INTERGRAPH

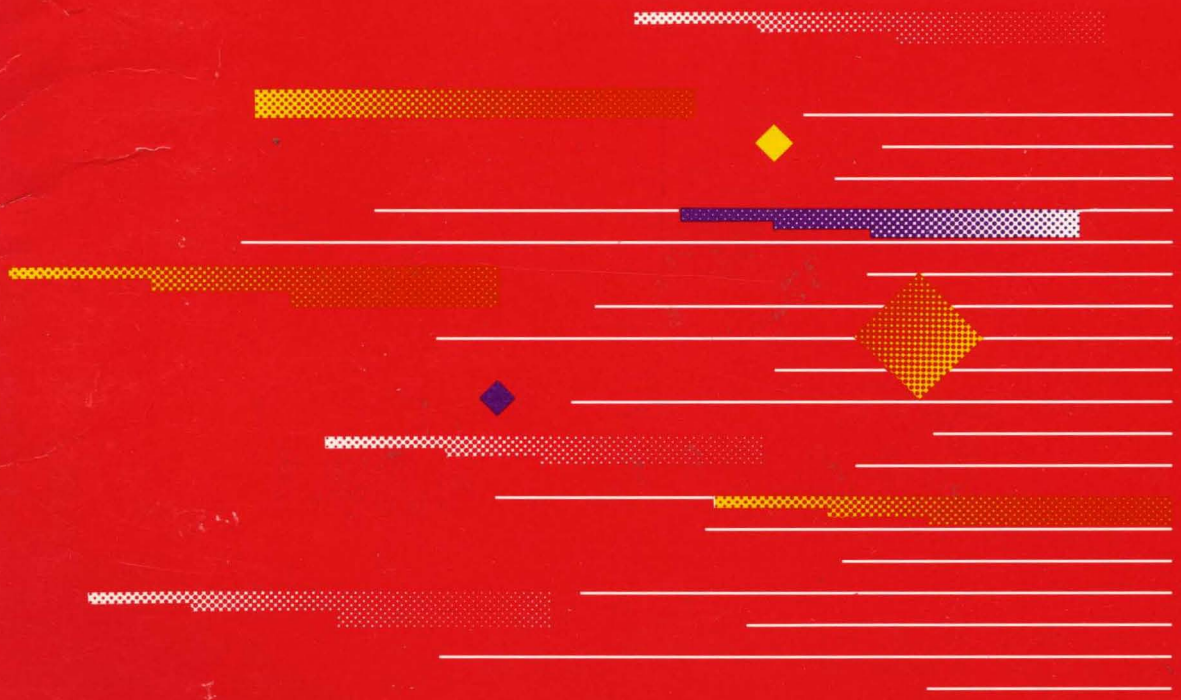
CLIPPER™ C100

32-Bit Compute Engine

Dec. 1987

Advanced Processor Division

Data Sheet



INTERGRAPH

CLIPPER™ C100

32-Bit Compute Engine

Dec. 1987

Advanced Processor Division

Data Sheet

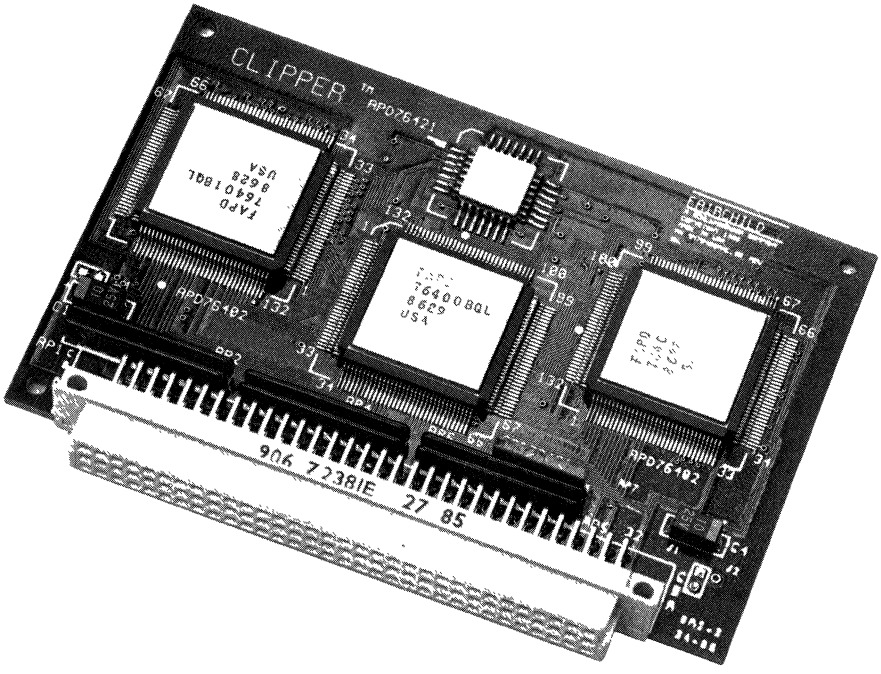
CLIPPER™ C100
32-Bit Compute Engine

Advance Information

Table of Contents

List of Figures

List of Tables



This document provides a comprehensive description of the CLIPPER 32-bit microprocessor module including a functional description, signal description, timing waveforms, module dimensions with connector pinout, and AC/DC parametric values. The CLIPPER module, as shown in the photograph above, contains three CMOS VLSI chips and clock circuitry implemented on a small multilayer printed circuit board.

The three VLSI chips include a CPU/FPU and two combined memory management/cache units, one for data and one for instructions. In addition, the module contains all appropriate bypass capacitors and pull-up resistors. The module interfaces to the CLIPPER bus via the 96 pin connector.

INTERGRAPH

CLIPPER™ C100 32-Bit Compute Engine

Advance Information

Features

High Performance
- 33 MHz single-phase clock
- 33 MIPS peak execution rate
- Separate CPU data and instruction buses
- Full 32-bit internal and external architecture
- 3-stage integer execution pipeline and IEEE floating-point execution unit with overlapped instruction fetch and decode operations
- On-chip IEEE Floating-Point Execution Unit

Streamlined Instruction Set
- 9 addressing modes
- Most frequently used instructions execute in one clock cycle
- Macro instructions for operating system support and optimal use of bus bandwidth
- Multiple programmable register sets for efficiency
  - — 16 32-bit user registers
  - — 16 32-bit supervisor registers
  - — 8 64-bit floating-point registers

8K Byte Total Instruction and Data Caches
- 4 K-byte instruction cache
- 4 K-byte data cache
- 256 line two-way set-associative, 16-byte line size cache organization
- User-enabled instruction prefetch for maximum hit rate and performance of the pipeline
- Bus Watch for system data integrity
- Write-through, copy-back, and noncacheable caching policies on a per-page basis

Memory Management
- Demand paged virtual memory
- 4 G-byte virtual address space per process
- 4 G-byte real memory address space
- Separate user and supervisor modes
- 128 line two-way set-associative Translation Lookaside Buffer each for data and instructions
- Memory read, write, and execute access protection on a per-page basis
- Dynamic Translation Unit and page table update

High-Speed and Flexible Bus
- High-bandwidth synchronous bus
- Byte, halfword, word, and quadword transfers

Interrupt/Exception Processing
- Macro instructions for exception processing
- 256 vectored interrupts with 16 priority levels
- Separate interrupt bus for high-speed interrupt processing
- 18 predefined traps
- 128 system calls

The CLIPPER C100 Module is an architecturally advanced, very high-performance CMOS 32-bit microprocessor compute engine consisting of a CPU, two Cache/MMU chips, and clock control circuitry. The CPU includes an IEEE standard Floating-Point Execution Unit.

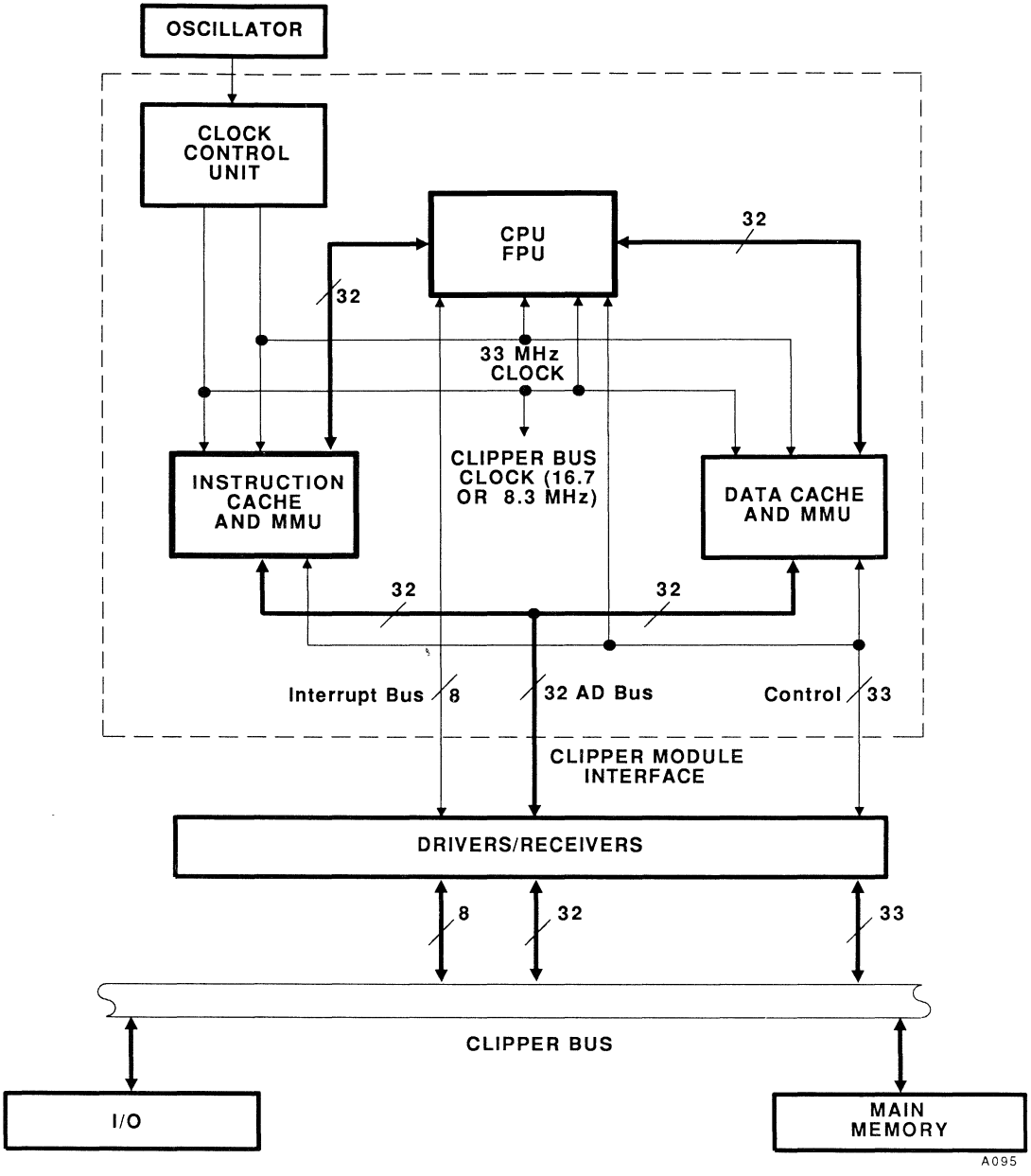
The CLIPPER Compute Engine is a Single Instruction/Single Data architecture with instruction prefetch overlapped on multiple execution units. The basic instruction set is streamlined and hardwired for maximum performance. Because the control section of the CPU is a hardwired logic state machine, rather than a microcoded engine, instructions execute at a maximum rate of one per clock cycle. The CPU contains two 32-bit buses: one for data and one for instructions. Multistage pipelined instruction processing, combined with a dual cache/MMU design, permit concurrency at all stages of program execution. In addition, the integrated Floating-Point Unit executes instructions concurrently with the integer execution unit. A high-bandwidth synchronous bus architecture easily interfaces to high-speed peripherals, I/O, and memory subsystems.

