 2-5 Cache Replacement Time

are more complicated, because a strategy must be developed for when main memory is to be updated. There are two main cache write strategies, and the CLIPPER supports them both:

— write-through
— copy-back

The user may designate any page in memory as cacheable or noncacheable, and if cacheable, select one of the two caching strategies for that page. In addition, pages may be given protection attributes, such as read-only (see Chapter 4).

The simplest write strategy is called **write-through**. Under this scheme, main memory is updated every time the cache is altered. Since the cache and main memory are changed at the same time, they always contain the same data and main memory is always "up to date". The penalty is that little or no benefit is derived from the cache for writes, and thus the cache is only about half as effective as it might be. The designers of the CLIPPER felt that an exclusive reliance on write-through seriously limits performance.

Under the **copy-back** strategy memory is updated only when a line that has been modified in the cache needs to be overwritten by another line. At that point, the old modified line is "copied back" to memory before it is overwritten. The copy-back strategy minimizes memory accesses and thus provides the highest performance, but this strategy requires more hardware support. The designers of the CLIPPER built the needed hardware support mechanisms into the CAMMU, so the user can use copy-back mode whenever high performance is required. Figure 2-6 shows the difference between write-through and copy-back.

The **bus watch** mechanism in the CAMMU insures data consistency between cache and main memory, so the copy-back strategy can be successfully employed. The basic problem that bus watch solves is that of getting old, out-of-date data (also called **stale** data), which can occur when multiple bus masters access memory independently. For example, the I/O subsystem communicates with memory much like a conventional DMA controller. If this device reads a region of memory that is also cached on the CAMMU using a copy-back strategy, the bus adapter will get stale data if the memory is not up to date. Similarly, if the I/O subsystem writes to a cached region, the CPU will then be reading stale data in the cache.

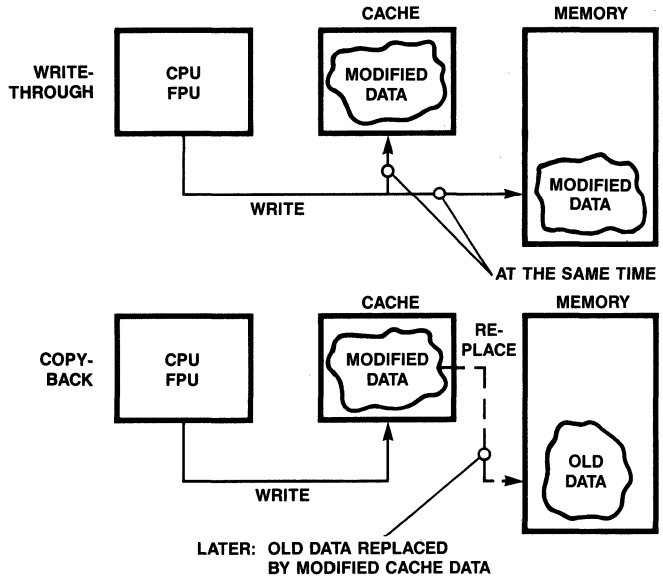


Figure 2-6 Cache Writing Strategies

Bus watch eliminates this problem by automatically insuring consistency. The bus watch mechanism fulfills read requests by other bus masters from the cache instead of memory, if there is any difference between the two. It also insures that writes by other bus masters update the cache as well as main memory. Bus watch is controlled by setting bits in the CAMMU control register (see Chapter 3, Programming Model). Figure 2-7 shows an example of how bus watch works.

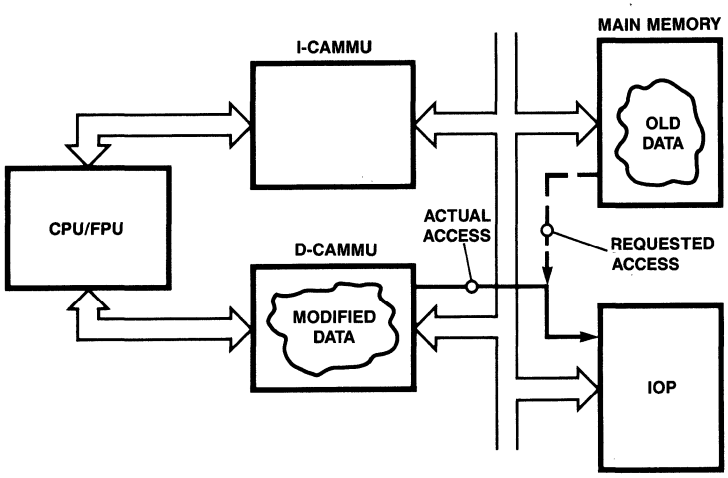


Figure 2-7 Bus Watch

2.1.4 CLIPPER Cache Implementation

The CLIPPER Cache is closely tied to the MMU and its TLB, which are described in Chapter 4, Memory Management. The CPU generates 32-bit virtual addresses, which are translated by the MMU/TLB into real addresses. The cache mechanism then compares these real addresses with addresses stored internally, and if a match is found, the word associated with the internal address is returned to the CPU.

In more detail, the following process occurs every time the CPU generates a virtual address:

1. If it is an instruction access, the virtual address is sent to the instruction CAMMU; a data address is sent to the data CAMMU.

2. The virtual address is compared with the value stored in a buffer called the **virtual address cache**. This buffer contains the *last* virtual address sent to the cache. (The quadword buffer contains the last line accessed, one of whose bytes was pointed to by the last virtual address.)

3. If the two values match, we have a *quadword buffer hit*, and bits 2-3 of the virtual address are used to select one of the four data words in the quadword buffer to be returned to the CPU. (Actually, each CAMMU contains *two* quadword buffers, one for each compartment, but the most recently accessed one is used.)

4. If the two values do not match, the quadword buffer misses, and the CAMMU proceeds concurrently to translate the virtual address and access the cache.

5. The MMU/TLB translates bits 12-31 of the virtual address into a 20-bit real value. Bit 11 of the virtual address is appended, yielding a 21-bit value called the **real address tag**. Meanwhile, bits 4-10 of the virtual address have selected one of the 128 sets in the cache.

6. The 21-bit real address tag from the TLB is compared with the real address tag field of both lines in the selected set of the cache.

7. If the two values match for one of the two lines, we have a *cache hit*, and again bits 2-3 of the virtual address are used to select one of the four data words to be returned to the CPU. The line itself is loaded into the quadword buffer.

8. If a cache miss occurs, the cache control mechanism causes a main memory access to be generated, by appending virtual address bits 0-10 to the 21-bit real address tag to produce a 32-bit real address.

9. The main memory access causes a new line to be loaded into the cache and the quadword buffer, and steps 6 and 7 are performed again.

Figure 2-8 shows a schematic representation of this process.

Figure 2-9 shows the layout of the CLIPPER cache, along with the fields in each line. The two flag bits, LV (Line Valid) and LD (Line Dirty) are used by the bus watch mechanism to insure consistency between the cache and main memory.

2.2 PIPELINING

Pipelining is a technique extensively used in supercomputers to improve performance by overlapping operations that can be performed concurrently. Most 32-bit microprocessors make use of a simple form of pipelining. By contrast, the CLIPPER pipeline system is very sophisticated, making full use of supercomputer technology.

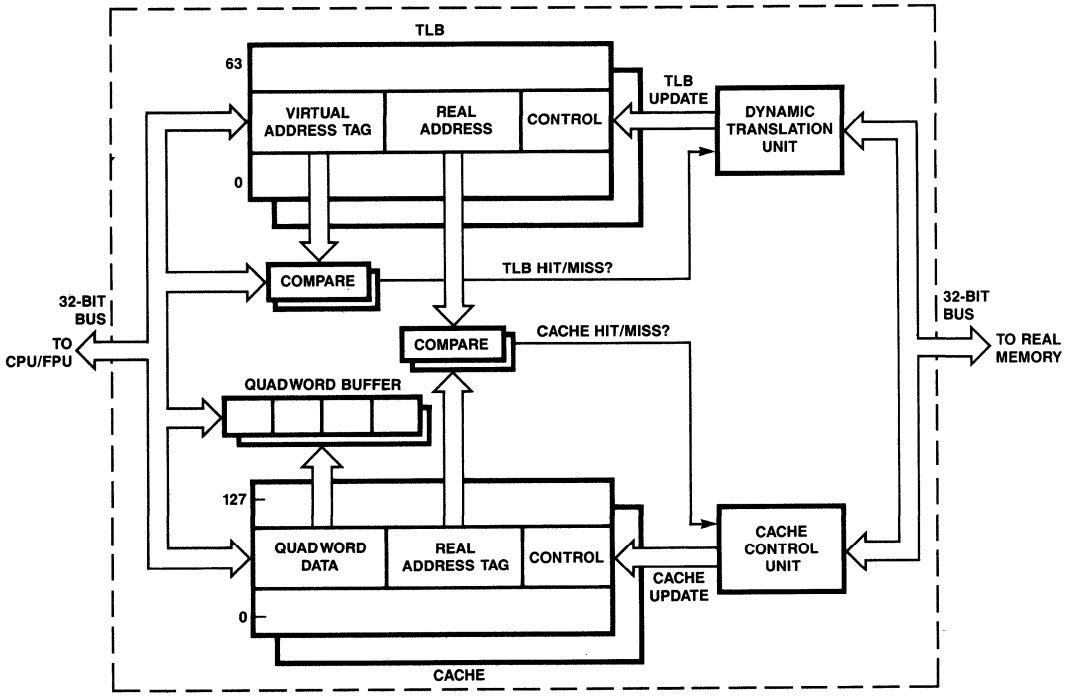


Figure 2-8 CLIPPER Cache Access Mechanism

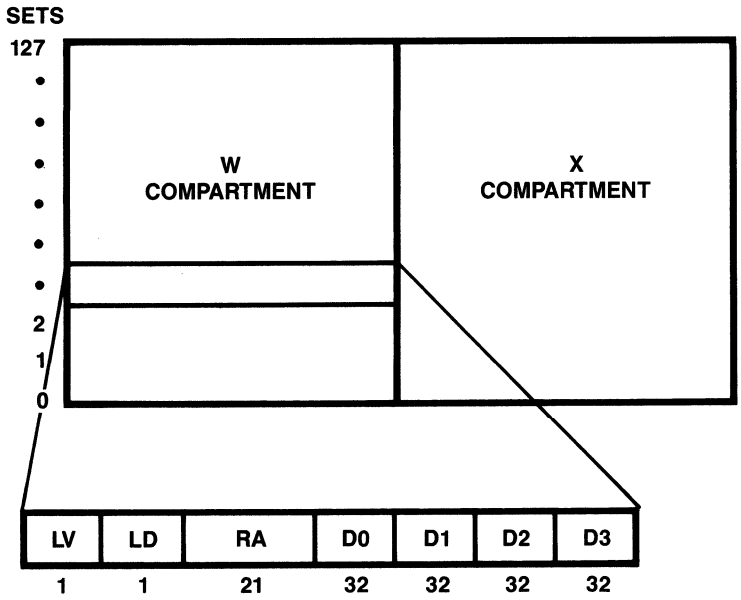


Figure 2-9 CLIPPER Cache

Figure 2-10 provides an illustration of the performance benefits of even a simple form of pipelining. In non-pipelined systems, each instruction follows sequentially after the previous, and all phases of an instruction (fetch, decode, execute) must complete before any phase of the next instruction can start. By contrast, in a pipelined system, each phase proceeds independently, so the fetch phase of the second instruction can begin as soon as the fetch phase of the first instruction completes, without having to wait for the first instruction to finish its decode or execute phases.

The CLIPPER combines the benefits of this simple three-phase pipeline with extensive use of concurrency and parallelism *within* each phase of the pipe, especially the execute phase.

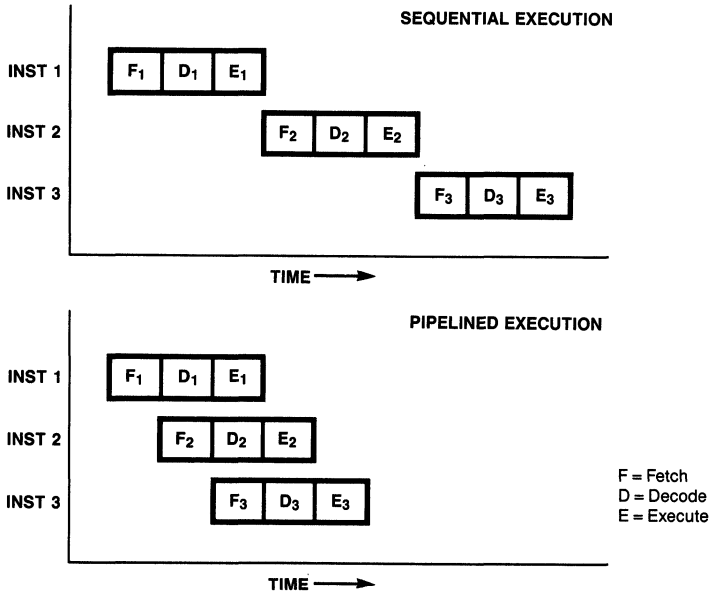


Figure 2-10 Pipelining

2.2.1 Three-Phase Pipeline

The CLIPPER pipeline has three phases:

fetch — Instructions are fetched from the instruction cache to the CPU's instruction buffer (which can hold two words, or up to four instructions), or instructions can be fetched from the Macro Instruction Unit as a result of some previous macro instruction. In addition, the instruction cache can prefetch instructions ahead of actual demand.

decode — Instructions from either the instruction buffer or the Macro Instruction Unit are decoded into requests for computational resources. The resource manager then allocates the required resources by consulting a table of busy resources.

execute — The instructions are issued for execution to the three-stage integer execution unit or the floating-point execution unit, each with its own set of registers and its own ALU. Up to four instructions can be executing at the same time — three integer instructions and one floating-point calculation.

Figure 2-11 shows the three phases of the CLIPPER pipeline. Since the CLIPPER arithmetic and logical instructions operate only on registers, there is no need for a special "address pipeline" or separate effective address calculating phases. Those instructions that require an effective address calculation simply make one pass through the integer execution unit's ALU, since these instructions will not require the ALU for arithmetic or logical operations.

Notice that the figure shows a cache prefetch subphase in the fetch phase. This item refers to the instruction CAMMU's prefetch mechanism. The I-CAMMU maintains a copy of the Instruction Pointer and fetches lines of code into the cache ahead of the fetch from the CPU. This prefetch helps the CPU to keep the instruction buffer full and improves the cache hit rate for the I-CAMMU.

The resource manager keeps a table of all resources and which instruction is using each one; it also maintains the status of all instructions that are currently executing. Because of this detailed tracking, the CLIPPER can restart instructions that cause page faults or continue instructions after interrupts and traps. Programs, interrupts, and traps cannot crash the pipeline.