1. Introduction

The CLIPPER C100 32-bit Microprocessor Module (see *Figure 1*) consists of three CMOS VLSI chips together with a Clock Control chip. The VLSI chips are: 1) a high-performance, dual bus CPU/FPU, 2) an instruction cache/MMU chip (I-CAMMU), and 3) a data cache/MMU chip (D-CAMMU). The CLIPPER Module interface is a 96-pin connector which is buffered through a set of user-supplied drivers.

The CLIPPER Module interface signals comprise the CLIPPER Bus and consist of a 32-bit, multiplexed data/address bus, bus arbitration control, bus control, clock control, interrupt control, error signalling, diagnostics, and reset.

Figure 1 CLIPPER C100 Module Block Diagram



1.1. CPU

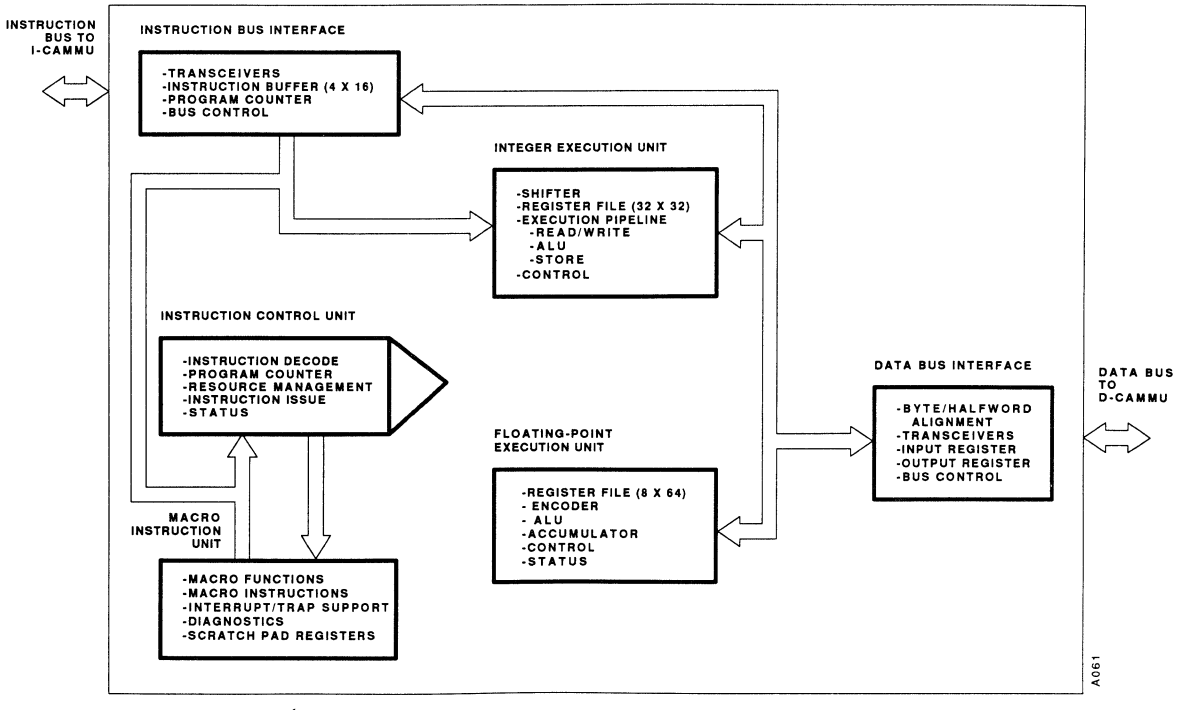
The CLIPPER CPU is a high-performance, full 32-bit internal and external (via separate 32-bit instruction and data buses) processor with a load/store architecture. The CPU is highly pipelined for maximum instruction execution and contains a 32 x 32-bit general register file, two ALUs (one for integer execution and one for floating-point execution), a streamlined instruction set, a Macro Instruction Unit (for exception processing instructions, interrupt handling instructions, and macrocoded instructions), and a complete Floating-Point Unit. *Figures 2* and *3* show simplified and detailed block diagrams of the CPU.

1.1.1. Pipelining and Concurrency

The CPU utilizes a fetch, decode; and execution pipeline as shown in *Figure 4*. The CPU two-stage instruction control unit (see *Figure 2*) supports two instruction execution units that operate in parallel, permitting up to four instructions (three integer and one floating-point) to be in the execution stage concurrently. Instruction control (the upper pipeline) includes both fetch and decode, decode includes both resource management and issue. The parallel execution units (lower pipeline) execute integer and floating-point operations concurrently. Program counter values accompany instructions through the upper pipeline for exception processing and branch control by the CPU.

There are two stages of instruction fetch, namely, from memory to the instruction cache (ahead of actual CPU demand) and from the cache to the CPU's Instruction Buffer. The Instruction Buffer can hold up to four instructions. Immediate values are sent from the Instruction Buffer via the J register to an L register in the ALU

Figure 2 Simplified CPU Block Diagram



pipeline, for use as address offsets or data values. The J register and L register stages are shown in *Figure 3.*

Instruction decode and resource management are performed in the B stage. The B stage obtains instruction parcels from either the Instruction Buffer or the Macro Instruction Unit. Resource management is accomplished by comparing an instruction request for a resource against a table of resources busy.

In the final stage of the upper pipeline (decoded and assembled instruction is in the C stage), instructions are issued for execution to the integer execution unit or the floating-point execution unit if no resource conflict exists.

The lower pipeline consists of two parallel execution units, an integer execution unit and a floating-point execution unit. The integer execution pipeline has three stages. In the first stage, operands are read from the general register file. The general register file has three ports that operate concurrently in a single clock period; two ports are for reading and the third is for writing. Thus, two reads and a write may be performed in a single clock. In the second stage, the ALU output is written to the A register, and in the third stage, the contents of the A register are output to the FPU, the bypass mux (to the ALU), to the general register file or to the D-CAMMU interface.

1.1.2. Integer Execution Unit

The Integer Execution Unit executes all instructions except those handled by the FPU. It contains a register file (supervisor and user sets), a serial double-bit shifter, and a 32-bit Arithmetic Logic Unit (ALU).

The ALU is used for address computation as well as data manipulation. Nine addressing modes are supported, each requiring only one pass through the ALU.

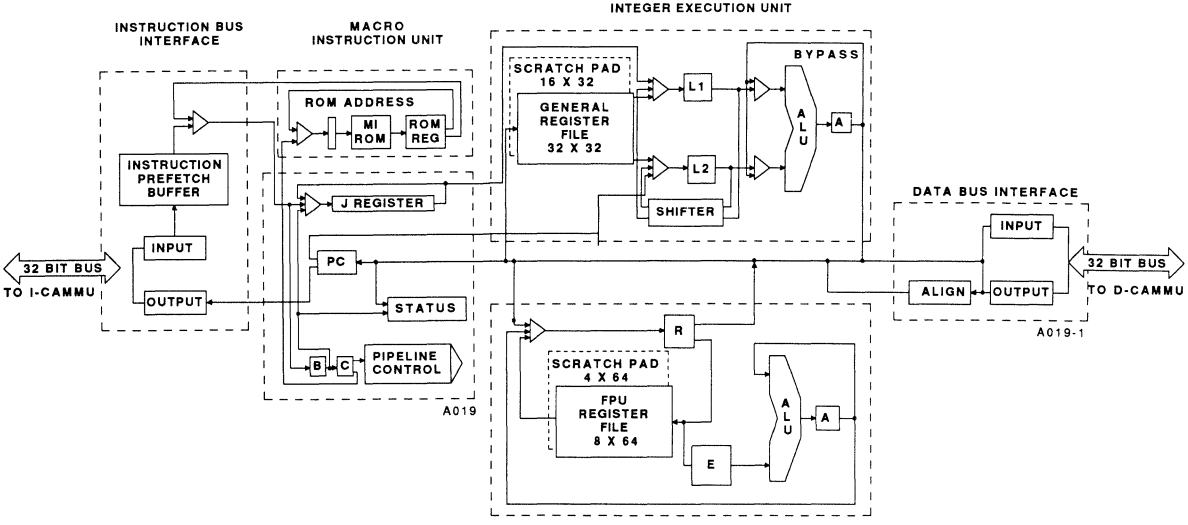
When the result output by the ALU is needed by the instruction immediately following it, a feedback mechanism allows the result from the current ALU operation to be fed back into the ALU for the next operation.

1.1.3. Floating-Point Execution Unit (FPU)

An integrated 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. These registers are accessible to either the user or supervisor. Because the Floating-Point Unit is on the CPU chip, CLIPPER Bus accesses are usually not required. This reduces bus traffic and improves performance.

All CLIPPER floating-point arithmetic operations support the IEEE 754 Standard. For more information, refer to *Section 4, Floating-Point Unit.*

Figure 3 Detailed CPU Block Diagram



1.1.4. Macro Instruction Unit

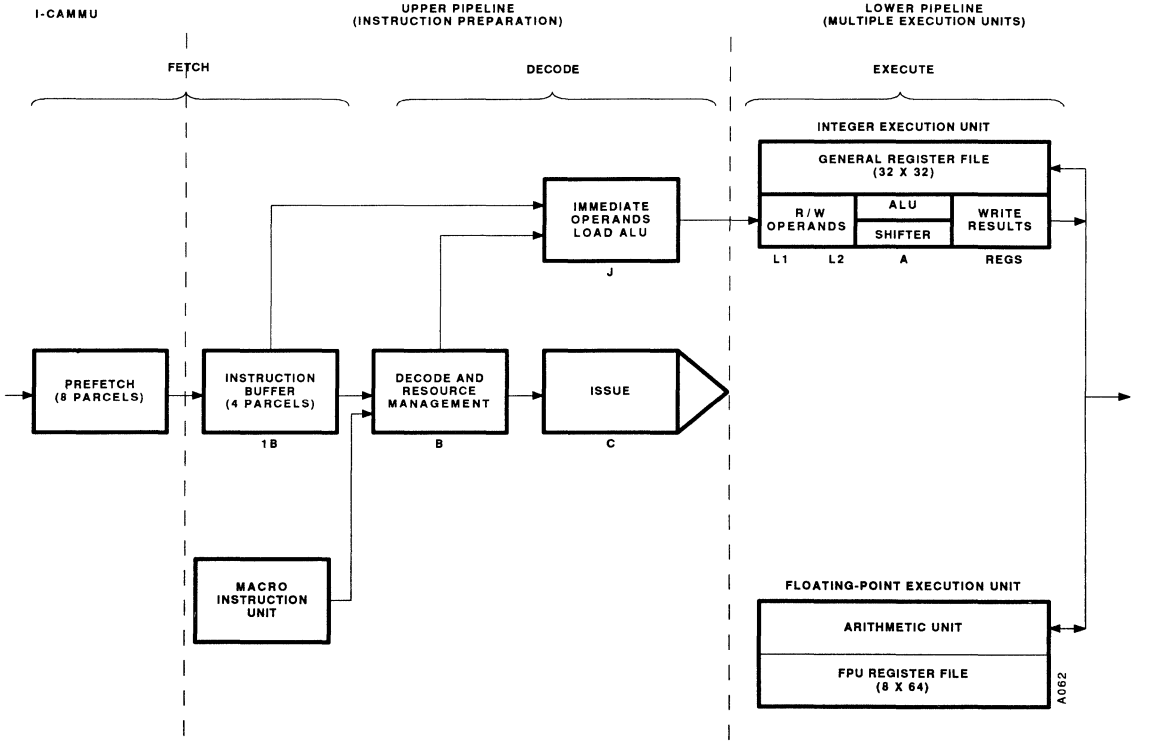
The Macro Instruction (MI ROM) Unit stores instruction sequences of the basic hardwired instruction set. When a macro instruction is encountered in the instruction stream, an instruction sequence is read from the MI ROM and inserted into the B stage of the upper pipeline. The width of the ROM word is such that the instruction pipeline can be maintained at the maximum of one parcel (one halfword) every clock. When the MI sequence ends, the instruction stream is switched back to the Instruction Buffer as the source.

The MI Unit provides three types of instruction sequences:

— Those that provide direct support for the operating system: for example, context switching and trap/interrupt entry and return;

— Those that perform certain floating-point operations not directly implemented in the Floating-Point Unit: for example, single to double and double to single-precision floating-point conversions;

Figure 4 CLIPPER Pipeline



— Commonly used complex instructions which are typically found in so-called "complex instruction set computers:" for example, character string manipulations.

Instructions from the MI ROM are provided with additional MI register files, thus avoiding resource conflicts with the floating-point and general-purpose registers.

1.2. CAMMU

In addition to the CPU, the CLIPPER Module includes two Cache/Memory Management Unit (CAMMU) chips, an Instruction Cache/MMU, and a Data Cache/MMU. The CAMMUs interface to the CPU via a high-speed, 32-bit internal module bus and interface to main memory and I/O devices via the CLIPPER Bus.

1.2.1. Instruction and Data Caches

Two separate, 4 K-byte cache memories, one for data and one for instructions, act as transparent high-speed buffers between the CPU and main memory. Each cache is two-way set-associative, containing 256 quadword lines of frequently used instructions or data. For fast CPU access, each cache also contains a virtual address cache consisting of a 16-byte buffer containing the quadword that was most recently accessed from the cache, and a register containing the virtual address of the quadword.

Because most CPU fetches are satisfied directly from the cache, the access time of real memory has far less effect on total system performance. Minimizing fetches from main memory also reduces bus traffic and allows greater bandwidth for other bus masters or I/O processors.

Bus Watch is the monitoring of the CLIPPER Bus transactions by the CAMMUs. It is used to ensure data consistency between the cache and main memory, and to ensure that the latest data is always transferred to an I/O device reading main memory. Bus Watch is transparent to software.

A demand fetch algorithm is implemented in both the I-CAMMU and D-CAMMU. A prefetch algorithm is also implemented in the I-CAMMU; it can be enabled or disabled under program control.

1.2.2. Memory Management Unit (MMU)

The Memory Management Unit translates CPU virtual addresses to real addresses in one of three separate real spaces (I/O, Boot, or Main Memory) using translation tables located in main memory. In order to minimize the time required to obtain these translations, an additional two-way set-associative Translation Lookaside Buffer (TLB) in each CAMMU holds 128 of the most frequently used values from the translation tables for both instructions and data.

When the TLB does not contain the required translation entry, the MMU fetches the required value from main memory and updates the TLB.

The MMU also supports main memory access protection (read, write, and execute).

1.3. Clock Control Unit

The CLIPPER clock chip provides two clock signals. MCLK is an internal clock not available to the user. The MCLK frequency is the rate of operation of the CPU and CAMMUs. BCLK is the CLIPPER Module bus clock. The BCLK frequency is the rate of operation of the CLIPPER bus. With an externally supplied 66.7 MHz oscillator, MCLK is 33.3 MHz, and BCLK is either 16.7 MHz or 8.3 MHz depending on the state of the RATE control pin on the CLIPPER Bus. See *Section 9, CLIPPER Bus*, for details.

2. Memory Organization

The real memory of a CLIPPER system is organized as a sequence of 32-bit words, each word consisting of four 8-bit bytes. Each byte is assigned a unique address ranging from 0 to 4,294,967,295 (4 G-bytes).

By using virtual memory techniques, a CLIPPER system can appear to have a full 4 G-bytes of physical memory available to each user program. See *Section 9, CLIPPER Bus*, for details.

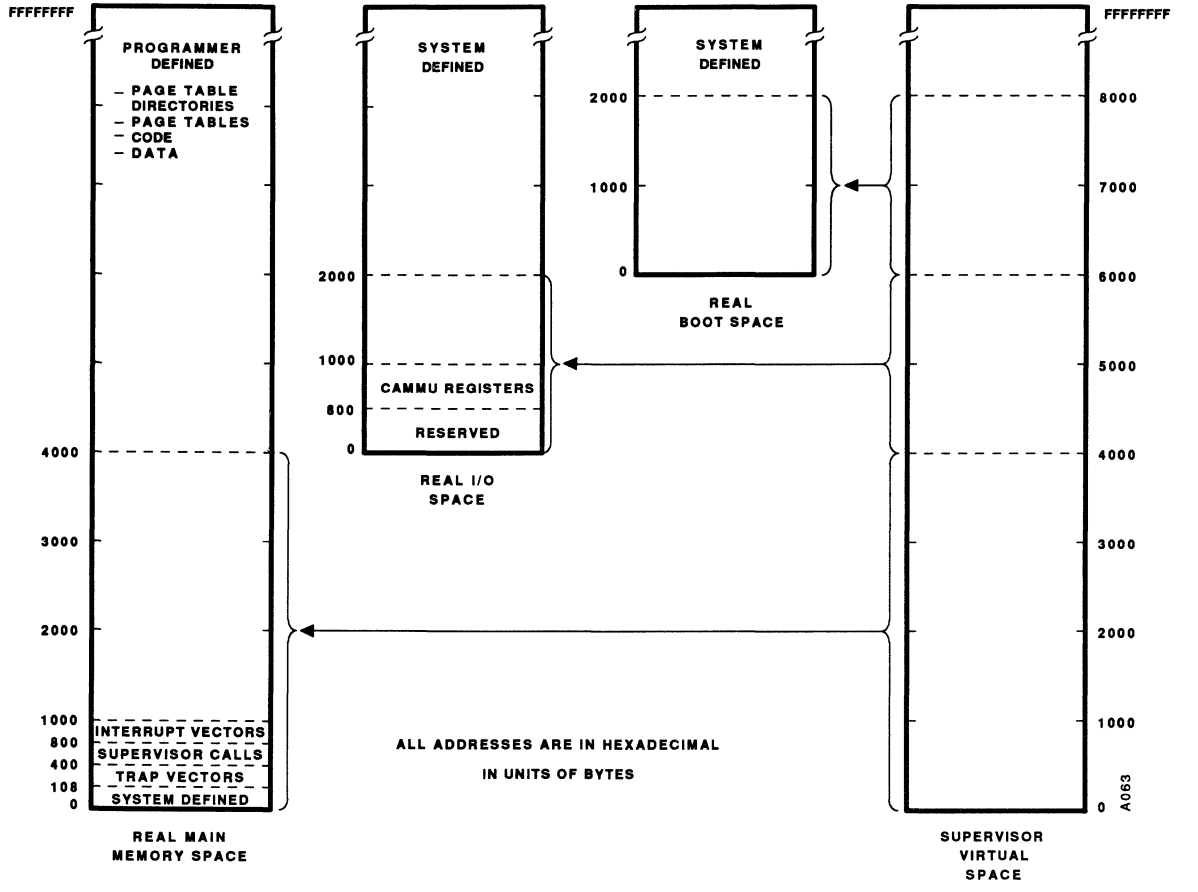
There are three real address spaces defined in the CLIPPER architecture:

— Main memory space
— I/O space
— Boot space

Main memory, I/O space, and Boot space are accessible in both user and supervisor modes. The memory space accessed by a given address is determined by the System Tag associated with the page.

The Hardwired Translation Lookaside Buffer (HTLB) is a feature of the CAMMU which guarantees TLB hits of special memory pages by the supervisor. The first four pages of real main memory space have HTLB entries in the CAMMUs, as do the first two pages of both I/O space and Boot space. The HTLB is used in supervisor mode only. The HTLB is described in detail in *Section 7.2.2, Fixed Address Translation*. CLIPPER's three memory spaces and the mapping of the HTLB are shown in *Figure 5*.

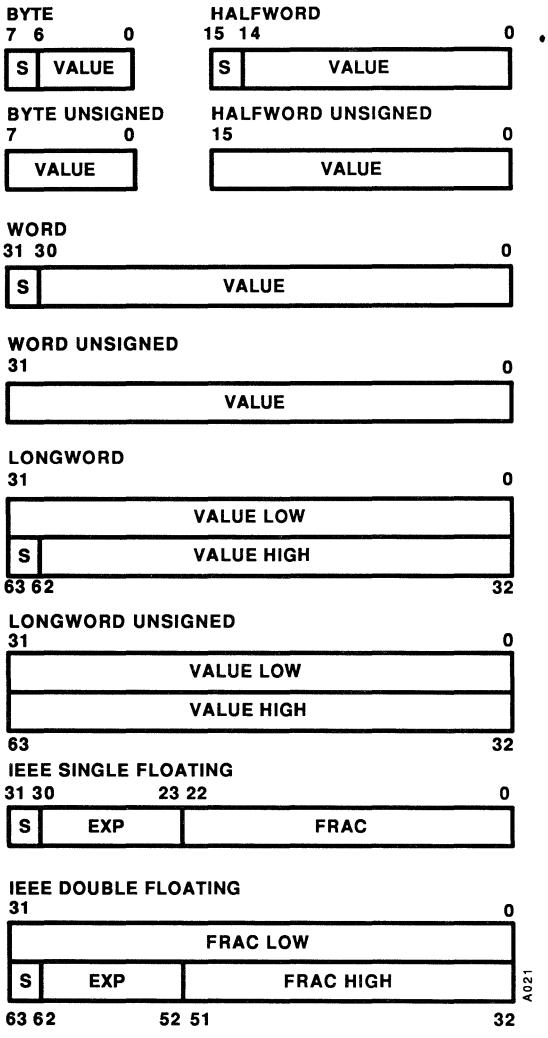
Figure 5 Real Address Spaces—HTLB Mapping



2.1. Data Types

The CLIPPER architecture supports the primitive data types shown in *Figure 6*. There are signed and unsigned bytes, halfwords (16 bits), words (32 bits), and longwords (64 bits), as well as single-precision (32-bit) and double-precision (64-bit) IEEE Standard floating-point numbers.

Figure 6 CLIPPER Primitive Data Types



The rules for the storage and alignment of memory data types are illustrated in *Figure 7* and summarized below:

1. Bit 0 is the least-significant bit (LSB) of all data types. Bit numbers increase from right to left.
2. The least-significant byte of multiple-byte data types is stored at the lowest memory address.
3. The most-significant byte of multiple-byte data types is stored at the highest memory address.
4. All data types must begin at an address that is a multiple of their size. For example, a halfword must begin on a halfword boundary.

3. Programming Model

The basic programming model for the CLIPPER Compute Engine is shown in *Figure 8*. CPU registers are discussed in this section; CAMMU registers are discussed in *Section 7, Cache and MMU*.