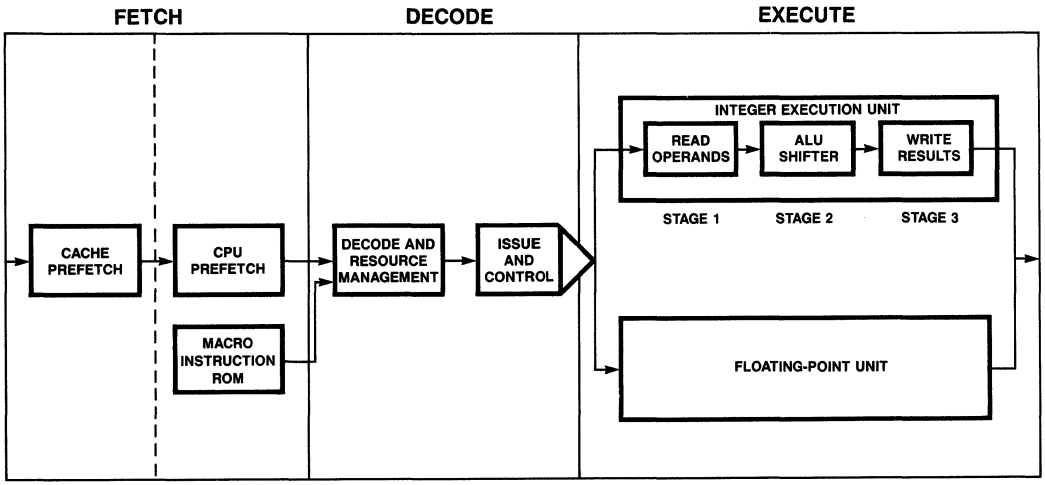


Figure 2-11 CLIPPER Pipeline

2.2.2 Dual Execution Units

The CPU contains two execution units that comprise the final phase of the pipeline. The integer execution unit handles all integer arithmetic operations plus all address calculations; the floating-point execution unit handles floating-point arithmetic.

The integer execution unit is itself subdivided into three stages, each of which can proceed concurrently and can be overlapped for successive instructions. The three are called the L stage, the A stage, and the O stage. Figure 2-12 shows a detailed block diagram of the CPU, which can be useful in tracing what happens in each stage.

L stage — In this first stage, operands are read from the general register file into the L registers. Immediate operands are taken directly from the instruction buffer to the L registers, via the J register.

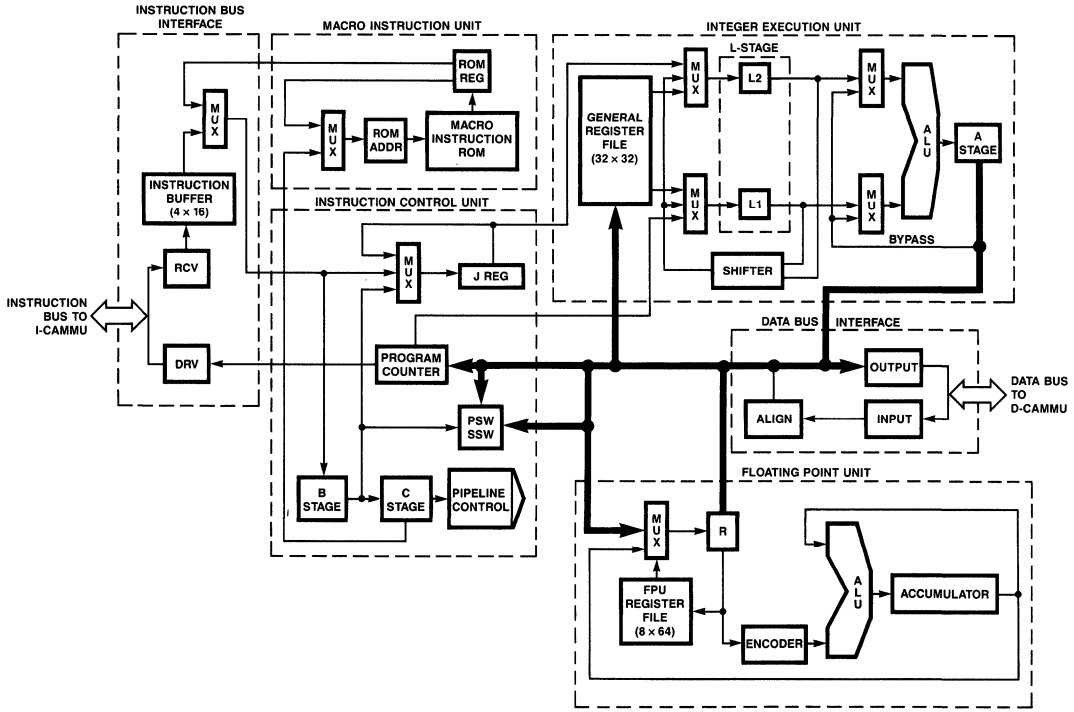


Figure 2-12 CPU/ FPU Block Diagram

A stage — In this second stage, arithmetic, logical, or shift operations are performed on the operands in the L registers or on intermediate results from the last operation, and the output is stored in the A register.

O stage — In this final stage, the contents of the A register are sent to the FPU, stored in the general register file, fed back via the bypass loop to the A stage (intermediate results), or sent out to the data CAMMU.

The bypass loop makes it possible for the intermediate results of multi-instruction calculations to be fed back immediately to the next instruction. Without this loop, the whole pipeline would have to be flushed.

The instructions from the Macro Instruction Unit make use of their own register files, so they do not require general register resources being used by ordinary instructions. In addition, they can use all the other registers on the CPU.

The Floating-Point Unit performs single- and double-precision floating-point operations concurrently with the integer execution unit, using its own ALU and set of eight 64-bit registers. Because the FPU is on the CPU chip, floating-point operations do not require any additional bus transfers; thus bus traffic is reduced and performance is improved. All CLIPPER floating-point arithmetic operations conform to the IEEE 754 standard.

CHAPTER 3
PROGRAMMING MODEL

A primary design objective of the CLIPPER was to provide an instruction set architecture that was well suited for operating systems and compilers. In this context, "well suited" means that the architecture facilitates two somewhat divergent goals:

— The instructions should execute rapidly, preferably one per clock cycle.

— It should be easy to write programs using the instructions, especially operating systems, and easy for compiler code generators to target them.

The first goal pushes the designer toward a very simple instruction set, with few higher level concepts supported in the hardware. This approach is often called the Reduced Instruction Set Computer (RISC) philosophy. The second goal pushes the designer in quite a different direction — toward a richer instruction set with direct support for concepts from high level languages and operating systems. This latter approach has several names — language directed architectures, high-level architectures, etc., depending on the concepts supported — but we will refer to them all as the Complex Instruction Set Computer (CISC) philosophy.

The CLIPPER instruction set is a balance between the RISC and CISC approaches. This balance is also a characteristic of supercomputer architectures, which also contain both RISC and CISC features. In more detail, the CLIPPER instruction set architecture derives the following basic features from the RISC and CISC approaches:

RISC

Arithmetic and logical instructions operate on registers; basically only loads, stores, and branches, along with stack manipulation instructions access memory.

There are plenty of registers: thirty-two 32-bit registers, sixteen for the operating system and sixteen for the user.

The instruction format is designed to simplify decoding by making all instructions simple multiples of one basic size (16 bits), and the most frequently used instructions are shortest.

CISC

Instructions that do access memory are provided with a complete set of addressing modes that facilitate access to high-level data structures.

Explicit support for stacks and strings is provided.

Separate modes of operation are provided for the user and the operating system, each mode having its own resources and privileges.

Explicit support is provided for key operating system functions, such as system calls, exception handling, and virtual memory.

In this chapter, we will discuss the instruction set architecture of the CLIPPER in four sections: 1) registers, 2) data types and instructions, 3) addressing, and 4) exception handling. Chapter 4 will cover the whole topic of memory management, including address translation and virtual memory support.

3.1 REGISTERS

The principal reason that motivated minicomputer manufacturers to move from 16-bit machines (such as the PDP-11) to 32-bit machines (such as the VAX) was the restricted address space of 16-bit processors. The address space is the number of separate memory locations that can be addressed by the processor, a value that is determined by the width of the Program Counter (PC) register. Sixteen bit machines usually have a 16-bit PC, which means they can address only 2^{16} (64K) locations, so the address space is limited to 64K. By contrast, 32-bit machines have a 32-bit PC and so can address 2^{32} (more than 4 billion) locations. This same motivation is primarily responsible for the evolution of microprocessors from 8- to 16- and now to 32-bit architectures.

In addition to the PC, most architectures provide several other registers that are used as fast local storage for addresses and data. The width of these registers is determined by the size of the typical unit of data manipulated by the processor (called the **word size**). Since addresses are stored as well as data, and since the biggest address is determined by the PC size, it is usually convenient for the word size to be the same as the PC size and for storage registers to have the same size as the PC. Thus, most 16-bit machines (such as the PDP-11) have 16-bit storage registers as well as a 16-bit PC, and 32-bit processors have their PC and storage registers 32-bits wide.

Originally, only a single such storage register was provided. Called the Accumulator, its principle legacy is in the frequent use of the designation "A" or "AX" for a register. The A once stood for Accumulator. But it quickly became apparent that more than one accumulator was needed, so other registers were added. In the days of the PDP-II, it was thought that 8 registers were sufficient, because procedures seldom had more than half a dozen local variables, and it was assumed that all variables would be saved when a new procedure was called. But the development of globally optimizing compilers has invalidated that assumption, since modern compilers routinely keep as many variables permanently in registers as they can.

The CLIPPER architecture has a 32-bit PC and thirty-two 32-bit storage registers, more than any other commercial microprocessor. In addition, the CLIPPER contains eight 64-bit storage registers that are dedicated to floating-point arithmetic. The 32-bit registers are completely general purpose; each one can be used to store either an address or a word of data. The use of general-purpose registers, instead of special-purpose data and address registers, is preferred because it eliminates unnecessary register-to-register transfers when address arithmetic must be performed.

The CLIPPER has two **operating modes**, user mode and supervisor mode, distinguished by the instructions they are permitted to execute and the registers they can use. A program executing in supervisor mode (usually the operating system) can access data in all 32 general-purpose registers and all eight floating-point registers. User mode programs can only access 16 of the general-purpose registers (called the **user registers**) and the floating-point registers; the 16 registers that are inaccessible to user programs are called the **supervisor registers**.

Figure 3-1 shows a diagram of all the registers in the CLIPPER Module, including both the CPU/FPU registers and the CAMMU registers.

Besides the PC, general-purpose registers, and floating point registers, the CLIPPER CPU/FPU has two 32-bit status words (the PSW and the SSW). The status words contain bits, called **flags**, which identify and control the CPU's state. The PSW, which is accessible in either mode, primarily contains flags which identify exception conditions (see below "Exception Handling"). The SSW, which is accessible only in supervisor mode, contains bits which control interrupts, address translation and protection, and mode of operation.

Each CAMMU contains five software-accessible registers, which are used for initialization and control. Two of these registers (Supervisor PDO and User PDO) are used in address translation; they contain the base addresses of the supervisor and user Page Table Directories (see Chapter 4 for details). The Fault register is

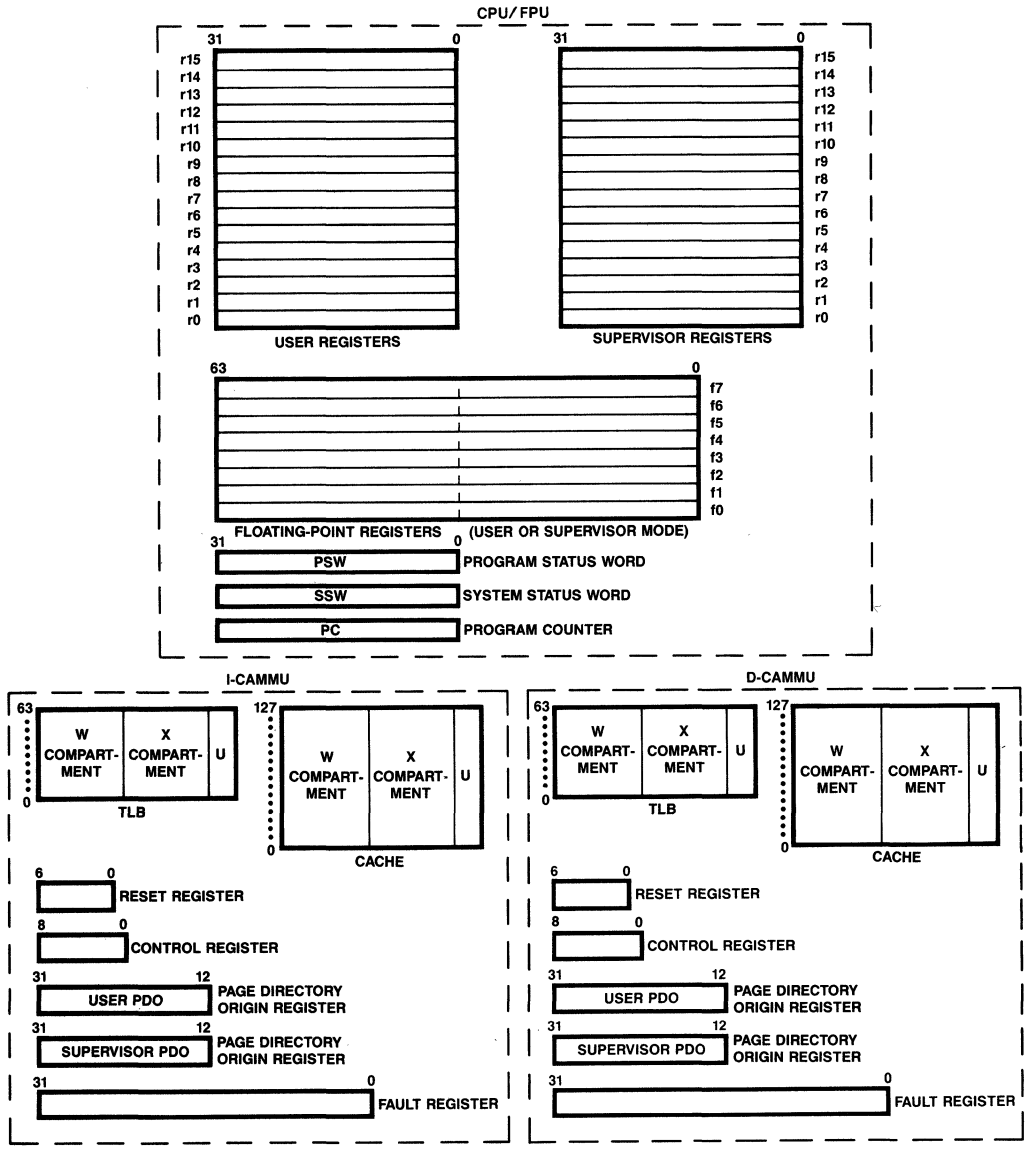


Figure 3-1 CLIPPER Registers

loaded with the virtual address associated with certain fault conditions; it can be used by the operating system to support virtual memory (again, see Chapter 4). The Control register and Reset register are used to control the CAMMU.

3.2 DATA TYPES AND INSTRUCTIONS

Many applications that require a high-performance computer are calculation-intensive, "number crunching" problems. In these applications 32-bit integers are often used because variables of this length provide more precision than 16-bit integers. (A 32-bit integer contains the equivalent of ten decimal digits.) A key requirement for high-performance microprocessors is therefore the ability to manipulate 32-bit integers; that is, they need operators such as Add and Multiply that directly (in one instruction) handle 32 bits. In the terminology of the previous section, we say that high-performance microprocessors need a *word length* of 32 bits; another way to say this is that these microprocessors need to support 32-bit **data types**.

All information in a computer's memory is stored as a pattern of ones and zeros. What these bits represent depends on the interpretation given to them by the computer. The data types of a computer architecture are the basic data representations that are recognized by the hardware. Each instance of a type (e.g., each integer) can be addressed *as a unit* in memory, no matter how many bytes it contains. And associated with each type is a set of operations that can be performed on it. The **instruction set** of a computer is basically just the combination of all the hardware supported operations with all the data types that can act as operands.