Figure 7 Addressing and Alignment of Data in Memory

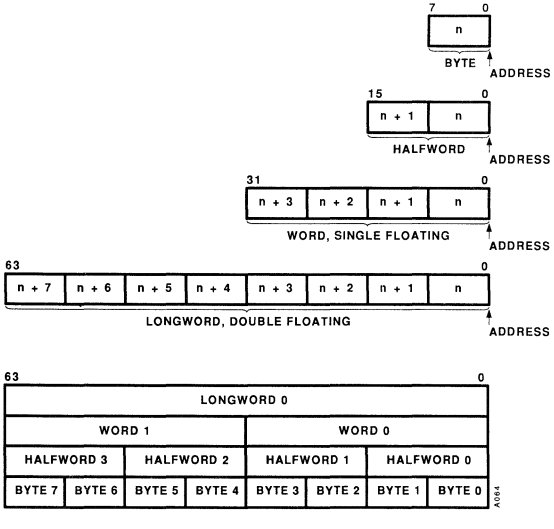
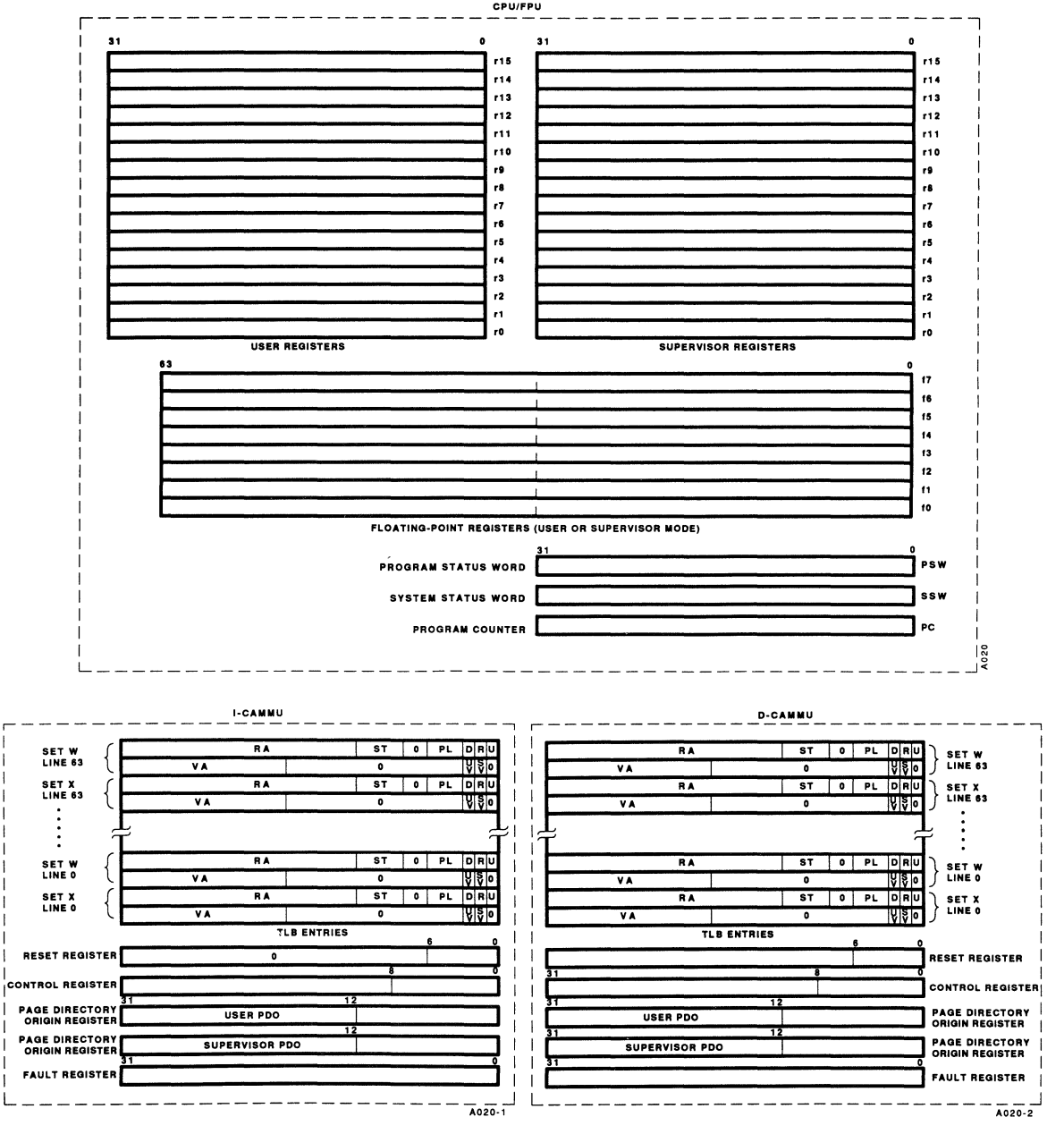


Figure 8 CLIPPER Programming Model



3.1. Register Sets

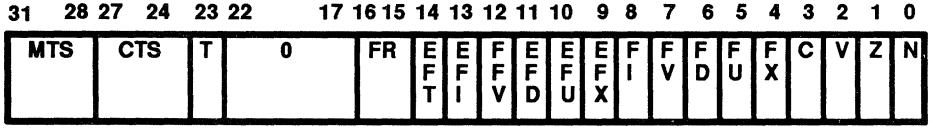
The CPU contains three sets of registers: 16 user registers (r0-r15), 16 supervisor registers (r0-r15) and 8 floating-point registers (f0- f7).

The user and floating-point registers are accessible in both user and supervisor modes; the supervisor registers are accessible only in supervisor mode.

There are two status and control registers: the Program Status Word (PSW) and the System Status Word (SSW). The PSW is accessible in both user and supervisor modes. The SSW is writable only in supervisor mode; it should not be read in user mode to ensure compatibility with future Intergraph CLIPPER products.

The Program Counter (PC) contains the address of the current instruction. This is interpreted as a virtual address if CLIPPER is operating in mapped mode, and as a physical address if CLIPPER is operating in unmapped mode (see *Section 3.3, Mapped and Unmapped Addresses).* The PC is accessible by both the user and the supervisor.

Figure 9 Program Status Word



FIELD	MEANING
N	Negative
Z	Zero
V	Overflow
C	Carry out or borrow in
FX	Floating inexact
FU	Floating underflow
FD	Floating divide by zero
FV	Floating overflow
FI	Floating invalid operation
EFX	Enable floating inexact trap
EFU	Enable floating underflow trap
EFD	Enable floating divide by zero trap
EFV	Enable floating overflow trap
EFI	Enable floating invalid operation trap
EFT	Enable floating trap
FR	Floating rounding mode
T	Trace trap
CTS	CPU trap status
MTS	Memory trap status

FR	MEANING
0	Round to nearest
1	Round toward + infinity
2	Round toward - infinity
3	Round toward zero

CTS	MEANING
0	No CPU trap
1	(Reserved)
2	Divide by zero
3	(Reserved)
4	Illegal operation
5	Privileged instruction
6	(Reserved)
7	Trace trap
8-15	(Reserved)

MTS	MEANING
0	No memory trap
1	Corrected memory error
2	Uncorrectable memory error
3	(Reserved)
4	(Reserved)
5	Page fault
6	Read or execute protect fault
7	Write protect fault
8-15	(Reserved)

A065

3.1.1. User and Supervisor Registers

The user and supervisor registers, r0-r15, are general-purpose, 32-bit registers. They are used for word (32-bit) and longword (64-bit) integer operations.

Bytes and halfwords used in load and store instructions are sign- or zero-extended to 32 bits before they are put in registers. Longword values are stored in register pairs, with the least significant word in the even-numbered register. When double-precision (64-bit) floating-point data types are moved to an integer register pair, the least-significant fraction occupies the even numbered register.

3.1.2. Floating-Point Registers

The floating-point registers, f0-f7, are used by the FPU for floating-point instructions, which are executed concurrently with instructions in the ALU. These 64-bit registers are used for floating-point operands in both single- and double-precision IEEE format. Single-precision operands stored in floating-point registers have zeros in the 29 least significant fraction bits and in the three most significant exponent bits.

The integer multiply, divide, and mod instructions are executed by the FPU, but use registers r0-r15 (user or supervisor).

3.1.3. Program Status Word (PSW)

The PSW, shown in *Figure 9*, contains flags which identify and together with the SSW, control a program's response to various exceptions resulting from integer and floating-point operations (see *Section 6, Exceptions*, for more details).

On hardware reset, the trace trap (T) flag is cleared; the remaining PSW bits are undefined.

C,V,Z,N: Condition Codes
The condition codes are modified only by the register-to-register integer instructions, string instructions, floating comparison, and by directly writing the PSW. They are tested by the **branch on condition** instruction.

FX, FU, FD, FV, FI: Floating-Point Exception Flags
The floating-point exception flags are set by hardware on exceptions arising from floating-point operations, in accordance with the IEEE 754 Floating-Point Standard. Once set, they are cleared only by user software or, for those conditions for which the corresponding trap is enabled (i.e., when both EFT and the individual enable flag are set) by the trap handler. They are tested by the **branch on floating exception** instruction (see *Section 6.2.2, Floating-Point Arithmetic Trap Group*, for more details).

EFX, EFU, EFD, EFV, EFI: Enable Floating Flags
The IEEE floating-point trap enable flags are set by software to request the result that would be given to a trap handler on an exception, rather than the IEEE default (no-trap) result. If the EFT bit is set, enabled exceptions also cause traps. See *Section 6.2.2, Floating-Point Arithmetic Trap Group*, for a description of the use of this field by trap handler routines.

EFT: Enable Floating Trap
When set, the enable floating trap flag enables traps to occur whenever an exception is signalled by the FPU and that exception's trap enable flag is also set. When this bit is clear, floating-point traps are disabled and program execution continues regardless of the values of the floating trap enable flags.

FR: Floating Rounding Mode
The floating-point rounding mode field is set by software to select the IEEE rounding mode for floating-point operations.

The default is round to nearest, in which the rounded result is the closest representable number to the exact result, with ties decided in favor of the representable number with zero as its least-significant fraction field bit.

When rounding toward + ∞, the result is the format's value (possibly + ∞) closest to and no less than the infinitely precise result. When rounding toward - ∞, the result is the format's value (possibly - ∞) closest to and no greater than the infinitely precise result. When rounding toward 0, the result is the format's value closest to and no greater in magnitude than the infinitely precise result.

T: Trace Trap Enable
The trace trap enable flag is set by the user or supervisor to request a trace trap following execution of the

next instruction. It is cleared by the user or supervisor to disable the trace.

CTS: CPU Trap Status
The CPU trap status field is set by the hardware to indicate the cause of a CPU-related trap (see *Section 6.2, Traps).*

MTS: Memory Trap Status
The memory trap status field is set by the hardware to indicate the cause of a memory-related trap (see *Section 6.2, Traps).*

3.1.4. System Status Word (SSW)

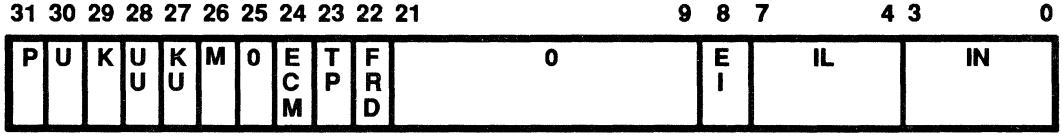
The SSW controls the CLIPPER Module's mode of operation (user or supervisor) and provides status and control for program protection and the response to interrupts (see *Figure 10*). It may be written in supervisor mode only. Reset clears the following SSW flags: EI, TP, M, U, K, KU, UU and P. The remaining flags are undefined. This represents unmapped supervisor mode with all maskable interrupts disabled.

The SSW is written using the **movwp** (move word to processor register) instruction. When used with the SSW, this instruction can take as its second operand either processor register 1 (ssw) or processor register 3 (sswf). **movwp** using processor register 1 behaves like a branch instruction, causing the upper pipeline to be flushed. **movwp** with processor register 3 does not cause the pipeline to be flushed, is thus faster, but must only be used in cases where the modification of the SSW will not compromise the memory mapping of the subsequent code in the upper pipeline. That is, because the K, U, KU, and UU protection bits are compared with the PL field of the TLB or HTLB entry for memory access protection, a memory reference that would have failed may succeed or vice versa, or it may fail differently, or it may succeed for the wrong reasons. Therefore, processor register 3 may only be used when modifying the IN, IL, EI, FRD, TP, ECM, KU and UU flags; modifications of the M, K, U and P flags must use the **movwp** instruction with processor register 1.

Descriptions of IN, IL, EI, TP, and ECM are given below and in *Section 6, Exceptions.* M, KU, UU, K, U, and P are described below and in *Section 7.2.1, Translation Lookaside Buffer* (see Protection Level field description).