The CLIPPER supports ten data types (shown in Figure 3-2). Signed and unsigned bytes, halfwords (16 bits), words (32 bits), and longwords (64 bits) are all available, along with single-precision (32-bit) and double-precision (64-bit) IEEE Standard floating-point numbers. These primitive data types can be used to build complex structured data types, such as arrays and records, that are typically found in high-level languages. CLIPPER provides several addressing modes (see below, "Addressing") that facilitate accessing structured data types.

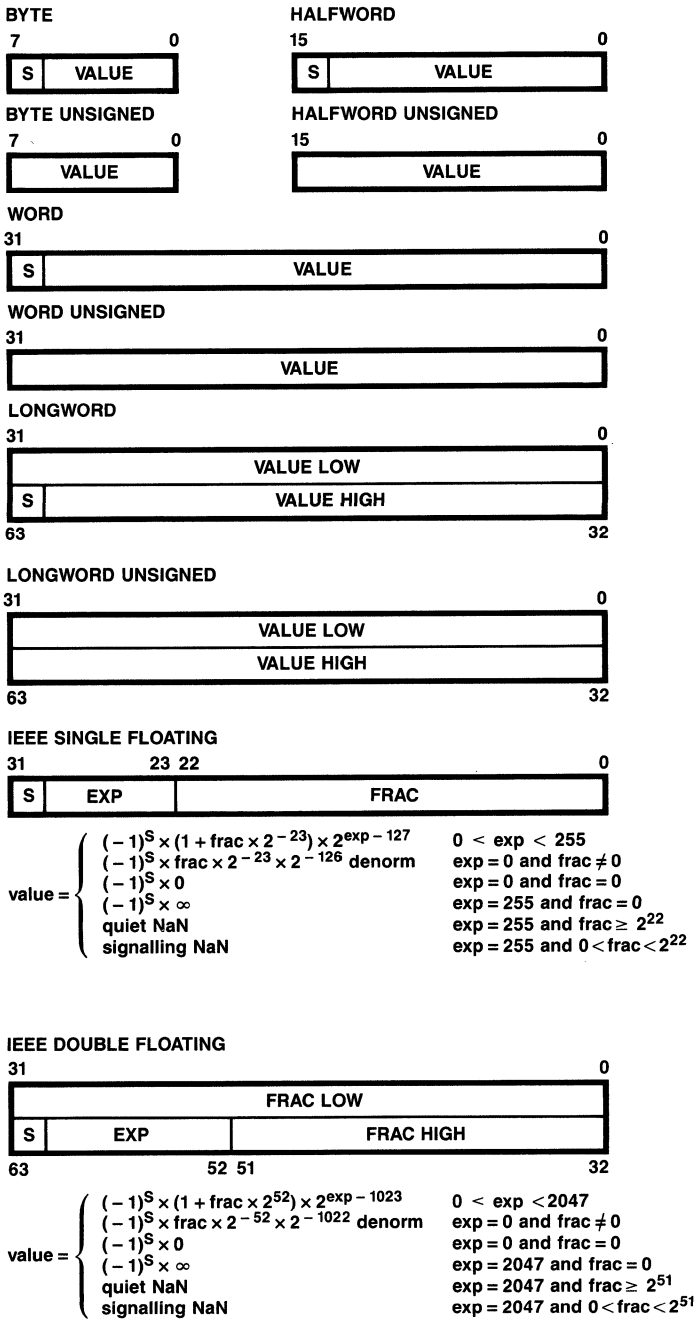


Figure 3-2 CLIPPER Primitive Data Types

The CLIPPER instruction set contains 101 hardwired instructions and 67 macro instructions for operating on the basic data types. Each instruction specifies an operation to be performed, and the type and location of the operands. These operands can be located either in memory, in a register, or in the instruction itself. To facilitate rapid instruction decoding, all instructions are built of halfword units called **parcels**. Depending on the instruction, from one to four parcels may be used. Figure 3-3 shows the formats of all CLIPPER instructions.

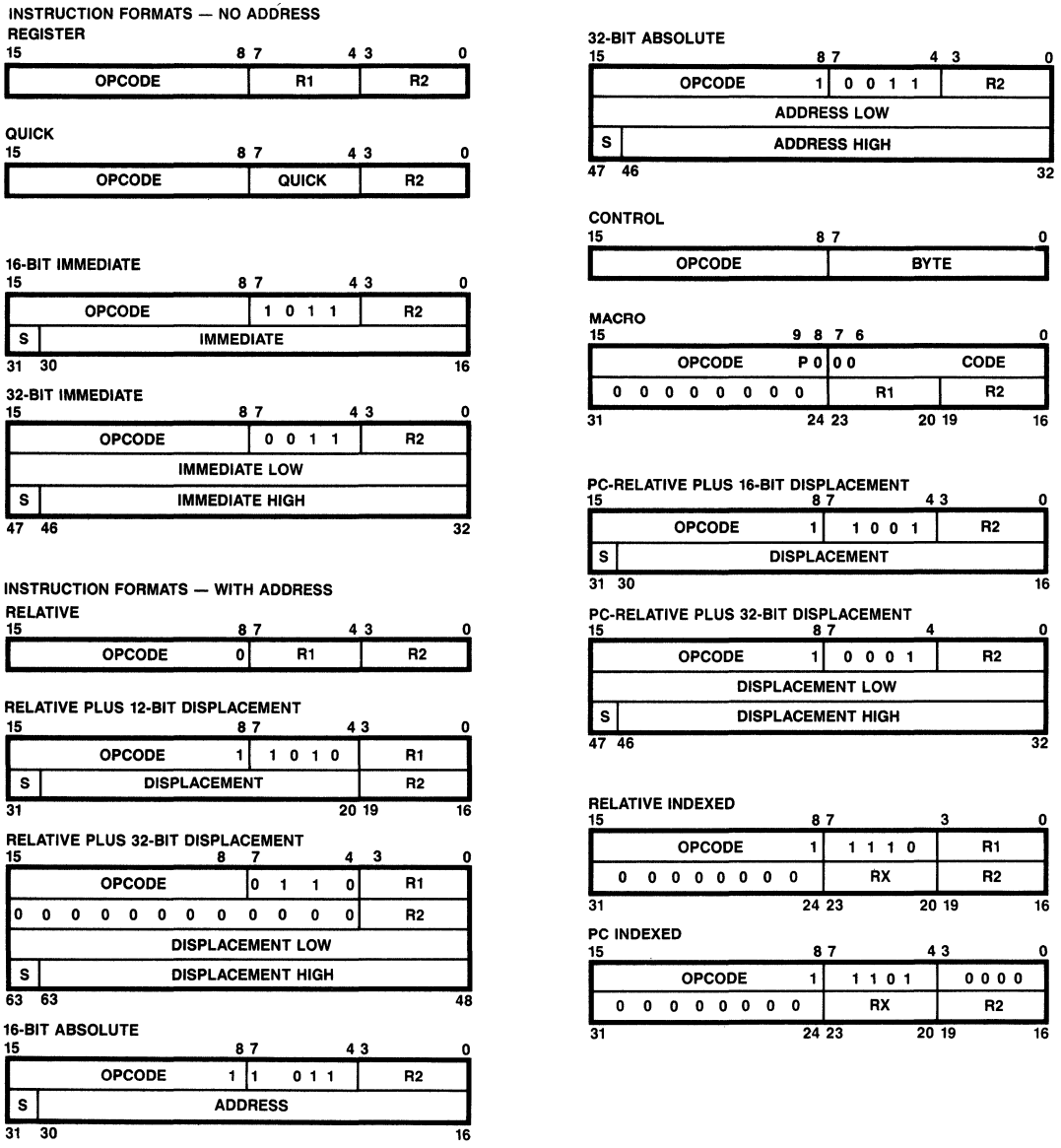


Figure 3-3 Instruction Formats

Notice that the instruction formats fall into two groups: those with addresses and those without. The former are the instructions that need to access memory (primarily loads, stores, and branches); the latter are the arithmetic/logical instructions that can generally execute in a single clock cycle. CLIPPER instructions have zero, one, or two operands, but only one operand can be accessed by a memory address.

The CLIPPER instruction set consists of ten categories of instructions:

load/store	These instructions transfer addresses, bytes, halfwords, words, longwords, and floating-point quantities between memory and registers.
move	These instructions transfer 32- and 64-bit quantities between registers.
arithmetic	Add, Subtract, Multiply, Divide, Negate, Modulus, and Scale (multiply by a power of 2) are supported for registers or variable-length immediate values.
logical	And, Or, Xor, and Not are supported for registers.
shift/rotate	Arithmetic and logical shifts (the difference lies in their treatment of right shifts for signed quantities) of both words and longwords are supported.
conversion	Floating-point numbers of both precision can be converted to integers using the IEEE rounding modes.
compare	The values of words and both precisions of floating-point numbers can be compared; an atomic test-and-set instruction is also available.
string	These instructions manipulate character strings; compare, initialize, and move are provided.
stack	These instructions manipulate the stack; push and pop, along with save multiple registers and restore multiple registers are available.
control	Branches, call, call supervisor, returns, and NOP are provided.

The 67 macro instructions are implemented in the Macro Instruction Unit a sequences of hardwired istructions. Except for their distinctive format, nothing distinguishes the macro instructions from hardwired instructions as far as the programmer is concerned. The macro instructions are scattered over the ten categories; for example, all conversions and string instructions are macros, along with most stack instructions (except push and pop), and also one or two load/store, move, arithmetic, and control instructions. Six instructions (all macros) are **privileged**, that is, they can only be accessed by a program in supervisor mode.

3.3 ADDRESSING MODES

If an operand of an instruction is in memory, there are several alternative ways of accessing it, called **addressing modes**. An addressing mode is basically just a way of specifying a virtual address as the sum of several factors, which may be stored in registers or provided with the instruction itself. The CLIPPER supports nine memory addressing modes, which are shown diagrammatically in Figure 3-4.

With the relative mode and the two relative with displacement modes, the virtual address is either in a register or is computed from the sum of the contents of a register and a displacement value carried with the instruction. The two absolute modes carry the virtual address as a pure displacement value with the instruction. The two PC relative modes are useful for branches relative to the current value of the PC. The two indexed modes compute the virtual address by summing the contents of two registers.

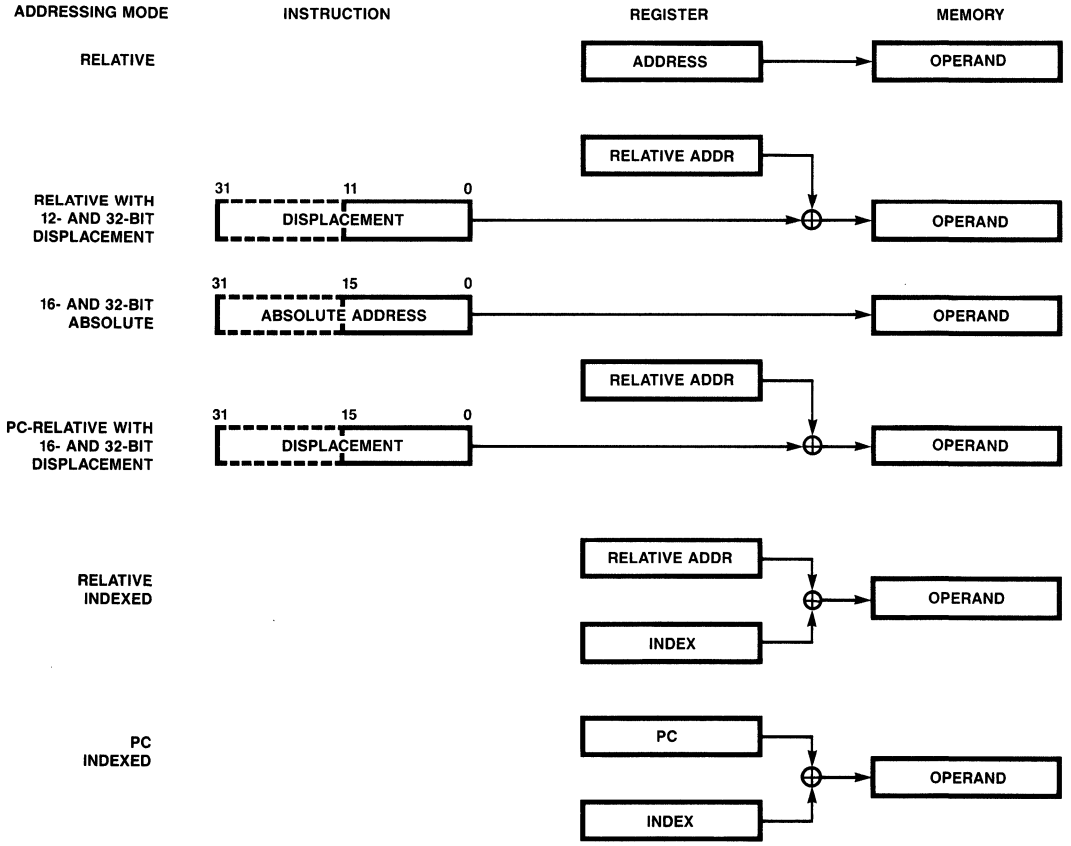


Figure 3-4 Addressing Modes

Use of these addressing modes facilitates access to **structured data types**, such as arrays and records, that are commonly found in high-level languages. Figure 3-5 shows how the relative plus displacement mode can be used to access an entry in an array, while relative indexed mode is useful for accessing a particular item in a two dimensional array.

In the first example, the displacement value points to the base of the array, while the value in the register defines the offset of the item of interest. After the register is incremented by a fixed amount, the same addressing mode will point to the next item in the array; thus a sequence of items from an array can be accessed quickly using a loop with one basic instruction whose addressing mode is fixed.

In the second example, the value in one register points to the first item in the selected row of the two dimensional array, while the second register's contents define the offset of the item within the row. Incrementing the first register will yield the same offset in a new row, while incrementing the second register will yield the next item in the same row. The instruction can stay the same as a whole set of two dimensional arrays is accessed simply by incrementing registers.

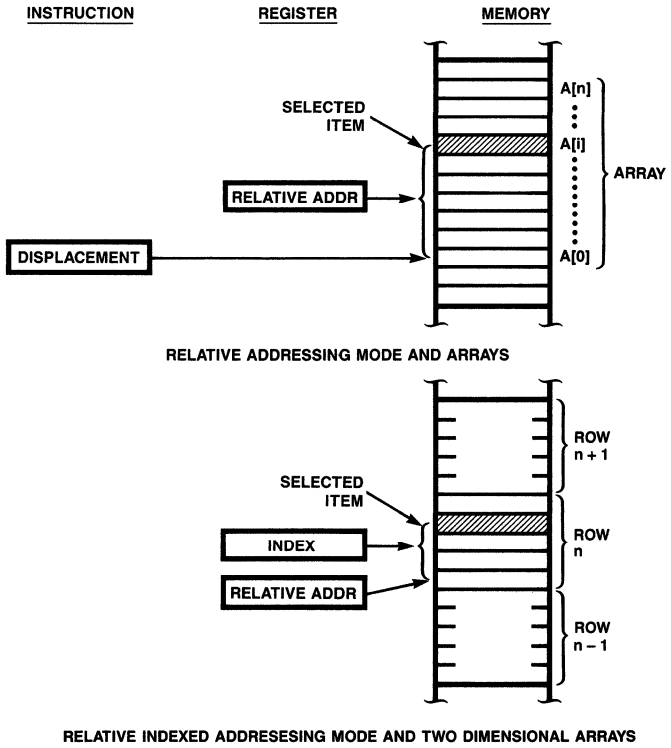


Figure 3-5 Addressing Modes and Structured Data Types

3.4 EXCEPTIONS

Exceptions are internal hardware conditions, external events, or even particular instructions, that cause the normal operation of the processor to be suspended and a special sequence of operations performed. There are basically three types of exceptions:

traps	Anomalous internal events occurring during the processing of an instruction. Classic examples include an attempt to divide by zero, or a page fault in a virtual memory system.
interrupts	An external device's method of signaling the CPU that it needs servicing. For example, a DMA controller signaling that it has transferred a block of data into memory.
supervisor calls	Program generated requests for operating system services.