IN: Interrupt Number
The interrupt number field is set by hardware (INTRAP and **retl**) and by software (**movwp**) to indicate the

Figure 10 System Status Word



FIELD	MEANING
IN	Interrupt number
IL	Interrupt level
EI	Enable interrupts
FRD	Floating registers dirty
TP	Trace trap pending
ECM	Enable corrected memory error

FIELD	MEANING
M	Mapped mode
KU	User protect key
UU	User data mode
K	Protect key
U	User mode
P	Previous mode

system's current interrupt number. If an interrupt of equal or higher priority occurs during the service of an interrupt, this value (along with the interrupt level) will be pushed on the stack, and this field will be updated with a new interrupt number value. Interrupt numbers are not prioritized.

IL: Interrupt Level
The interrupt level field is set by hardware (INTRAP and **reti**) and by software (**movwp**) to establish the system's current interrupt priority level. Only interrupts of equal or higher priority (equal or lesser value) than this value will be recognized. If an interrupt of equal or higher priority occurs during an interrupt service routine, this value will be pushed on the stack, and this field will be updated with the new interrupt level.

EI: Enable Interrupts
The enable interrupt flag is set by software to enable interrupts. It is cleared by software to disable interrupts.

FRD: Floating Registers Dirty
The floating register dirty flag is set by hardware whenever a floating-point register is written. This flag may be cleared by software. Its purpose is to permit operating systems to reduce context switching overhead.

TP: Trace Trap Pending
The trace trap pending flag is automatically set by hardware whenever a trap or interrupt occurs during an instruction and the T flag is set. This ensures that the trace trap is taken immediately after the trap or interrupt handler has finished executing, and that a single instruction is traced exactly once.

On data page faults, the supervisor must clear TP before restarting the faulting instruction in order to ensure that the instruction is traced exactly once.

ECM: Enable Corrected Memory Error Trap
The enable corrected memory flag is set by software to request a trap whenever a corrected memory error occurs. When this flag is set, a logic low on the $\overline{\text{MSBE/RETRY}}$ signal line (indicating a single-bit memory error) causes a trap.

M: Mapped Mode
The mapped mode flag is set by software to cause all address references to be mapped through the page tables. When set, virtual address (VA) to real address (RA) translation by the CAMMUs is enabled (mapped mode). When cleared, VA to RA translation by the CAMMUs is disabled (unmapped mode). The only exceptions are the first eight pages when in supervisor mode. These pages are always mapped via the HTLB, regardless of the state of this flag.

U:User Mode
The user mode flag is set by the supervisor to indicate user mode of operation and cleared to indicate supervisor mode of operation.

K: Protect Key
The protect key flag is set and cleared by the supervisor to select one of two sets of memory access protection codes for memory access validation and protection during program execution. This flag is used for the access protection code selection in user mode, and in supervisor mode when the UU flag is clear (see *Table 10* in *Section 7.2, Memory Management Unit*).

KU: User Protect Key
The user protect key flag is set and cleared by the supervisor program to select one of two sets of memory access protection codes for memory access validation and protection during program execution. This flag is used for the access protection code selection only during supervisor program execution when the UU flag is set (see *Table 10* in *Section 7.2, Memory Management Unit*).

UU: User Data Mode
The user data mode flag is set and cleared by the supervisor to select either supervisor or user data address space access when memory data is referenced in supervisor mode, and to select either the K or KU key flags for selection of the access protection codes used during supervisor memory references. When the UU flag is set, supervisor data references access user data space, and the KU flag is used for access protection code selection. When the UU flag is clear, supervisor data referen-

ces access supervisor data space, and the K flag is used for access protection code selection. This flag is significant only in supervisor mode (see *Table 10* in *Section 7.2, Memory Management Unit*).

P: Previous Mode
The previous mode flag is copied from the U flag whenever the INTRAP sequence is executed.

3.2. Supervisor and User Modes of Operation

The CLIPPER Module has two modes of operation, user and supervisor, as selected by the SSW's U flag. User and supervisor modes are distinguished by the set of instructions which they are permitted to execute, and by the registers and logical address space they are permitted to access.

All instructions can be executed in supervisor mode. Instructions which can be executed only in supervisor mode are called privileged instructions. When a program in user mode attempts to execute these instructions, a privileged instruction trap occurs.

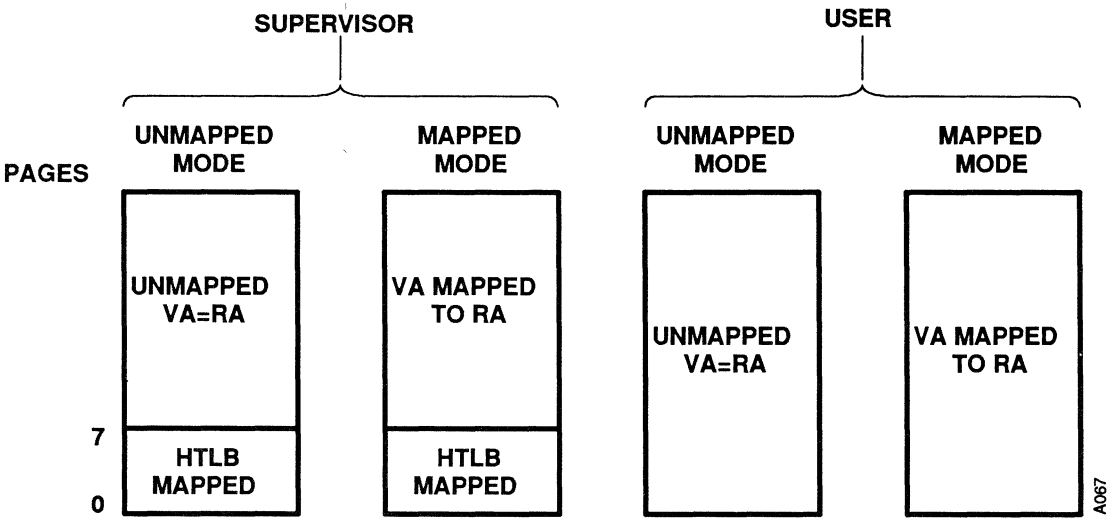
Programs executing in user mode have access only to the user registers (r0-r15), floating-point registers (f0-f7), the PSW, and the PC. Supervisor mode programs have access to all programmer-accessible registers. Memory address spaces are distinct for user and supervisor modes. Different translation tables may be used for translating user and supervisor mode addresses, and the access protection provided by the memory management mechanism allows access by supervisor programs to memory locations inaccessible to user mode programs.

Supervisor mode is entered only via the INTRAP sequence, or when the system is reset.

3.3. Mapped and Unmapped Addresses

CLIPPER can operate in two modes: mapped and unmapped. In mapped mode, the CAMMU translates user and supervisor virtual addresses to real addresses using the TLB or the HTLB (for supervisor virtual addresses 0 - 777F Hex); in unmapped mode, only the HTLB is used for translation. The mode is selected by the M (mapped mode) flag in the SSW. When this flag is set, CLIPPER operates in mapped mode; when this flag is clear, CLIPPER operates in unmapped mode. The two modes are shown in *Figure 11*. Virtual to real address translation is discussed in *Section 7.2, Memory Management Unit.*.

Figure 11 Address Mapping—Mapped/Unmapped Modes



4. Floating-Point Unit

The CLIPPER Floating-Point Unit (FPU) executes addition, subtraction, multiplication, and division operations conforming to the IEEE Standard for Binary Floating-Point Arithmetic (ANSI/IEEE Std 754-1985) in the single- and double-precision formats. In addition, the floating-point comparison operations are provided for both precisions. The floating-point execution unit also performs integer multiplication, division, and mod operations.

Comparisons of floating-point numbers can result in the familiar trichotomy of b < a, b = a, b < a, as well as the condition b .unordered. a, which arises when either b or a is a non-numeric value (NaN). Results of the comparison are indicated in the PSW condition codes at the conclusion of a floating-point comparison. Conditional branch instructions allow these condition codes to be used.

The floating-point execution unit performs one operation at a time, reusing internal resources over a number of CPU clocks in order to complete the operation, including the handling of special case operands and results mentioned below.

4.1. Floating-Point Register Usage

All of the floating-point arithmetic instructions are register-to-register operations, using the eight floating-point registers available to software. These registers are capable of holding either single or double format operands interchangeably. The floating-point registers may be directly loaded from memory or may be loaded by transfer from the integer register file. Storing of operands may be direct to memory or by transfer to the integer register file. Additional "scratch pad" registers, transparent to the user, are available to the Macro Instruction Unit.

Integer multiplication, division, and mod operations are also register-to-register, but in this case the registers used are in the integer register file; no floating-point registers are involved.

Because separate instructions are provided for single- and double-precision operations, a rounding precision mode field is not required because the precision is defined by the instruction field. All four rounding modes called for in the Standard are provided by the FR field in the PSW.

4.2. Floating-Point Exceptions and Traps

Exceptional operands and operation results are handled in conformity with the requirements of the IEEE Standard. The special operands include signalling and quiet Not-a-Number (NaN), signed infinities, signed zeros, and denormal numbers, as well as the wealth of ordinary normalized numbers.

If the corresponding trap enable flag in the PSW is set, and the PSW's floating-point trap group enable flag is also set, then a floating-point trap occurs. The CPU then invokes a program called a trap handler, which may be user-specified. When a trap handler is entered, the **load floating status (loadfs)** instruction can be executed to acquire useful information about the instruction causing the exception. Floating-point exceptions are discussed in greater detail in *Section 6, Exceptions*.

4.3. FPU Software Initialization

The IEEE Standard requires the following initial conditions:

— The rounding mode must be round nearest.

— The floating-point exception flags must all be cleared.

— All floating-point traps must be disabled, and default results for all exceptions must be enabled.

This initialized state is accomplished by clearing all FPU-related bits in the PSW.

The contents of f0-f7 should be set to a known value. Some programming languages require that these values be initialized to zero. The IEEE Standard, on the other hand, provides for special reserved values and calls these NaN, or Not-A-Number. Whichever of these is chosen, this value should be created and loaded into each of the floating-point registers.

An example FPU initialization is as follows:

```
loadq   $0, r0              # Create zero
movwp   r0, psw             # Load PSW with rounding
                            # mode 00 (nearest) and
                            # clear all exception
                            # flags and trap enable bits
loadi   $0x7ffbad75, r1     # Load high half of hex
                            # NaN 1.bad75a
loadi   $0x00000000, r0     # Load low half of NaN
movld   r0, f0              # Store in floating register 0
movld   r0, f1              # Store in floating register 1
movld   r0, f2              # Store in floating register 2
movld   r0, f3              # Store in floating register 3
movld   r0, f4              # Store in floating register 4
movld   r0, f5              # Store in floating register 5
movld   r0, f6              # Store in floating register 6
movld   r0, f7              # Store in floating register 7
```

The NaN used in the initialization above is a quiet NaN. A quiet NaN propagates through arithmetic operations unchanged, except for the sign bit, which is undefined for NaNs. Thus, any user who operates on a register not yet defined will receive this NaN as a result.

5. Instruction Set

The CLIPPER instruction set of 101 basic and 67 macro instructions is streamlined for speed and the most effective use of the system's resources and register sets. This smaller, faster instruction set is especially useful to high-level language compilers that optimize register usage, branch timing for maximum speed, and pipeline sequencing.

Memory access is by load/store instructions to minimize memory-dependent execution delays. All data operations are performed on registers by hardwired instructions.

There are two units in the CLIPPER CPU that execute instructions: the Integer Execution Unit (IEU) and the Floating-Point Execution Unit (FPU). The integer instructions (with the exception of integer multiplies and divides) are executed by the IEU. Floating-point instructions (and the integer multiplies and divides) are executed by the FPU.

Most instructions are fetched from main memory. Each instruction is fetched (through the instruction cache), decoded, then executed, either by the IEU or by the FPU. The only exceptions are the macro instructions.

A macro instruction opcode selects a sequence of instructions in the macro instruction ROM (MI ROM). When a macro instruction is decoded, execution control is switched to the MI ROM, and the sequences of the macro instruction are executed.

The instruction set is listed in *Table 1.*

5.1. Instruction Formats

The information encoded in each instruction specifies the operation to be performed, the type of operands to use (if any), and the location of the operands. The mnemonic and operands of the assembly language source statement determine the instruction format used.

Most instructions require one or more operands in the source statement. These operands can be located in a register or in memory. For example, the **loadb** instruction contains operands that reference memory and a register. If an operand is located in memory, the instruction must calculate the address of the operand according to the address mode specified in the instruction format.

An operand can also be encoded within the instruction. The immediate and quick instructions use this type of format for efficient operation.

All instructions are constructed in multiples of halfwords called *parcels* (see the general instruction format below). Depending on the instruction format used, the size of an instruction varies from one to four parcels.

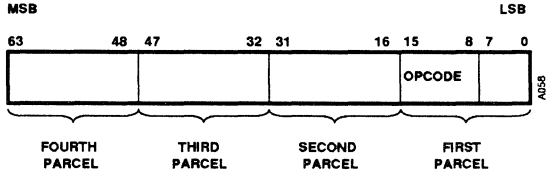
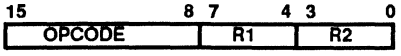


Figure 12 shows CLIPPER instruction formats. Notice that the formats are divided into two main categories, non-memory referencing instructions (NO ADDRESS) and memory referencing instructions (WITH ADDRESS).

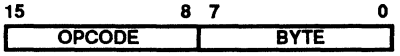
Figure 12 Instruction Formats

INSTRUCTION FORMATS - NO ADDRESS

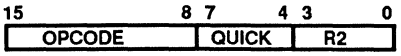
REGISTER



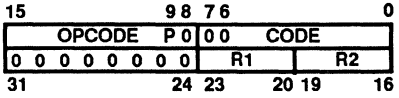
CONTROL



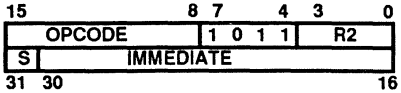
QUICK



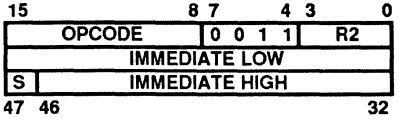
MACRO



16-BIT IMMEDIATE

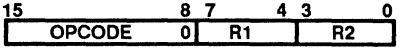


32-BIT IMMEDIATE

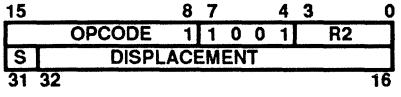


INSTRUCTION FORMAT - WITH ADDRESS

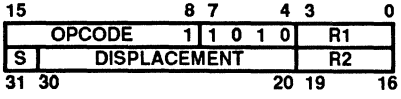
RELATIVE



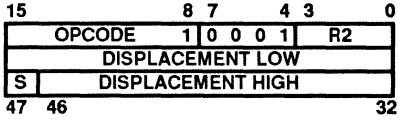
PC-RELATIVE PLUS 16-BIT DISPLACEMENT



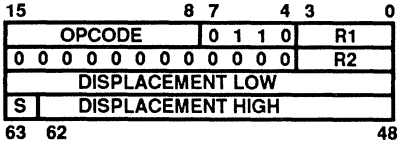
RELATIVE PLUS 12-BIT DISPLACEMENT



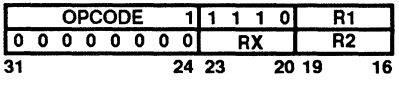
PC-RELATIVE PLUS 32-BIT DISPLACEMENT



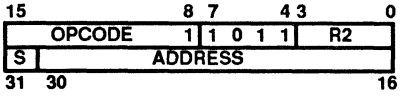
RELATIVE PLUS 32-BIT DISPLACEMENT



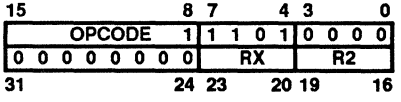
RELATIVE INDEXED



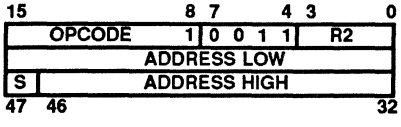
16-BIT ABSOLUTE



PC INDEXED



32-BIT ABSOLUTE



A022

Table 1 Instruction Set, by Function

LOAD/STORE INSTRUCTIONS
Load Address
Load Byte
Load Byte Unsigned
Load Double Floating
Load Floating Status
Load Halfword
Load Halfword Unsigned
Load Immediate
Load Quick
Load Single Floating
Load Word
Store Byte
Store Double Floating
Store Halfword
Store Single Floating
Store Word

DATA MOVEMENT INSTRUCTIONS
Move Double Floating
Move Double to Longword
Move Longword to Double
Move Processor Register to Word
Move Single Floating
Move Supervisor to User (privileged)
Move Single to Word
Move User to Supervisor (privileged)
Move Word
Move Word to Processor Register
Move Word to Single Floating

ARITHMETIC INSTRUCTIONS
Add Double Floating
Add Immediate
Add Quick
Add Single Floating
Add Word
Add Word with Carry
Subtract Double Floating
Subtract Immediate
Subtract Single Floating
Subtract Word
Subtract Word with Carry
Multiply Double Floating
Multiply Single Floating
Multiply Word
Multiply Word Unsigned
Multiply Word Extended
Divide Double Floating
Divide Single Floating
Divide Word
Divide Word Unsigned
Negate Double Floating
Negate Single Floating
Negate Word
Modulus Word
Modulus Word Unsigned
Scale by, Double Floating
Scale by, Single Floating

Table 1 Instruction Set, by Function (cont.)

LOGICAL INSTRUCTIONS

AND Immediate
AND Word

OR Immediate
OR Word

Exclusive-OR Immediate
Exclusive-OR Word

Not Quick
Not Word

CHARACTER STRING INSTRUCTIONS

Compare Characters

Initialize Characters

Move Characters

CONVERSION INSTRUCTIONS

Convert Double to Single
Convert Double to Word
Convert Rounding Double to Word
Convert Rounding Single to Word
Convert Single to Double
Convert Truncating Double to Word
Convert Truncating Single to Word
Convert Word to Double
Convert Word to Single

COMPARE AND TEST INSTRUCTIONS

Compare Double Floating
Compare Immediate
Compare Quick
Compare Single Floating
Compare Word

Test and Set

SHIFT/ROTATE INSTRUCTION

Shift Arithmetic Immediate
Shift Arithmetic Longword
Shift Arithmetic Longword Immediate
Shift Arithmetic Word
Shift Logical Immediate
Shift Logical Longword
Shift Logical Longword Immediate
Shift Logical Word

Rotate Immediate
Rotate Longword
Rotate Longword Immediate
Rotate Word

STACK MANIPULATION INSTRUCTIONS

Pop Word
Push Word

Restore Registers fn-f7
Restore User Registers (privileged)
Restore Register rn-r14

Save Registers fn-f7
Save User Registers (privileged)
Save Registers rn-r14

CONTROL INSTRUCTIONS

Branch on Condition
Branch on Floating Exception

Call Subroutine
Call Supervisor

No Operation

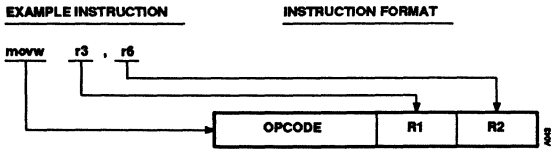
Return From Subroutine
Return From Interrupt (privileged)

Trap on Floating Unordered

Wait for Interrupt (privileged)

5.1.1. Instruction Formats—No Address Register

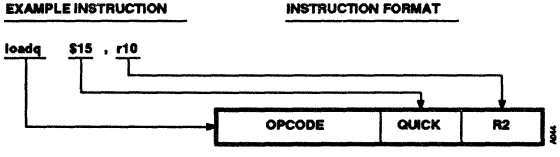
The Register format is used for most instructions that take just one or two register arguments.



The opcode specifies the interpretation of the R1 and R2 fields. Usually the R1 field contains the source operand register number, and R2 contains the destination operand register number. For example, in the **movsw** instruction, the R1 field contains the number of the single-precision floating-point register containing the source operand, and the R2 field contains the number of the general register in which to store the result.

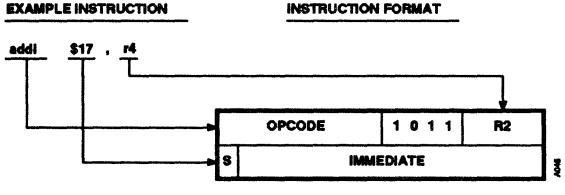
Quick

The Quick format encodes constant, 4-bit unsigned source operands directly in the instruction. The quick value is always zero-filled at the left before use.



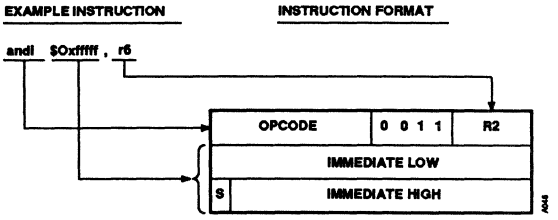
16-bit Immediate

The 16-bit Immediate format encodes a 16-bit source operand constant directly in the instruction. The immediate value is always sign-extended before use.



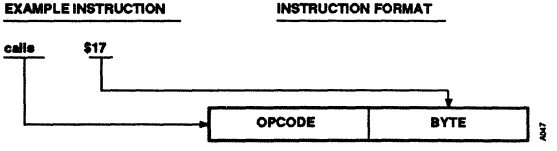
32-bit Immediate

The 32-bit Immediate format encodes a constant, 32-bit source operand directly in the instruction.



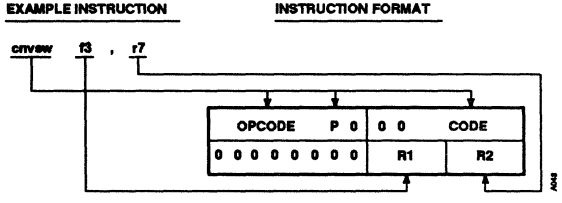
Control

The Control format encodes up to 8 bits of a constant value that is used by several special instructions. For example, the byte operand specifies the system call number in the **calls** instruction.



Macro

The Macro format is used by those instructions that are implemented as macros rather than directly in the hardware. The P bit in the opcode, bit 9 of the first instruction parcel, selects a privileged macro.



5.1.2. Instruction Formats—With Address

The remaining instruction formats specify an address operand and a register operand. Several address formats, or modes, are provided to support typical high-level language operations. The address mode is selected first by the opcode (bit 8 of the first instruction parcel), and if necessary, by the AM field (bits <7:4> of the first instruction parcel). Displacements and absolute addresses are always sign-extended.

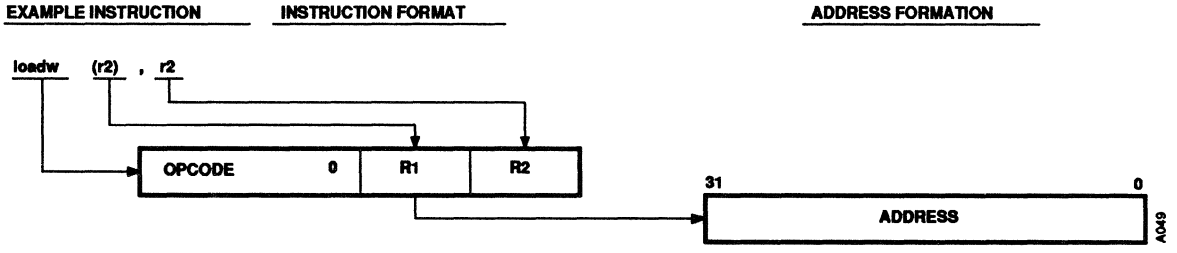
The address modes used in the memory referencing instructions are summarized in *Table 2* and explained in the following pages.