If any one of these exceptions occurs, it is usually necessary to immediately invoke a software handler to respond to the exceptional condition. It is this need for immediate action that makes exceptions suspend normal processing — they simply can't wait. When the handler finishes its work, control returns to the point where it was interrupted.

Pipelining complicates exception handling, since the pipeline must be cleared out as soon as the exception occurs, so the exception handler can be executed immediately after the instruction that was in the execution

phase when the exception occurred. Then when the normal processing resumes, the instruction pointer must be backed up to refetch the instruction immediately following the one that was executing when the exception occurred. Multiple exceptions present additional complications; for example, a divide by zero during the same clock cycle as a page fault.

The CLIPPER architecture supports 18 traps, 256 vectored interrupts, and 128 programmable supervisor calls. The traps are caused by page faults, memory protection violations, floating point errors such as overflow, integer arithmetic errors such as division by zero, and privileged instruction violation by a user-mode program. When one or more of these conditions occurs, the hardware automatically generates the appropriate trap. Interrupts are signaled by activity on the interrupt pins, with the type of interrupt encoded as an eight bit quantity on the interrupt bus. Supervisor calls are made by executing the **calls** instruction with a parameter specifying which call.

No matter what their cause or type, all exceptions are handled in much the same way. First, the current PC, SSW, and PSW are saved on the supervisor stack, then a new SSW and PC are copied from a data structure called the Vector Table. This table, located in the first real page of memory, contains the address and SSW value for every exception handler routine. These address/SSW pairs are stored in predefined locations in the Vector Table, with each location corresponding to a particular type of trap, interrupt, or call. Figure 3-6 shows the layout of the Vector Table.

Real Address (Hex)	Description
Data Memory Trap Group:	
108	Corrected Memory Error
110	Uncorrectable Memory Error
128	Page Fault
130	Read Protect Fault
138	Write Protect Fault
Floating-Point Arithmetic Trap Group:	
180	Floating Inexact
188	Floating Underflow
190	Floating Divide by Zero
1A0	Floating Overflow
1C0	Floating Invalid Operation
Integer Arithmetic Trap Group	
208	Integer Divide by Zero
Instruction Memory Trap Group:	
288	Corrected Memory Error
290	Uncorrectable Memory Error
2A8	Page Fault
2B0	Execute Protect Fault
Illegal Operation Trap Group:	
300	Illegal Operation
308	Privileged Instruction

Real Address (Hex)	Description
Diagnostic Trap Group:	
380	Trace Trap
Supervisor Calls:	
400	Supervisor Call 0
408	Supervisor Call 1
.	
.	
.	
7F8	Supervisor Call 127
Prioritized Interrupts:	
800	Non-Maskable Interrupt
808	Interrupt Group 0 Number 1
810	Interrupt Group 0 Number 2
.	
.	
.	
838	Interrupt Group 0 Number 15
840	Interrupt Group 1 Number 0
848	Interrupt Group 1 Number 1
.	
.	
.	
FF8	Interrupt Group 15 Number 15

Figure 3-6 Exception Vector Table

After the exception has been handled by software, the handler routine executes a **reti** (Return from Interrupt) instruction, which causes the old PC, SSW and PSW values to be restored from the supervisor stack, and the program picks up where it was interrupted.

CHAPTER 4
MEMORY MANAGEMENT

The physical main memory of most computers is organized as a set of consecutively numbered storage cells, each containing a byte of data. The location numbers associated with these storage cells are called **real addresses** (or **physical addresses**), and the set of all the real addresses is called the **real address space** of the computer. The real address space is thus determined by the actual hardware of the computer's memory system.

On the other hand, a program running on the computer generates a set of addresses as it executes, and this set is limited only by the maximum number of bits possible in an address (i.e., the width of the Program Counter). This set of possible addresses is called the **virtual address space** (or sometimes the **logical address space**) of the computer. Note that the virtual address space need not be the same size as the real address space; in fact the virtual space is usually much larger. For example, consider a 32-bit computer, such as the CLIPPER, with 4 million bytes of memory (a fairly typical configuration). A program on this computer can address more than 4 *billion* locations, for the simple reason that 4,294,967,296 (2^{32}) 32-bit numbers exist. Thus the virtual address space is a thousand times larger than the physical memory.

Memory management is basically concerned with the organization of a computer's virtual and real address spaces, how they are related to each other, and how memory can most effectively be used. In this chapter we will discuss three aspects of the CLIPPER's memory management scheme:

memory architecture	The organization of the virtual and real address spaces.
address translation	The mapping between virtual and real addresses; CLIPPER uses a two-level, page-based mapping system, with a 4K byte page size.
virtual memory	Architectural mechanisms in the CLIPPER that permit use of the complete virtual address space in systems with limited physical memory.

In addition, we will describe how the CAMMU chips implement the memory management functions.

4.1 MEMORY ARCHITECTURE

The CLIPPER memory architecture is defined by the layouts of physical address space and virtual address space. Figure 4-1 shows a diagram of the CLIPPER's memory architecture.

4.1.1 Physical Address Space

The maximum amount of physical memory that can usefully be connected to a computer is determined by the number of address lines in the system bus. For the CLIPPER this number is 32, so the CLIPPER's maximum physical memory and thus its maximum real address space is 4 gigabytes (2^{32} bytes). Of course, most systems will not implement the maximum possible real address space; several megabytes will be far more typical. We will however continue to refer to "32-bit real addresses", even though in a system with, say, 16 megabytes of physical memory only the low-order 24 bits of any real address will ever be nonzero (assuming physical memory begins at location 0 and has no gaps).

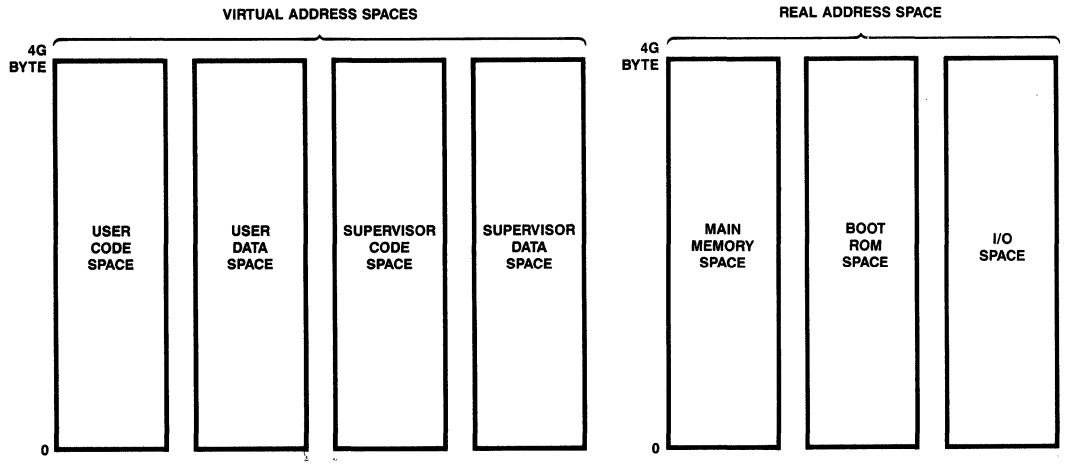


Figure 4-1 CLIPPER Memory Architecture

The CLIPPER puts additional information on the system bus along with the real address; one important item is a 3-bit quantity called the **system tag**. The tag will be described in more detail below, but one of its intended purposes is relevant here, namely, to distinguish three separate physical address spaces:

— main memory space
— boot ROM space
— I/O space

Main memory space is the usual physical read/write memory. A Boot ROM is a separate memory containing initialization code that is automatically activated when the computer system is reset. It is useful to have these two be separate address spaces, so that the Boot ROM can start at location 0 without either permanently taking up a chunk of the lowest addresses in main memory or requiring sophisticated circuitry to switch it in and out of main memory as required. CLIPPER provides this switching logic on chip, so the memory control logic need only decode the system tag and send addresses to main memory or to the Boot ROM as appropriate.

The CLIPPER uses **memory-mapped I/O**, that is, it does not employ special I/O instructions. Instead the usual loads and stores that are employed to access memory are also used for input and output. The I/O devices on the system bus are responsible for recognizing which addresses are intended for them. Memory-mapped I/O is fairly common in microprocessors, but it usually requires dedicating a range of main memory addresses for I/O. In the CLIPPER system, I/O has its own complete 4 gigabyte address space, which doesn't interfere at all with the complete 4 gigabyte main memory space. The I/O devices and memory control logic only have to decode the system tag to determine which of them is the intended recipient of an address.

4.1.2 Virtual Address Space

The virtual address space of the CLIPPER is very straightforward: it is a linear, unsegmented 4 gigabyte (2^{32} bytes) space. The alternative to such a linear space has been a **segmented** address space, which is basically a collection of linear address spaces. In the segmented model, addresses are two-component values, with the *segment number* specifying one of the address spaces, and the *displacement* specifying the offset of the operand within the segment.

Segments were introduced into microprocessor architectures as a compromise between a pure 16-bit linear architecture (with a 16-bit PC and a 64K address space) and a true 32-bit architecture. Each segment was a simple linear 64K byte space, but by using several segments, programmers could work around the address space limits. From the beginning, this approach was difficult to use and trouble prone; and when designers attempted to add memory mapping and virtual memory (see below), the overhead involved in manipulating segments became prohibitive. As a result, the latest generation machines from families with segmented architectures, even though they retain segments for historical reasons, also include mechanisms that permit segmentation to be escaped. The CLIPPER architects started with a clean slate and avoided the problems of segments altogether.

Just as the system tag on the bus effectively creates three separate real address spaces, so the internal operating modes of the CLIPPER create different virtual address spaces. It is useful to think of the CLIPPER as having four virtual address spaces at any one time: **user instruction space, user data space, supervisor instruction space**, and **supervisor data space**. Programs running in user mode generate addresses in one of the two user spaces (just as they access the user registers), while supervisor-mode programs access the two supervisor spaces. As we shall see in the next section, the CLIPPER provides hardware mechanisms that support these four separate spaces.

4.2 ADDRESS TRANSLATION

Address translation is the process which maps virtual address space onto real address space. **Mapping** is the address translation scheme that allows virtual addresses to be translated into *arbitrary* real addresses; it provides a kind of generalized relocation mechanism.

Unmapped memory systems simply equate virtual addresses to real addresses — virtual address 00001234 becomes real address 00001234. This approach is appropriate for simple, single-user or embedded systems, which is why most 8- or 16-bit microprocessor applications have been unmapped.

Most 32-bit applications, on the other hand, will employ some form of multiprogramming (either multitasking or multiuser operation), and in these cases mapping is essential to prevent the multiple programs from interfering with each other. Mapping allows different programs, users, or processes to have their own virtual address spaces, almost as if each had the computer to itself. The mapping hardware provides the mechanism for translating these independent virtual address spaces into the same physical space and for protecting each user's space from interference by another program. Without mapping, multiprogramming is a very difficult and risky proposition.

For efficiency, mapping is usually done in blocks of addresses instead of independently for each virtual address. The earliest mapping systems translated the entire virtual address space of a program as a single unit (see Figure 4-2). More recent systems employ smaller blocks in order to get a more fine-grained control over memory. The most widely used systems are based on fixed-sized blocks called **pages**, which are usually some low multiple of 1K bytes in length. The CLIPPER uses a paged-based mapping scheme with 4K byte pages.

In the page-based mapping system, virtual address space is broken up into thousands of pages, each with the same size. Similarly, real address space is broken up into pieces, also called pages, having this same size. The mapping operation associates a page in real space with each page of virtual space (see Figure 4-3). Since mapping is done in units of a page, two addresses close to each other in the same virtual page will also be close in the same real page, but two contiguous virtual pages may not be contiguous in real memory.

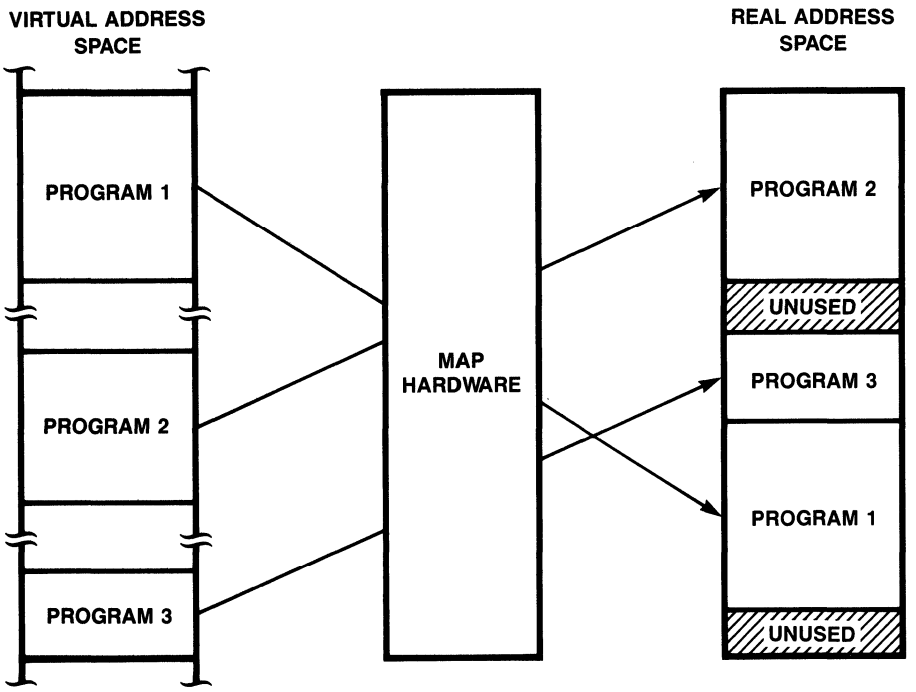


Figure 4-2 Simple Mapping

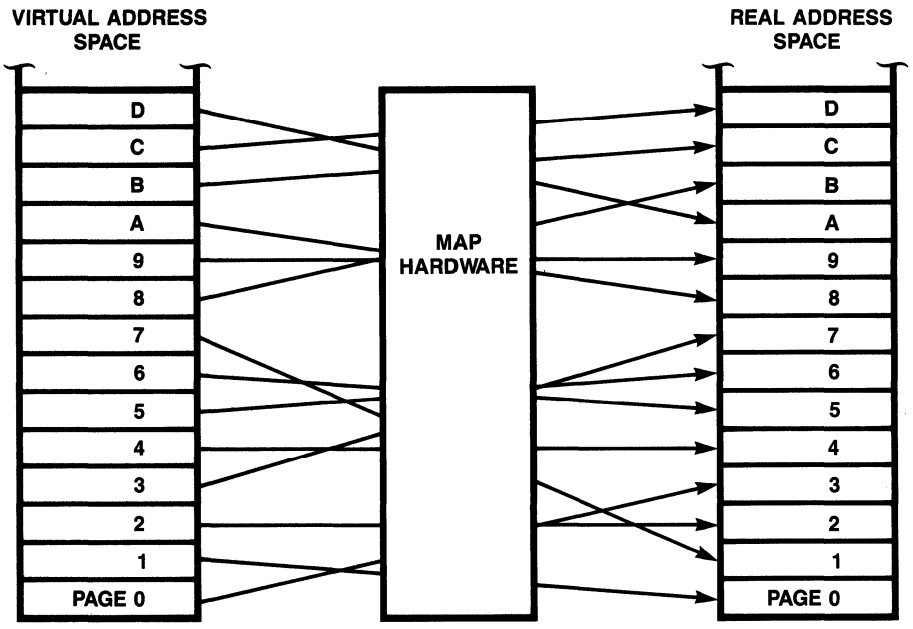


Figure 4-3 Page Based Mapping

4.2.1 CLIPPER Two-Level, Page-Based Mapping

The CLIPPER's virtual address space of 4 gigabytes is divided into 1 million pages, each with 4K bytes. The real address space is similarly divided into 4K-byte page frames. The 4K-byte size was chosen after careful study. It is consistent with the industry trend toward larger pages, and it helps performance in four ways:

1. It provides a high hit rate for the TLB (the on-chip buffer of page addresses described below).
2. It is an efficient unit of disk transfer, thus improving I/O performance.
3. It allows a larger cache to be accessed concurrently with the address translation process — since fewer bits participate in address translation, more can be used to select a set in the cache.
4. It permits a two-level mapping (see below); smaller pages would require more levels.