Table 2 Memory Addressing Modes

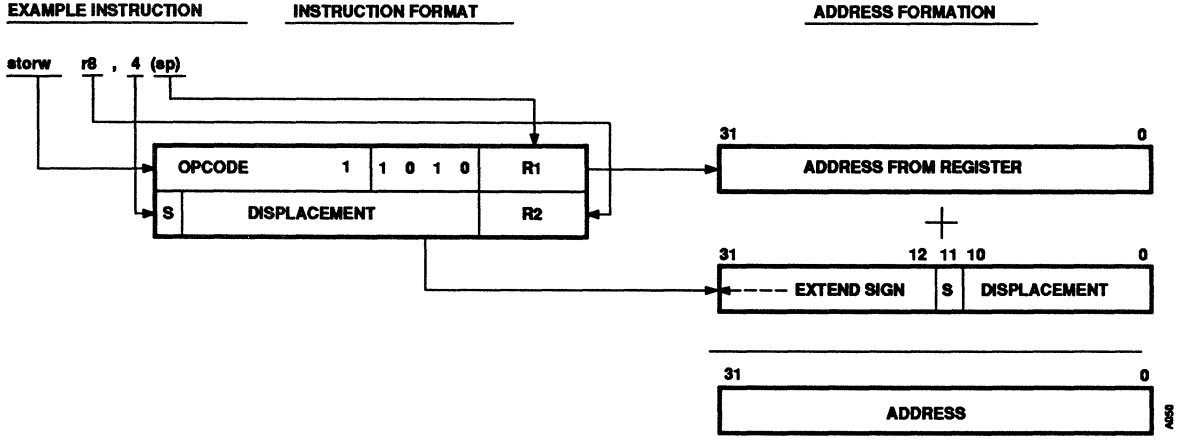
Memory Addressing Mode	Address Formation
Relative	Address ← (R1)
Relative plus 12-bit displacement	Address ← (R1) + 12-bit displacement
Relative plus 32-bit displacement	Address ← (R1) + 32-bit displacement
16-bit Absolute	Address ← 16-bit displacement
32-bit Absolute	Address ← 32-bit displacement
PC Relative plus 16-bit displacement	Address ← (PC) + 16-bit displacement
PC Relative plus 32-bit displacement	Address ← (PC) + 32-bit displacement
Relative Indexed	Address ← (R1) + (RX)
PC Indexed	Address ← (PC) + (RX)

Notes:
All displacements are signed.
PC addresses the first parcel of the current instruction.
RX is any general register containing the index modifying the effect of the source register.

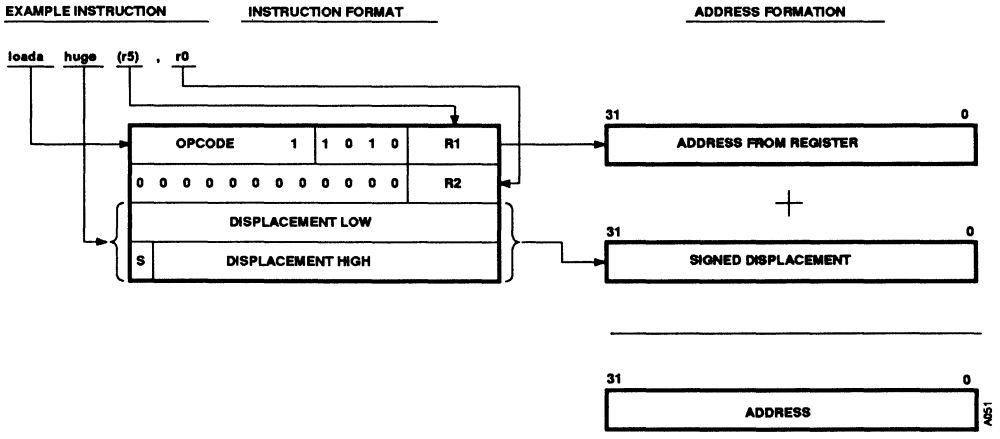
Relative
The Relative format uses the address in a register (R1) to compute an address.



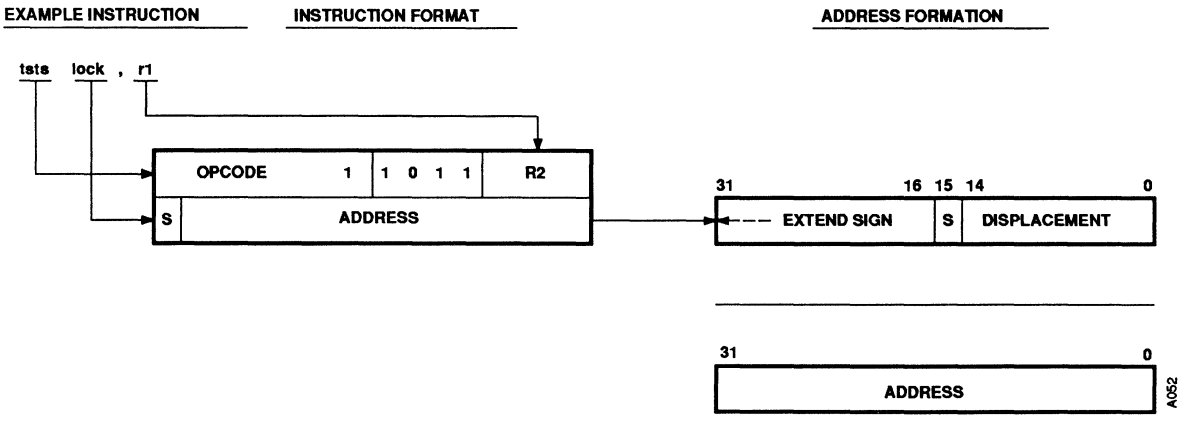
Relative Plus 12-bit Displacement
The Relative Plus 12-bit Displacement format uses the address in a register (R1), plus a signed 12-bit displacement, to compute an address. The displacement is sign-extended to 32 bits before the address calculation.



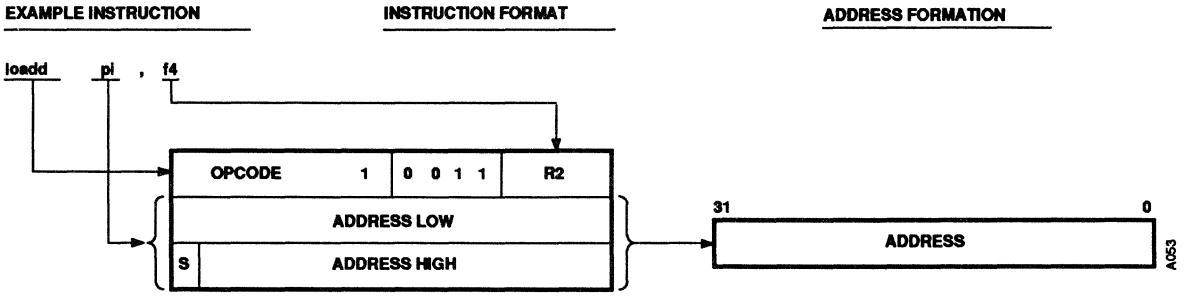
Relative Plus 32-bit Displacement
The Relative Plus 32-bit Displacement format uses the address in a register (R1), plus a signed 32-bit displacement, to compute an address.



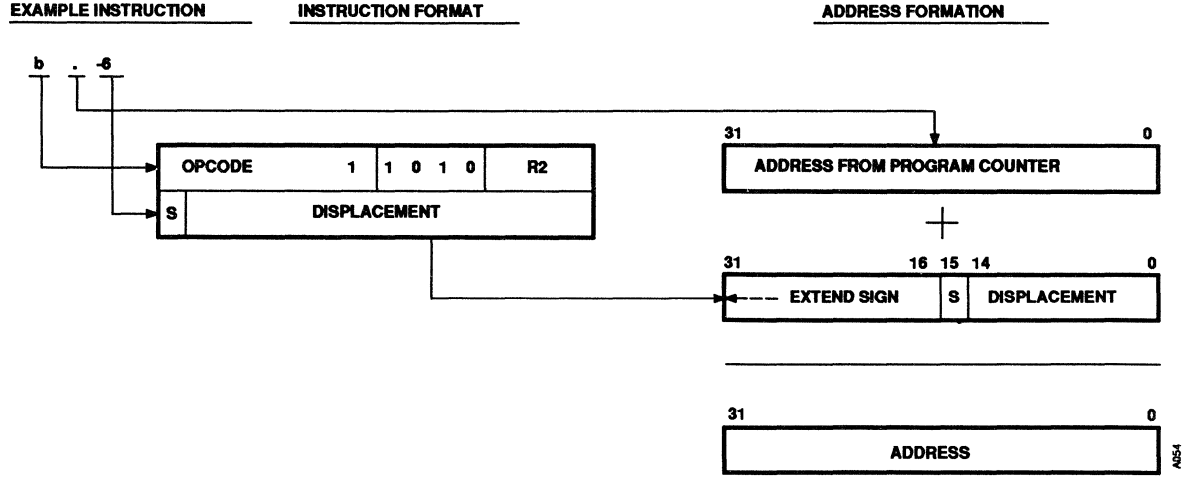
16-bit Absolute
The 16-bit Absolute format uses the signed 16-bit address, which is sign-extended to 32 bits before use. Because the address field is signed, the range of addresses that can be accessed with this format is $-2^{15} \leq$ address $\leq 2^{15} - 1$.



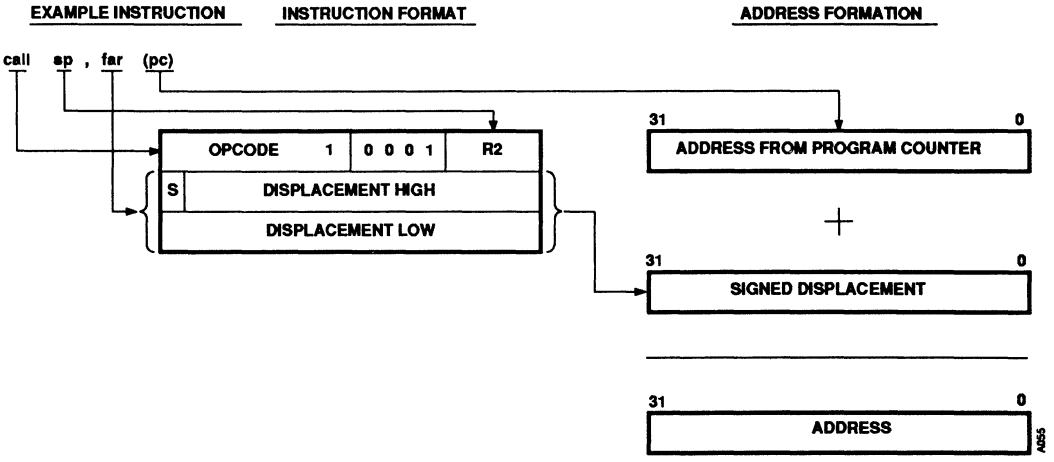
32-bit Absolute
The 32-bit Absolute format uses the 32-bit displacement portion of the instruction as an address.



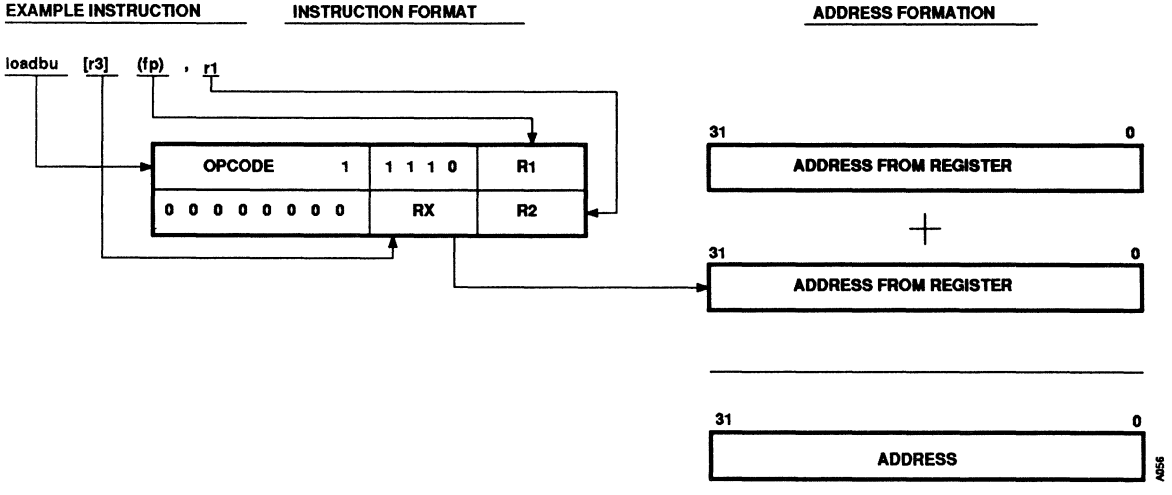
PC Relative Plus 16-bit Displacement
The PC Relative Plus 16-bit Displacement format adds a signed 16-bit displacement to the contents of the Program Counter (PC) to compute an address.