When mapping is enabled (by setting a bit in the System Status Word), the CAMMUs translate virtual addresses generated by the CPU/FPU into real addresses. In much the same manner as in mainframes, the translation process is accomplished with a two-level hierarchy of **page tables** that is shown in Figure 4-4. Each process has its own collection of page tables that define its virtual-to-real address map and thus its own address space. The base of the hierarchy is the **page table directory**; it is one page long and contains 1024 32-bit entries, each of which can point to a page table. Page tables themselves are also one page long and contain pointers to pages.

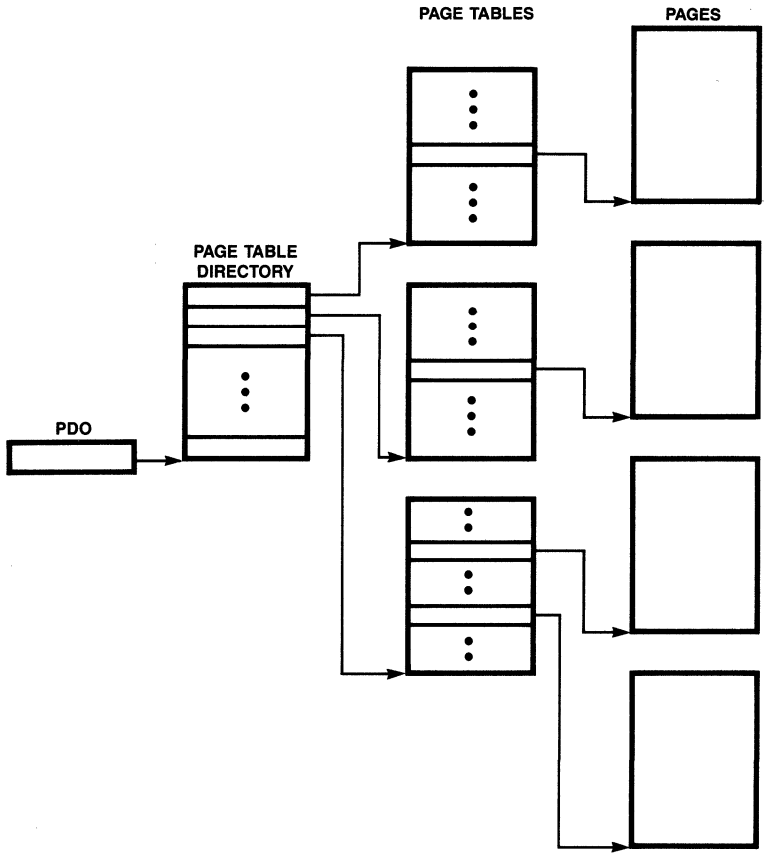


Figure 4-4 Two-Level Address Translation

Each CAMMU contains two special **Page Directory Origin** (PDO) registers; one points to the base of the page table directory of the supervisor mode program (the operating system), the other points to the base of the page table directory for the currently executing process. Upon a process swap, the OS can simply change the user PDO to obtain a new user address space. Since there are separate CAMMUs for instructions and data, each with its own PDO, there are actually four memory maps, and thus four address spaces, active at any one time:

— supervisor instruction space
— supervisor data space
— user instruction space
— user data space

Figure 4-5 shows in more detail how the address translation process works. The operating system has programmed the PDO register to point to the base of the appropriate page table directory. When the CPU sends a virtual address to the CAMMU, the upper ten bits of the virtual address are used as an index into the current page table directory. The selected entry contains the real address of a page table. The middle ten bits of the virtual address are used as an index into this page table. The entry selected this time contains the real address of a page in real memory. Concatenating the lower 12 bits of the virtual address to the real address of the memory page produces the 32-bit real address of the operand.

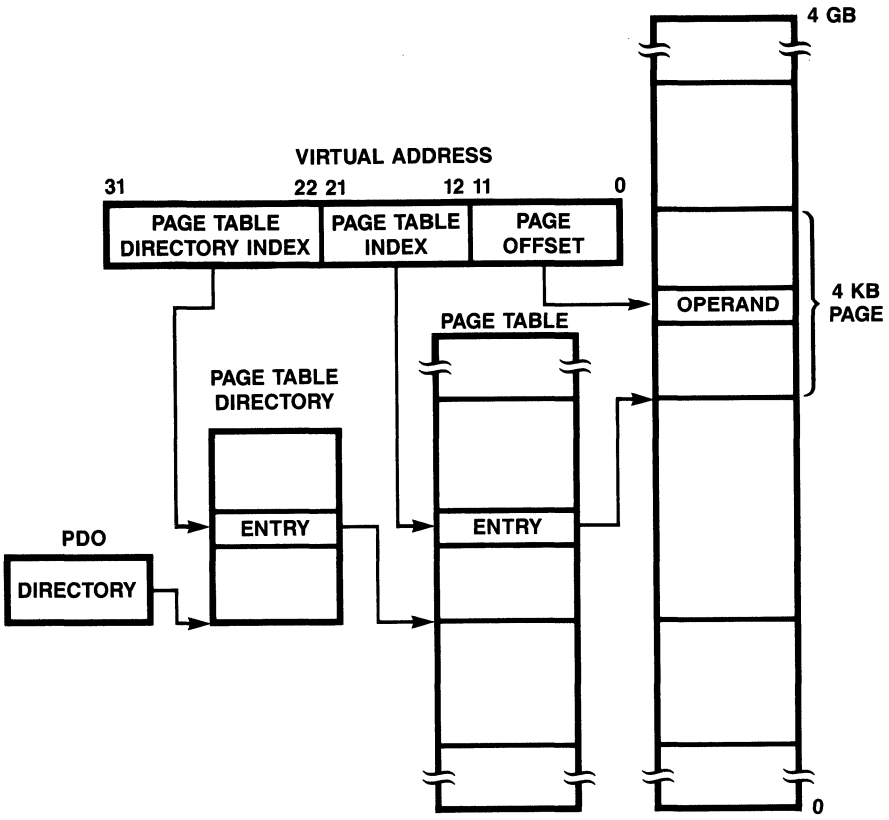


Figure 4-5 CLIPPER Address Translation

To save the overhead of page table lookups every time an address is sent from the CPU, each CAMMU caches address translation information on 128 frequently used pages in a Translation Lookaside Buffer (TLB). This TLB is searched concurrently with cache accesses in the CAMMU; only when mapping data for the requested page is not in the TLB does an access to a memory-resident directory or page table have to be made. The CLIPPER TLB has a hit rate in excess of 99%.

As was mentioned above, the CLIPPER at any one time has four virtual address spaces (supervisor data and instruction, user data and instruction) and three real address spaces (main memory, boot ROM, and I/O). The PDOs in each CAMMU determine the maps that define the two supervisor and two user virtual spaces. The page table entries (see Figure 4-6) contain the system tags that define which of the three real address spaces is accessed. When a virtual address is translated, this tag value is put on the system bus along with the translated real address. The table entry also contains protection information about the page. For example, pages can be set to be read-only or execute only, or to limit access to the supervisor. Other bits in the table entry are useful for virtual memory (see below).

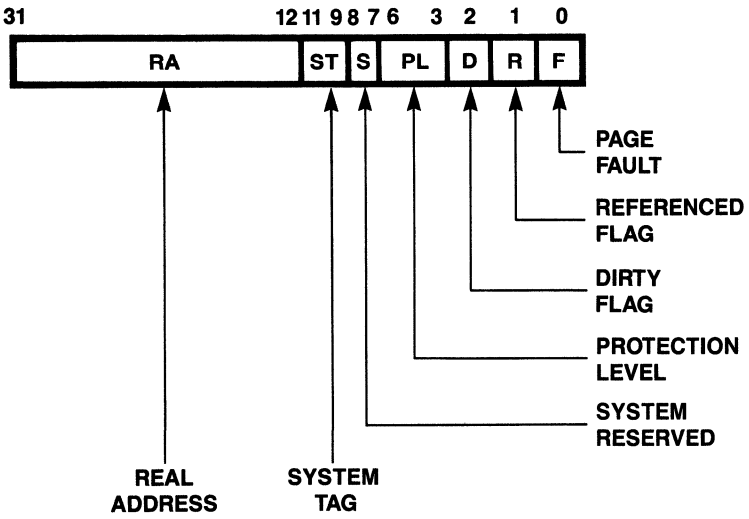


Figure 4-6 Page Table Entry

The lowest 8 pages of the supervisor virtual address space are permanently mapped, via hardwired logic in the CAMMU. Figure 4-7 shows this mapping. Virtual pages 0-3 are translated into real pages 0-3 (main memory); virtual pages 4-5 are translated into real pages 0-1 (I/O); and virtual pages 6-7 are translated into real pages 0-1 (Boot ROM). This permanent mapping provides several benefits: it makes the Boot ROM immediately available on reset; it also makes some I/O available during initialization; finally, it insures that the lowest 3 pages of the supervisor's address space (which are in constant use, since they contain the exception vector table) are always translated rapidly.

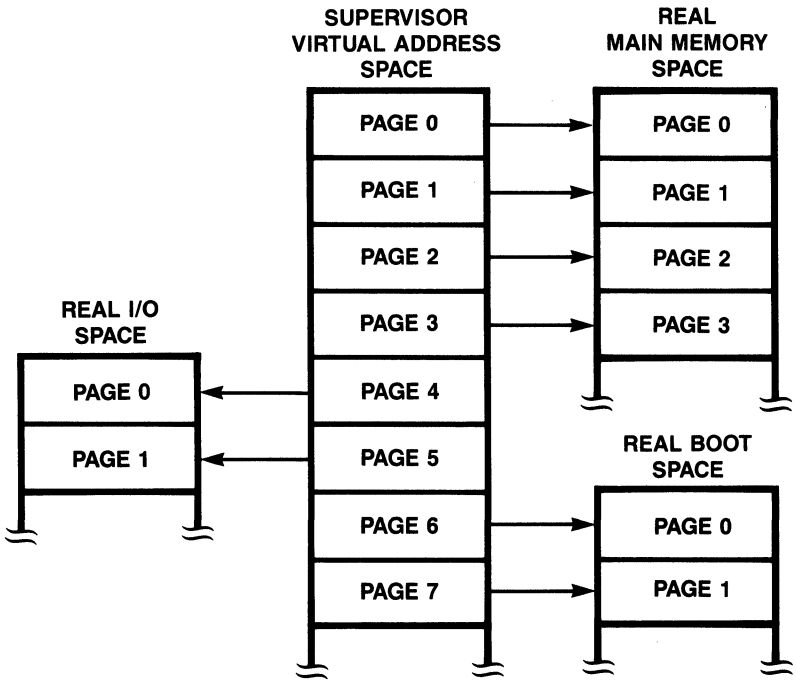


Figure 4-7 Hardwired Mapping

Pages may be shared between processes by putting an entry for the same real page frame in page tables belonging to each process. The supervisor may access user pages by a similar mechanism — mapping one of its pages into a page frame that is also being used by a user program. The CLIPPER also provides a special mechanism that lets the supervisor use the user PDO for operand addresses; this facilitates rapid access by the supervisor to the entire user address space.

4.3 VIRTUAL MEMORY

Virtual memory is an operating system mechanism (supported by computer hardware) that allows large programs, or groups of programs, to circumvent the limitations on the amount of physical memory in a computer system by exploiting disk storage for some pages. In a virtual memory system, it appears to the user that the entire virtual address space is available for access; but at any given moment only a few virtual pages are actually mapped into physical address space. The rest are stored on disk.

Whenever the processor generates a virtual address, the hardware checks to see if that address lies in a page that is actually in memory. If it does, the address is translated normally. However, if the page is not in memory, an operation called a **page swap** is performed, and the operating system loads the missing page from disk. If this swap is performed rapidly enough and if missing pages are relatively infrequent, the virtual memory system will perform nearly as well as one with far more memory, and at a fraction of the cost. This process, called **demand paging**, is diagrammed in Figure 4-8.

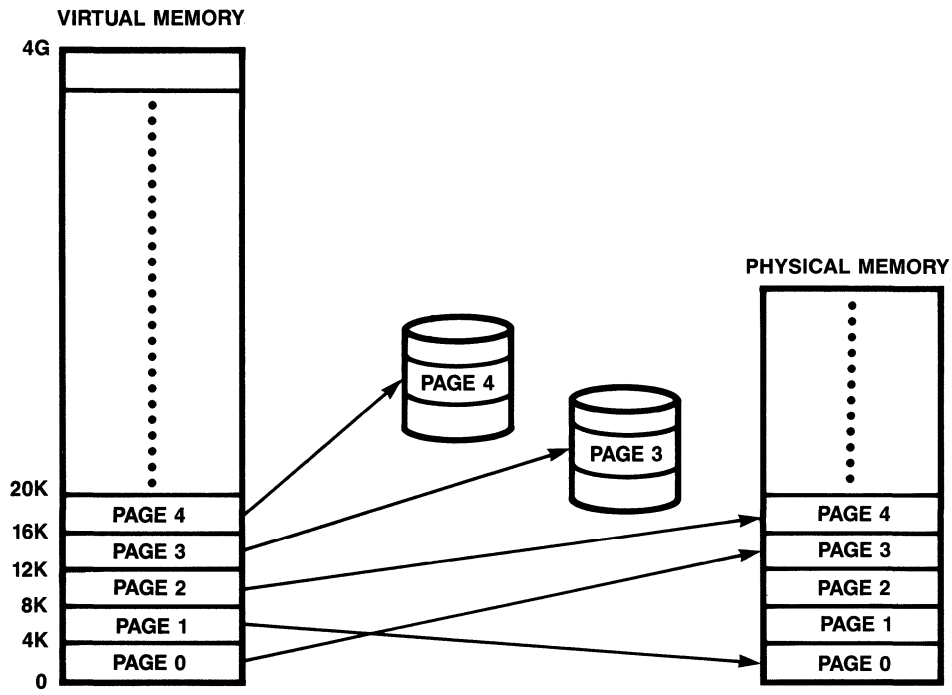


Figure 4-8 Demand Paging

The CLIPPER architecture provides four key architectural features to support demand-paged virtual memory:

1. A bit in the page table entry for each page that tells the CAMMU if a page is absent from main memory.

2. A special processor trap, called a **page fault**, that can be activated by the CAMMU when a not-present page is accessed.

3. The capability of aborting instruction execution when a page fault occurs and re-executing or resuming the instruction after the operating system has loaded the missing page.

4. Bits in the page table entries that facilitate the choice of the optimal page to replace when a newly swapped-in page must evict a page currently in memory.

As Figure 4-6 shows, a CLIPPER page table entry contains the page fault (F) bit which is set if the corresponding page is absent from memory. The page fault exception is one of the 18 hardware trap conditions; when it is activated, the operating system can inspect the address that caused the fault (it is stored in a special CAMMU register), then swap in the missing page. And all CLIPPER instructions are fully restartable.

Initially, all pages are on disk and memory is totally free. But as the program executes and generates addresses that cause page faults, more and more pages are brought into memory. Eventually memory

becomes full, and newly swapped-in pages must replace pages in memory. At this point the operating system attempts to replace pages that it predicts are least likely to be referenced in the near future. One popular algorithm for making this prediction is to select the page that was least recently accessed. The referenced (R) bit in the page table entry supports this algorithm. This bit is set automatically whenever the corresponding page is accessed. By periodically examining and clearing the bit, the operating system can identify pages that have not been recently used.

Once the operating system has selected a page to evict, it must decide whether to write the page back to disk, or simply discard it. The dirty (D) bit facilitates this decision. This bit is automatically set whenever the corresponding page is modified. Clearly, if this bit has not been set, then the data on disk already matches the page and there is no need to write the page back.

4.4 CLIPPER MMU IMPLEMENTATION

The CAMMU contains the cache mechanism and the memory management unit (MMU). The cache was described in Chapter 2 (see especially Figure 2-8); here we will cover the MMU. Basically the MMU contains two parts — the dynamic translation unit (DTU) and the translation lookaside buffer (TLB). Figure 4-9 shows a diagram of the CAMMU; the right side comprises the MMU.

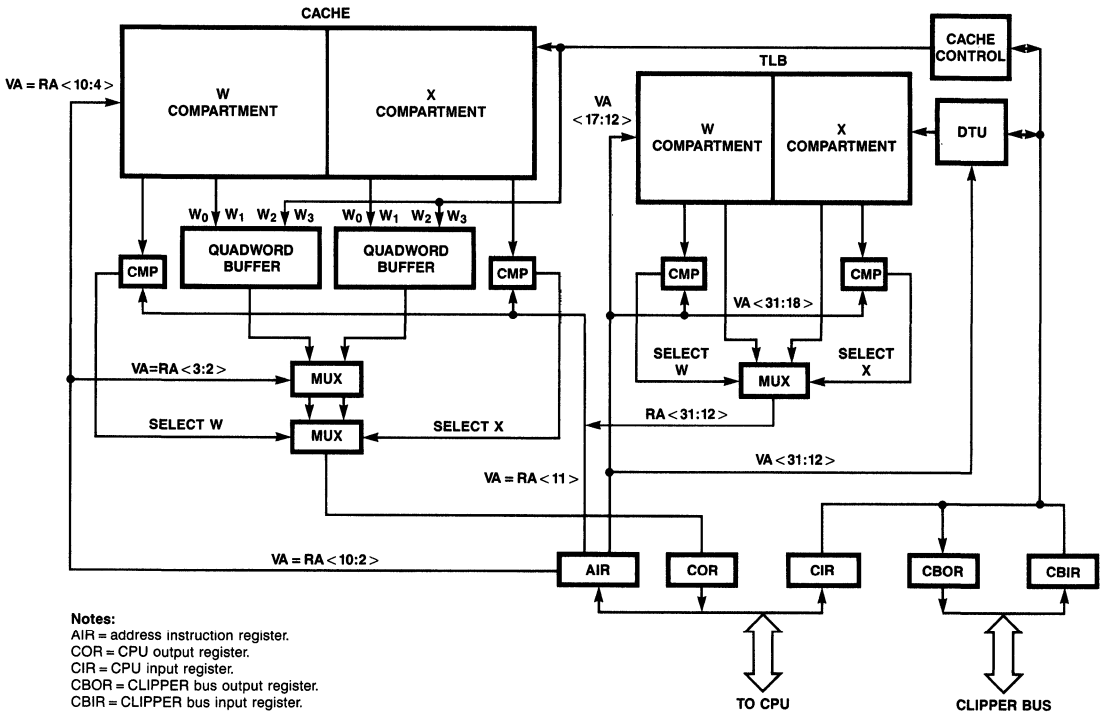


Figure 4-9 CLIPPER CAMMU

The TLB is a two-way set associative cache that is used by the MMU for fast address translation requiring no access to main memory. It consists of 64 sets of lines, with each set containing a W and an X compartment line. Figure 4-10 shows an abstract picture of the TLB, and Figure 4-11 shows the format of a TLB line.

The TLB operates concurrently with and in a very similar fashion to the cache for each virtual address presented to the CAMMU. Bits 12-17 of the virtual address are used to select a TLB line set at the same time bits 4-10 select a cache line set. Then bits 18-31 of the virtual address are compared with the virtual address field (VA) of both lines in the set. If a match occurs, we have a TLB hit, and the corresponding real address field (RA) is used by the cache in its comparisons. If there is no match, we have a TLB miss, and the DTU attempts address translation using page tables in main memory. Once the DTU has completed this process, the CAMMU updates the TLB with this latest translation, replacing an existing line if necessary. Then the cache access occurs again.

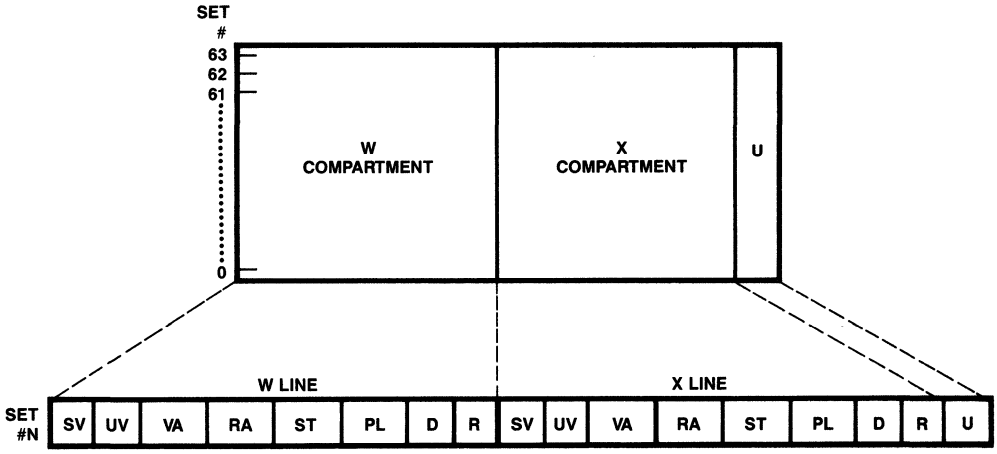


Figure 4-10 CLIPPER TLE

TLB Line Format

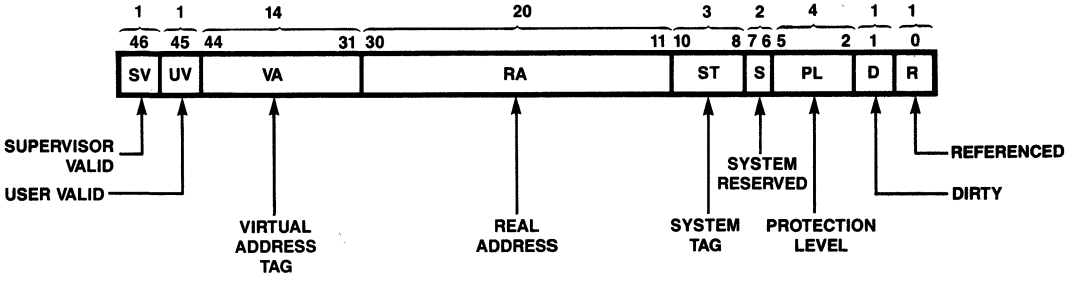


Figure 4-11 TLB Line Format

CHAPTER 5
SYSTEM ARCHITECTURE

Thus far we have focused primarily on the internal and instruction set architectures of the CLIPPER. This final chapter is devoted to the remaining features of the architecture of CLIPPER based systems. We will use the expression **system architecture** as a shorthand expression for these features.

System architecture refers to two separate but intimately related topics:

1. The *system configuration* architecture. This is basically a hardware notion; it means all the component parts of a CLIPPER based system: the CLIPPER module itself, the system bus, and the memory and I/O subsystems.

2. The *system software* architecture. This is a software notion; it means the application interface provided by the operating system and high-level languages. In addition, this topic will include the CLIPPER software development environments.

5.1 SYSTEM CONFIGURATION

CLIPPER based systems are modular. They consist of the CLIPPER module itself and two main subsystems: memory and I/O. Connecting the subsystems and the CLIPPER module is a bus called the **CLIPPER Bus**. Figure 5-1 shows a high-level diagram of a complete CLIPPER computer system.

5.1.1 CLIPPER Module

The CLIPPER Module is a small multilayer printed circuit board holding the CLIPPER CPU/FPU chip, the two CAMMU chips, and a clock chip. The components are surface mounted. A 96-pin DIN standard connector is also mounted on the edge of the module; this connector is used to interface the module to the rest of the system.

Seventy-three (73) of the 96 pins in the DIN connector are utilized for signals; these lines define the **CLIPPER Module Interface (CMI).** The CMI include 32 address/data lines, 8 interrupt vector lines, 6 lines defining the type of operation in progress (read, write, length of transfer, etc.), 3 lines defining the system tag, and a number of control lines, including interrupt control and bus arbitration for systems with multiple bus masters. Figure 5-2 shows the CLIPPER Module and CMI.

Two clock signals are generated by the clock chip, MCLK and BCLK. MCLK is the internal CLIPPER master clock, used to drive the CPU/FPU, the CAMMUs, and associated logic. The frequency of MCLK is half the frequency of the module's crystal, i.e., 33.3 MHz. BCLK is a signal on the CMI; it is the system clock, used for timing on the system bus. BCLK's frequency is one fourth the crystal frequency, i.e., 16.7 MHz.

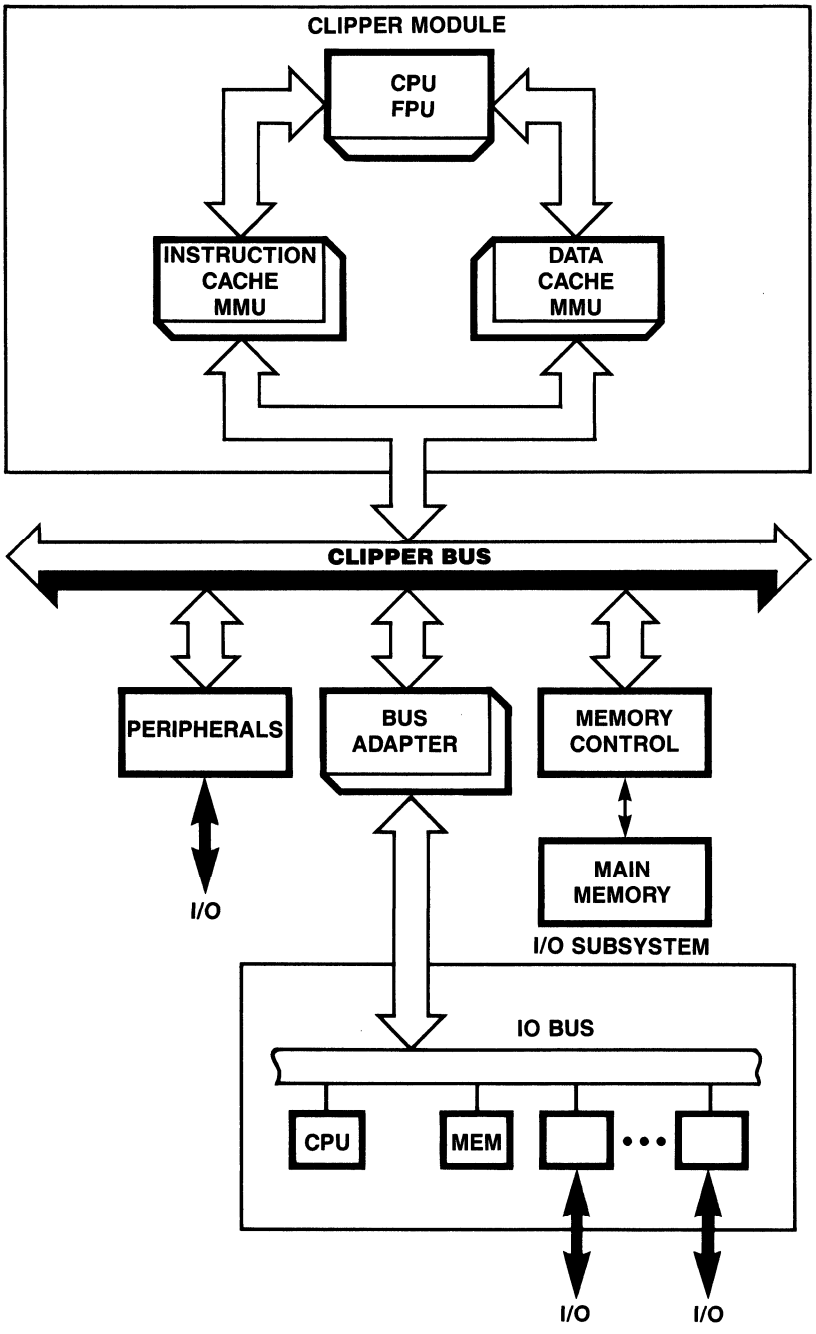


Figure 5-1 CLIPPER System

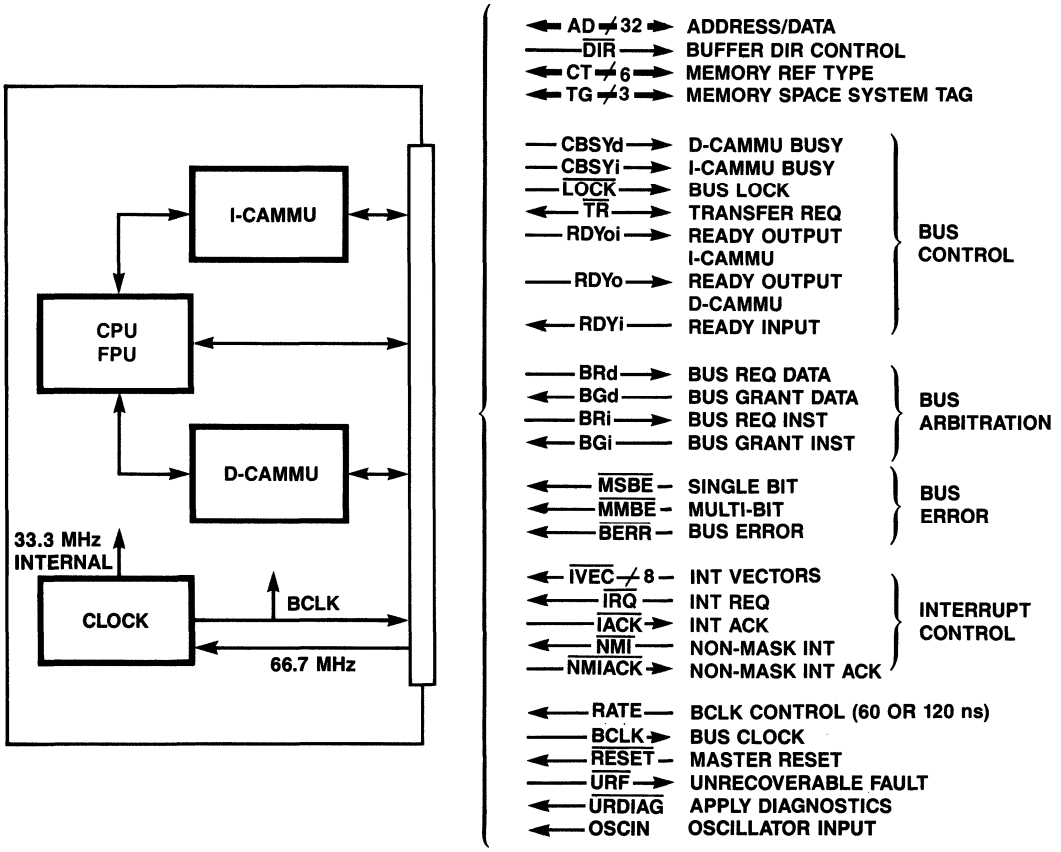
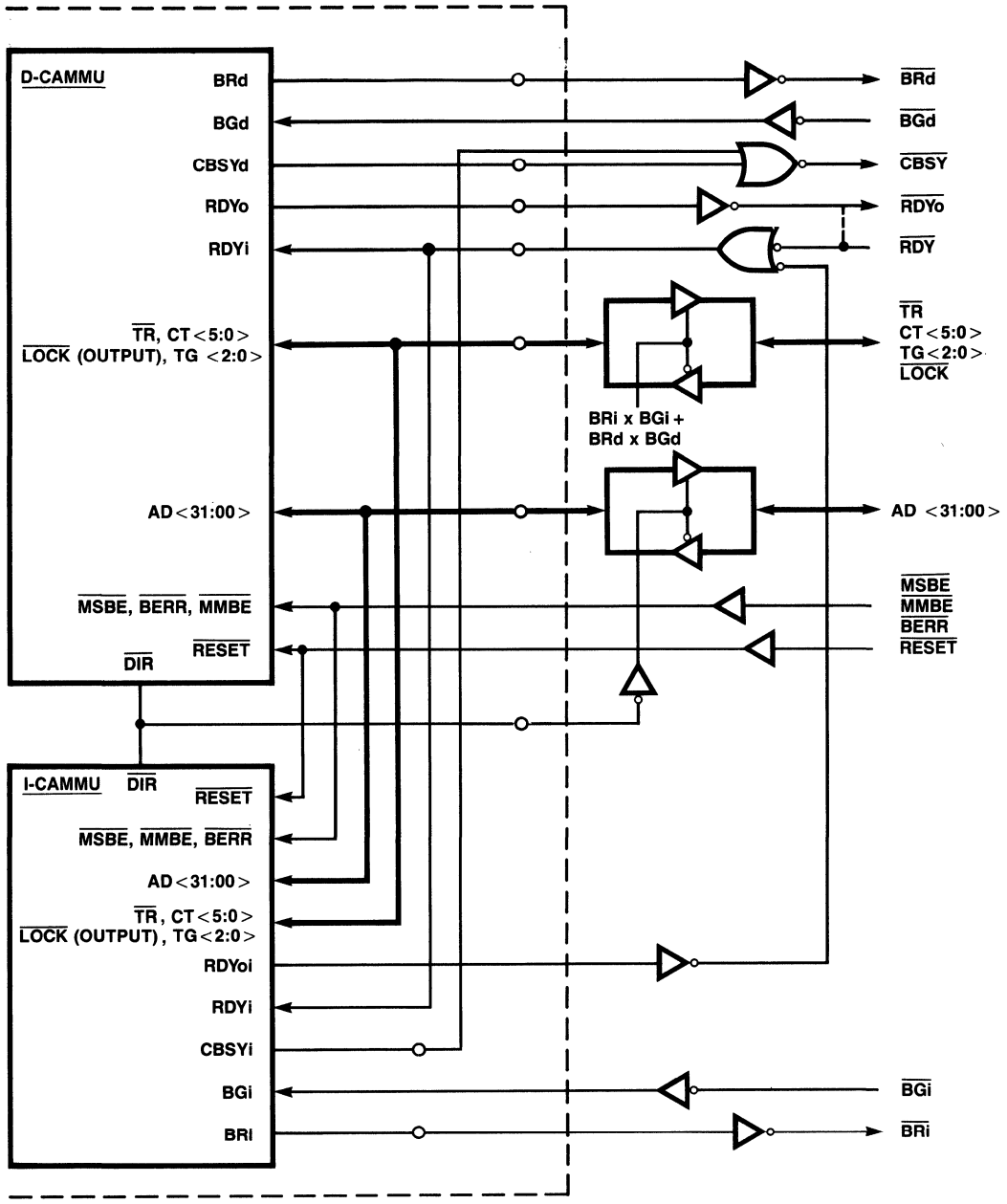


Figure 5-2 CLIPPER Module Interface

5.1.2 CLIPPER Bus and Subsystems

The lines from the CMI are connected to the system bus (the CLIPPER bus) through the buffers and simple interfacing logic shown in Figure 5-3. The CLIPPER Bus is a high-speed, 32-bit synchronous bus designed to support multiple bus masters (either multiple CPUs or CPUs and I/O controllers).

Bus masters gain control of the bus through **arbitration** logic. A would-be bus master asserts the Bus Request (BR) signal, and if the arbitration logic decides to grant it control of the bus, the arbitrator asserts the Bus Grant signal (BG). The CLIPPER Module uses the signals like any other bus master. Devices needing to interrupt the CPU employ the interrupt lines. A CLIPPER system will also contain an **interrupt controller** that coordinates the interrupts coming from several devices. In the case of simultaneous interrupt requests



NOTE: This diagram does not include interrupt signals.

Figure 5-3 CLIPPER Bus

from several devices, the controller selects the one with the highest priority, asserts the CLIPPER Bus interrupt request line, and puts the appropriate 8-bit interrupt vector on the interrupt vector lines. Figure 5-4 contains a diagram of a CLIPPER system, showing the arbitration and interrupt logic.

The memory subsystem is fairly straightforward. Because of the quadword (16 byte) line size, the memory must implement burst mode transfers, that is, it must transfer four successive words whenever an address is sent by a bus master. One way to set this up is to create a **four-way interleaved** memory. Memory is organized in four banks of words. When an address is sent to memory, all four banks are accessed simultaneously and the data is transferred on the next four bus cycles. Another method is to use nibble-mode RAMs.

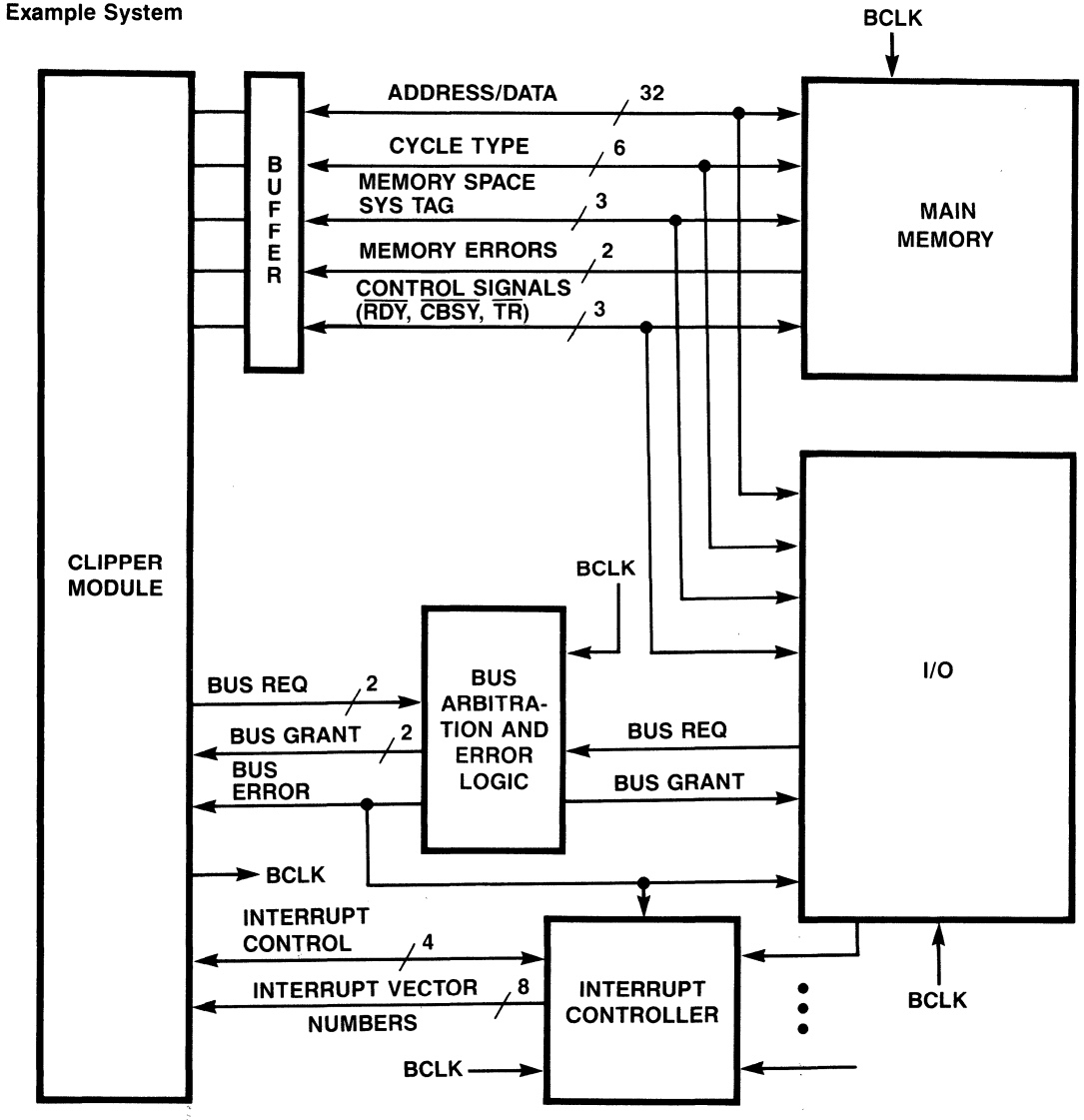


Figure 5-4 Example CLIPPER Configuration

The I/O subsystem can be implemented in several ways. The simplest mechanism is for device controllers to interface directly to the CLIPPER bus. This design is the same as with traditional microcomputer systems, but it burdens the main CPU with servicing individual devices. A more efficient way is to off-load device handling on a specialized I/O Processor (IOP). The IOP has its own local bus to which the device controllers are interfaced. When a device generates an interrupt, the interrupt is handled by the IOP, not the CLIPPER. The IOP can consolidate many device data transfers into one large block of data that is sent to the CLIPPER using a single CLIPPER interrupt. The result is much more efficient use of the CLIPPER's power. This technique is used often in mainframes and supercomputers, where the IOP is usually called a **channel.**

5.2 SYSTEM SOFTWARE

The CLIPPER high-level architecture is defined by the standard UNIX System V operating system provided with the processor, and by the standard high-level languages that are part of the software development environment.

5.2.1 UNIX System V

The CLIPPER port of UNIX is a high-performance implementation of the latest release of UNIX System V. UNIX System V is a universally accepted standard operating system, with a large amount of application software that can readily be inherited by CLIPPER based systems.

The CLIPPER implementation of UNIX System V makes use of CLIPPER's caching features as well as its demand-paged virtual memory mechanisms. Read, write, and execute access protection is used on a per-page basis, and all three caching strategies are employed:

— noncacheable: used for I/O and special purpose memory.
— write-through: used for shared pages.
— copy-back: used for process data and stacks.

The CLIPPER UNIX was designed to work in distributed I/O systems with an IOP. I/O drivers have been designed that can run on the IOP and pass messages to the main CLIPPER-hosted kernel. A distributed line discipline can be implemented to increase terminal handling capability.

The AT&T specified extensions have been implemented, namely, interprocess communication, shared memory, and file locking. In addition, Fairchild has a commitment to pass on all updates and enhancements as they are made available.

5.2.2 Optimizing Compilers

Fairchild provides optimizing compilers, initially for the following high-level languages:

— C
— Pascal
— FORTRAN

Each compiler is tuned to the CLIPPER architecture and designed to produce efficient and highly optimized code. Among the optimizations performed are

— Global register allocation by coloring.
— Loop optimizations, such as loop invariant analysis, local operator strength reduction, and loop rotation.
— Elimination of redundant computations.
— Constant expression folding.

— Copy propagation (for example, passing parameters in registers).

— Entry and exit code reduction (avoiding use of the frame pointer).

— Static address elimination.

One of the most important of these optimizations is global register allocation. The compiler performs a global data flow analysis to determine the lifetime of all local variables. Then, a **coloring** algorithm (the name comes from minimal map coloring problems in mathematics) is used to fit the variables into the register set (see Figure 5-5). Because of CLIPPER's large register set, most local variables are stored in registers for the entire length of their active lifetimes. The result is a dramatic improvement in execution speed.

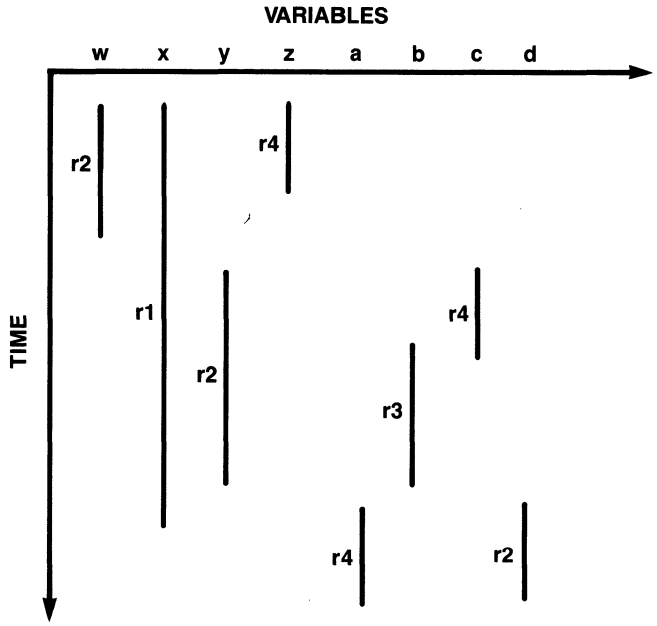


Figure 5-5 Register Coloring

5.2.3 Software Development Environments

The compilers are provided as part of a software development environment. Two different versions of the environment are available:

— The **self-hosted environment**, in which the tools run on the CLIPPER UNIX System V and also target the CLIPPER.

— The **cross-development environment**, in which the tools run on VAX hosts and target the CLIPPER.

The self-hosted environment is provided by add-in boards containing the CLIPPER and UNIX System V that plug into popular workstations.

Besides the compilers, the software development environment includes a number of useful utilities, including an assembler and linker, an interactive debugger, and a profiler for timing analysis. The cross environment also includes a simulator that provides a simulated CLIPPER execution environment on the VAX.

APPENDIX A
COMPATIBILITY

The CLIPPER's quantum leap in performance was made possible by its extensive improvements over the internal architecture of conventional 16-bit (and 32-bit) microprocessors; in Figure 5-8 this change is represented by the large lateral displacement of the lower level. These internal changes required that less extensive, but important changes be made to the CLIPPER's instruction set architecture (represented in the figure by the moderate lateral displacement of the middle levels). But no changes at all were required in the application architecture; programs written in C, FORTRAN, or Pascal, executing under UNIX, need only be recompiled for the new machine; no other changes need be made.

The dashed lines show that if an attempt had been made to freeze the instruction set instead of the application architecture, then much less extensive changes to the internal architectures could have been made, and the performance gains would have been correspondingly less dramatic. Any attempt to preserve instruction set compatibility with an existing 16-bit architecture drastically reduces the scope of the performance improvements that can be made.

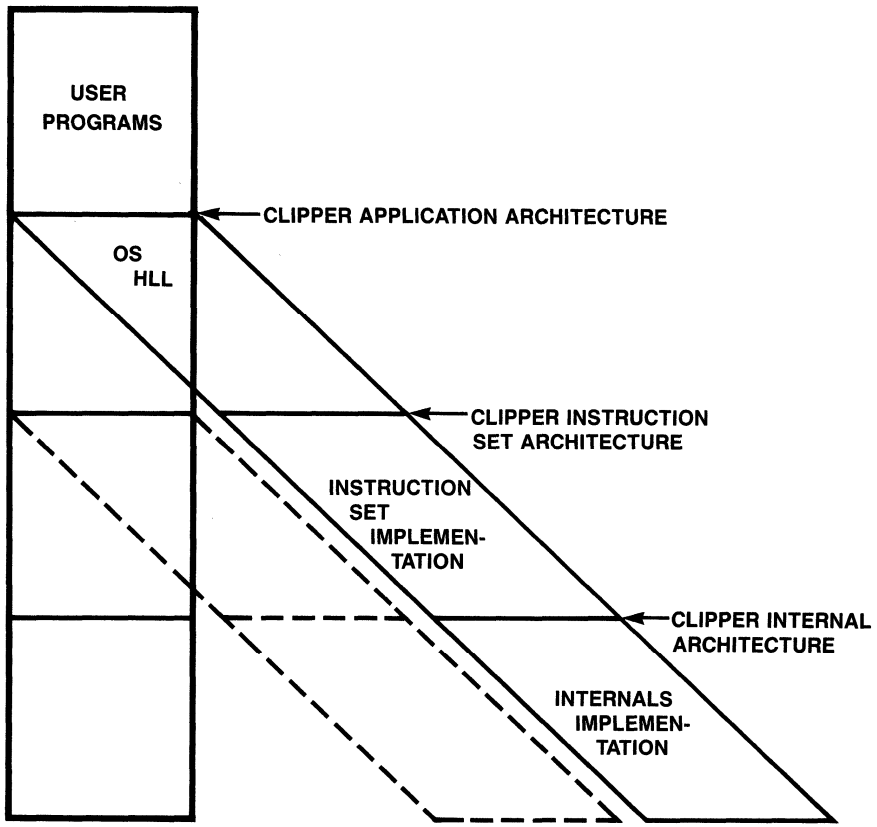


Figure A-1 CLIPPER Architectural Change

Application architecture compatibility is the way to preserve your investment in the software that you have developed as well as the software whose source code you own. But CLIPPER also provides a way to preserve your purchased binary code, while making possible increased performance for new programs.

Like most supercomputers and mainframes, CLIPPER-based systems use a separate I/O subsystem with its own I/O processor for handling all input/output functions, including disk and terminal I/O. This I/O subsystem is a computer system in its own right, and it can run its own binary programs. For example, the CLIPPER could be installed in an existing Personal Computer on an add-in board, and use the PC's own CPU as an I/O processor. The resulting system would have all the power of the CLIPPER for new programs, and be able to run all existing PC software without change.

Other strategies for making the transition from 16 to 32 bits invariably wind up crippling the performance of the 32-bit machine. We have seen how preserving strict binary compatibility with an existing 16 bit architecture limits the 32-bit processor. The other strategy that is often mentioned is the use of a **virtual machine**, that is the creation by the 32-bit hardware and OS of a simulated 16-bit execution environment. Whenever a 16-bit program running in the simulated environment tries a disallowed instruction (e.g. I/O), the hardware traps to the operating system, which simulates the instruction in a protected way. The unfortunate result is that the 16-bit programs often run *slower* than they did on the 16-bit machine itself, and any 32-bit programs running concurrently are significantly slowed by the high OS overhead.

So to summarize the compatibility issue: The CLIPPER maintains architectural compatibility at the right level — the application architecture; and it also provides, via its I/O subsystem, a way for existing 16-bit binaries to be run without modification.

FAIRCHILD
A Schlumberger Company

U.S. AND CANADIAN SALES OFFICES

ALABAMA (SE)

Huntsville Office
555 Sparkman Dr., Suite 1030
Huntsville, AL 35805
Tel: 205-837-8960

ARIZONA (W)

Phoenix Office
9201 N. 25th Ave., Suite 215
Phoenix, AZ 85021
Tel: 602-943-2100

CALIFORNIA (W)

Auburn Office
320 Aeolia Drive
Auburn, CA 95603
Tel: 916-823-6664

*** Costa Mesa Office**
3505 Cadillac Ave., Suite O-014
Costa Mesa, CA 92626
Tel: 714-241-5900
TWX: 910-595-1109

Encino Office
Crocker Bank Building
15760 Ventura Blvd., Suite 1027
Encino, CA 91436
Tel: 818-990-9800
TWX: 910-495-1776

*** Cupertino Office**
10400 Ridgeview Court
Cupertino, CA 95014
Tel: 408-864-6200

San Diego Office
4355 Ruffin Rd., Suite 100
San Diego, CA 92123
Tel: 619-560-1332

COLORADO (W)

Denver Office
10200 East Girard
Building B, Suite 222
Denver, CO 80231
Tel: 303-695-4927

CONNECTICUT (NE)

Woodbridge Office
131 Bradley Road
Woodbridge, CT 06525
Tel: 203-397-5001

FLORIDA (SE)

Ft. Lauderdale Office
450 Fairway Dr., Suite 107
Deerfield Beach, FL 33441
Tel: 305-421-3000

Altamonte Springs Office
Crane's Roost Office Park
399 Whooping Loop
Altamonte Springs, FL 32701
Tel: 305-834-7000
TWX: 810-850-0152

GEORGIA (SE)

Norcross Office
3220 Pointe Parkway
Suite 1200
Norcross, GA 30092
Tel: 404-441-2740
TWX: 810-766-4952

ILLINOIS (C)

Itasca Office
500 Park Blvd., Suite 575
Itasca, IL 60143
Tel: 312-773-3133
TWX: 910-651-0120

INDIANA (SE)

Indianapolis Office
7202 North Shadeland, #205
Castle Point
Indianapolis, IN 46250
Tel: 317-849-5412
TWX: 810-260-1793

IOWA (C)

Cedar Rapids Office
373 Collins Road N.E., Suite 200
Cedar Rapids, IA 52402
Tel: 319-395-0090

KANSAS (C)

Overland Park Office
8600 W. 110th St., Suite 209
Overland Park, KS 66210
Tel: 913-649-3974

Wichita Office
2400 Woodlawn, Suite 221
Wichita, KS 67220
Tel: 316-687-1111
TWX: 710-826-9654

MARYLAND (SE)

Columbia Office
2000 Century Plaza
Suite 114
Columbia, MD 21044
Tel: 301-381-2500
TWX: 710-826-9654

MASSACHUSETTS (NE)

*** Waltham Office**
1432 Main Street
Waltham, MA 02154
Tel: 617-890-4000

MICHIGAN (SE)

Farmington Hills Office
21999 Farmington Road
Farmington Hills, MI 48024
Tel: 313-478-7400
TWX: 810-242-2973

MINNESOTA (C)

*** Minneapolis Office**
3600 W. 80th St., Suite 590
Bloomington, MN 55431
Tel: 612-835-3322
TWX: 910-576-2944

NEW JERSEY (NE)

New Jersey Office
Vreeland Plaza
41 Vreeland Avenue
Totowa, NJ 07512
Tel: 201-256-9011

NEW MEXICO (W)

Albuquerque Office
North Building
2900 Louisiana N.E., Suite D
Albuquerque, NM 87110
Tel: 505-884-5601
TWX: 910-379-6435

NEW YORK (NE)

Endwell Office
421 East Main Street
Endicott, NY 13760
Tel: 607-757-0200

Fairport Office
815 Ayrault Road
Fairport, NY 14450
Tel: 716-223-7700

Hauppauge Office
300 Wheeler Road
Hauppauge, NY 11788
Tel: 516-348-0900
TWX: 510-221-2183

Poughkeepsie Office
19 Davis Avenue
Poughkeepsie, NY 12603
Tel: 914-473-5730
TWX: 510-248-0030

NORTH CAROLINA (SE)

Raleigh Office
5970-C Six Forks Road
Raleigh, NC 27609
Tel: 919-848-2420

OHIO (SE)

Cleveland Office
6133 Rockside Rd., Suite 407
Cleveland, OH 44131
Tel: 216-447-9700

Dayton Office
7250 Poe Avenue, Suite 260
Dayton, Ohio 45414
Tel: 513-890-5813

OREGON (W)

Portland Office
6600 S.W. 92nd Ave., Suite 27
Portland, OR 97223
Tel: 503-244-6020
TWX: 910-467-7842

PENNSYLVANIA (SE)

Willow Grove Office
Willow Wood Office Center
3901 Commerce Ave., Suite 110
Willow Grove, PA 19090
Tel: 215-657-2711

TEXAS (C)

Austin Office
8240 Mopac Expressway
Suite 270
Austin, TX 78759
Tel: 512-346-3990

*** Richardson Office**
1702 No. Collins St., Suite 101
Richardson, TX 75081
Tel: 214-234-3811
TWX: 910-867-4824

Houston Office
9896 Bissonnet-2, Suite 470
Houston, TX 77036
Tel: 713-771-3547
TWX: 910-881-8278

UTAH (W)

Salt Lake City Office
5282 South 320 West
Suite D120
Murray, UT 84107
Tel: 801-266-0773

WASHINGTON (W)

Bellevue Office
11911 N.E. First St., Suite 310
Bellevue, WA 98005
Tel: 206-455-3190

CANADA (NE)

Toronto Regional Office
2375 Steeles Ave. West
Suite 203
Downsview, Ontario M3J 3A8
Canada
Tel: 416-665-5903
TWX: 610-491-1283

Montreal Office
3675 Sources Blvd., Suite 203
Dollard des Omeaux
Quebec H9B 2K4 Canada
Tel: 514-683-0883

Ottawa Office
148 Colonnade Rd. S., Unit 3
Nepean, Ontario K2E 7J5
Canada
Tel: 613-226-8270
TWX: 610-562-1953

* FAIRTECH Design Center

FAIRCHILD
A Schlumberger Company

INTERNATIONAL SALES OFFICES

AUSTRALIA

Fairchild Australia Pty. Ltd.
Suite 1, 1st floor
366 Whitehorse Road
Nunawading 3131
Melbourne, Victoria
Australia

AUSTRIA

Fairchild Semiconductor GmbH
Assmayergasse 60
A-1120 Vienna, Austria
Tel: 43-222-858682
Telex: 115096

BENELUX

Fairchild Semiconductor B.V.
Ruysdaelbaan 35
NL-5613 DX-Eindhoven
The Netherlands
Tel: 31-40-446909
Telex: 51024

ENGLAND

***Fairchild Semiconductor Ltd.**
European Research, Design
and Applications Centre
Shire Hall, Shinfield Park
Reading
Berks. England
Tel: 44-734-752175
Telex: 849317

Fairchild Semiconductor Ltd.
230 High Street
Potters Bar,
Herts. England
Tel: 44-707-51111
Telex: 262835

FRANCE

Fairchild Semiconductor S.A.
12 Place des Etats-Unis
92120 Montrouge (Paris), France
(Mailing Address)
B.P. 655
92542 Montrouge Cedex, France
Tel: 33-1-47466161
Telex: 200614

HONG KONG

Fairchild Semiconductor Products Ltd.
12th floor, Austin Tower
22-26A Austin Avenue
Tsimshatsui
Kowloon, Hong Kong
Tel Nbr: (3) 7235256

ITALY

Fairchild Semiconductor S.p.A
Viale Corsica 7
20133 Milan, Italy
Tel: 39-2-7491271
Telex: 330522

Fairchild Semiconductdor S.p.A.
Via Francesco Saverio Nitti 11
00191 Rome, Italy
Tel: 39-6-328-7548 or 328717
Telex: 612046

JAPAN

Fairchild Japan Corporation
7th Floor Pola Shibuya Bldg.
15-21 Shibuya 1-Choma Shibuya-K
Tokyo 150, Japan
Tel: 81/3/4008351

Fairchild Japan Corporation
Yotsubashi, Cyhuo Building
4-26 Shinmachi 1-Chome
Nichi-Ku, Osaka, 550, Japan
Tel: 81/6/5416138

KOREA

Fairchild Electronics Ltd.
Korea Branch
10th floor, Life Building
61 Yuido-dong, Youngdongpu-ku
Seoul 150, Korea

SINGAPORE

Fairchild Sales Pte. Ltd.
75/77 Bukit Timah Road #03-01/02
Boon Siew Building
Singapore 0922
Republic of Singapore

SWEDEN

Fairchild Semiconductor AB
Bergsundsstrand 39
S-11738 Stockholm, Sweden
Tel: 46-8-840170
Telex: 17759

SWITZERLAND

Fairchild Semiconductor GmbH
Baumackerstr. 46
CH-8050 Zurich, Switzerland
Tel: 41-1-3114230
Telex: 823285

TAIWAN

Fairchild Electronics (Taiwan) Ltd.
Hsietsu Building, Room 502, 5th floor
47 Chung Shan N. Road, Sec. 3
Taipei, Taiwan

WEST GERMANY

Fairchild Semiconductor GmbH
Flughafen Franchtz. Geb. 458
D-6000 Frankfurt/M. 75
West Germany
Tel: 049-69-6905613
Telex: 411829

Fairchild Semiconductor GmbH
Oeltzenstr. 14
D-3000 Hannover, West Germany
Tel: 49-511-17844
Telex: 922922

Fairchild Semiconductor GmbH
Poststrasse 37
D-7250 Leonberg, West Germany
Tel: 049-7152-41026
Telex: 7245711

Fairchild Semiconductor GmbH
Zweigniederlassung Neufahrn
Hanns-Braun-Strasse 50
D-8056 Neufahrn (Munich)
West Germany
Tel: 49-8165-6180
Telex: 526770

*European Design Center

FAIRCHILD

A Schlumberger Company

Advanced Processor
Division

4001 Miranda Ave.
Palo Alto, CA 94304

800 423-5516

In California:
415 858-4249

AD764011.01