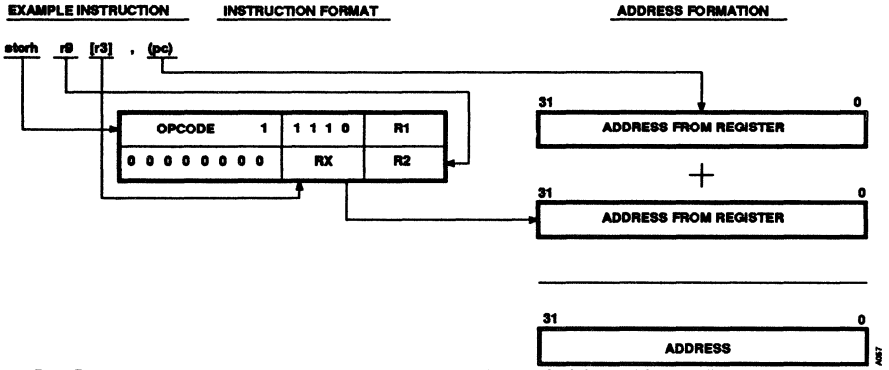
PC Relative Plus 32-bit Displacement
The PC Relative Plus 32-bit Displacement format adds a signed 32-bit displacement to the contents of the Program Counter (PC) to compute the address.



Relative Indexed
The Relative Indexed format uses the address in a register (R1), plus the contents of an index register (RX), to compute an address.



PC Indexed
The PC Indexed format adds the contents of an index register (RX) to the contents of the PC to compute an address.



5.2. Instruction Set Summary

Table 5 is a summary of the instruction set. Each instruction is described by several columns in the table. The columns are as follows:

Instruction Name
The full name of the instruction.

Syntax
Assembler instruction name and operand formats. The left letter of the operand code specifies the operand's type and size. The right letter of the operand code specifies the operand's field within the instruction and its location in the machine (immediate value, register, memory, etc.).

Operand Type				Operand Field			
b	byte	s	single floating	1	R1	a	address
h	halfword	d	double floating	2	R2	b	byte
w	word	p	processor register	q	quick		
l	longword			i	immediate		

For example, the operand code **w1** indicates a word operand in the general register whose number is encoded in the R1 field of the instruction. The code **sa** indicates a single floating operand in the memory location whose address is given by one of the addressing modes in *Section 5.1.2, Instruction Formats —With Address.* Quick and immediate operand types are always **w** because these directly encoded values are always zero or sign extended to a word before use.

Opcode
Hexadecimal opcode. Address format instructions use two opcodes; the first one listed is for relative mode, and the second is for all other addressing modes. Macro format instructions show the entire first parcel.

Format
Instruction format. See *Section 5.1, Instruction Formats.*

Operation
Basic operation of the instruction. The operations of complex instructions like **movc** are simplified or abbreviated. Fixed registers are given by name, e.g., r0, f1.

CVZN
Effect of the instruction on condition codes in the PSW.
- 0 = always set to 0.
- 1 = always set to 1.
- . = unaffected.
- * = set according to the operation.

FI, FV, FD, FU, FX
Effect of the instruction on the floating-point exception flags in the PSW. Same key as **CVZN**.

Traps

Traps that can be caused by the instruction.

- C = Corrected Memory Error
- U = Uncorrectable Memory Error
- P = Page Fault
- R = Read Protect Fault
- W = Write Protect Fault
- D = Divide by Zero
- I = Illegal Operation
- S = Supervisor Only (privileged)Iinstruction

All instructions can cause traps from the Instruction Memory Trap group in the I-CAMMU (for example, an execute protect fault), so these are not shown. Possible floating-point traps are indicated by an asterisk (*) in the **FI..FX** columns.

The instruction operand codes described above also describe the syntax of each instruction operand. Assembler operand syntax is given in *Table 3* below. Assembler instruction operands are generally given in source, destination order independent of their positions in the machine representation. *Table 4* lists the operators used in the operation field

Table 3 Assembler Operand Syntax

r0 . . r15	General registers. The even general registers address long operands. sp, fp, and ap are synonyms for r15, r14, and r13. Not to be confused with R1 or R2, which are register fields within an instruction.
f0 . . f7	Floating registers. Each register may contain a single or double floating value.
psw, ssw, sswf	Processor registers 0, 1,and 3.
$n	Quick, byte or immediate value.
n	Absolute address.
n(rm)	Relative or relative with displacement address. n may be 0 or absent.
[rx](rn)	Relative indexed address.
n(pc) or .±n	PC relative address.
[rx](pc)	PC indexed address.
label	Absolute or PC relative address depending on the circumstances.

Table 4 Operators

Notation	Meaning
rot	Rotate operator
sha	Shift arithmetic operator
shl	Shift logical operator
+	Add operator
−	Subtract operator
×	Multiply operator
÷	Divide operator
mod	Modulus operator
~	Logical complement operator
=	Equal operator
←	Assignment operator
&	AND logical operator
|	OR logical operator
⊕	Exclusive-OR logical operator
( )	Contents of operand within
[ ]	Separators used to indicate value inside as a unit
< >	Bit field of previous value
..	Indicates a range of values
↑	Exponentiation

Table 5 Instruction Operations

Instruction Name	Syntax		Opcode	Format	Operation	FFFFF IVDUX	CVZN	Traps
Add Double Floating	addd	*d1,d2*	22	Register	d2 ← (d2) + (d1)	* * . * *	. . . .	
Add Immediate	addi	*wi,w2*	83	Immediate	w2 ← (w2) + wi	. . . . .	* * * *	I
Add Quick	addq	*wq,w2*	82	Quick	w2 ← (w2) + wq	. . . . .	* * * *	
Add Single Floating	adds	*s1,s2*	20	Register	s2 ← (s2) + (s1)	* * . * *	. . . .	
Add Word	addw	*w1,w2*	80	Register	w2 ← (w2) + (w1)	. . . . .	* * * *	
Add Word with Carry	addwc	*w1,w2*	90	Register	w2 ← (w2) + (w1) + C	. . . . .	* * * *	
And Immediate	andi	*wi,w2*	8b	Immediate	w2 ← (w2) & wi	. . . . .	0 0 * *	I
And Word	andw	*w1,w2*	88	Register	w2 ← (w2) & (w1)	. . . . .	0 0 * *	
Branch Conditional	b*	*ha*	48,49	Address	IF cond, PC ← ha	. . . . .	. . . .	A,I
Branch on Floating Exception	bf*	*ha*	4c,4d	Address	IF cond, PC ← ha	. . . . .	. . . .	A,I
Call Routine	call	*w2,ha*	44,45	Address	w2 ← (w2) – 4, (w2) ← (PC), PC ← ha	. . . . .	. . . .	A,P,W
Call Supervisor	calls	*bb*	12	Control	trap 400 + 8 × bb<7:0>	. . . . .	. . . .	
Compare Characters r0=length, r1=string1, r2=string2	cmpc		b4 0f	Macro	while [(r0)≠0] & [((r2))=((r1))], r0 ← (r0) – 1, r1 ← (r1) + 1, r2 ← (r2) + 1	. . . . .	* * * *	C,U,P,R
Compare Double Floating	cmpd	*d1,d2*	27	Register	(d2) – (d1)	. . . . .	0 0 * *	
Compare Immediate	cmpi	*wi,w2*	a7	Immediate	(w2) – wi	. . . . .	* * * *	I
Compare Quick	cmpq	*wq,w2*	a6	Quick	(w2) – wq	. . . . .	* * * *	
Compare Single Floating	cmps	*s1,s2*	25	Register	(s2) – (s1)	. . . . .	0 0 * *	
Compare Word	cmpw	*w1,w2*	a4	Register	(w2) – (w1)	. . . . .	* * * *	
Convert Double Floating to Single	cnvds	*d1,s2*	b4 39	Macro	s2 ← (d1)	* * . * *	. . . .	
Convert Double Floating to Word	cnvdw	*d1,w2*	b4 34	Macro	w2 ← (d1)	* . . . *	. . . .	
Convert Rounding Double to Word	cnvrdw	*d1,w2*	b4 35	Macro	w2 ← (d1)	* . . . *	. . . .	
Convert Rounding Single to Word	cnvrsw	*s1,w2*	b4 31	Macro	w2 ← (s1)	* . . . *	. . . .	
Convert Single Floating to Double	cnvsd	s1,*d2*	b4 38	Macro	d2 ← (s1)	*	. . . .	
Convert Single Floating to Word	cnvsw	*s1,w2*	b4 30	Macro	w2 ← (s1)	* . . . *	. . . .	
Convert Truncating Double to Word	cnvtdw	d1,*w2*	b4 36	Macro	w2 ← (d1)	* . . . *	. . . .	
Convert Truncating Single to Word	cnvtsw	*s1,w2*	b4 32	Macro	w2 ← (s1)	* . . . *	. . . .	
Convert Word to Double Floating	cnvwd	*w1,d2*	b4 37	Macro	d2 ← (w1)	. . . . .	. . . .	
Convert Word to Single Floating	cnvws	*w1,s2*	b4 33	Macro	s2 ← (w1)	. . . . *	. . . .	
Divide Double Floating	divd	*d1,d2*	2b	Register	d2 ← (d2) ÷ (d1)	* * * * *	. . . .	
Divide Single Floating	divs	*s1,s2*	29	Register	s2 ← (s2) ÷ (s1)	* * * * *	. . . .	
Divide Word	divw	*w1,w2*	9c	Register	w2 ← (w2) ÷ (w1)	. . . . .	0 * 0 0	D
Divide Word Unsigned	divwu	w1,*w2*	9e	Register	w2 ← (w2) ÷ (w1)	. . . . .	0 0 0 0	D
Initialize Characters r0=length, r1=dest, r2=pattern	initc		b4 0e	Macro	while (r1)≠0, (r1) ← (r2<7:0>), r0 ← (r0) – 1, r1 ← (r1) + 1, r2 ← (r2) rot –8	. . . . .	. . . .	P,W
Load Address	loada	*ba,w2*	62,63	Address	w2 ← ba	. . . . .	. . . .	I
Load Byte	loadb	*ba,w2*	68,69	Address	w2 ← (ba)	. . . . .	. . . .	C,U,A,P,R,I
Load Byte Unsigned	loadbu	*ba,w2*	6a,6b	Address	w2 ← (ba)	. . . . .	. . . .	C,U,A,P,R,I
Load Double Floating	loadd	*da,d2*	66,67	Address	d2 ← (da)	. . . . .	. . . .	C,U,A,P,R,I
Load Floating Status	loadfs	*w1,d2*	b4 3f	Macro	w1 ← (FP PC), d2 ← (FP dest)	. . . . .	. . . .	
Load Halfword	loadh	*ha,w2*	6c,6d	Address	w2 ← (ha)	. . . . .	. . . .	C,U,A,P,R,I
Load Halfword Unsigned	loadhu	*ha,w2*	6e,6f	Address	w2 ← (ha)	. . . . .	. . . .	C,U,A,P,R,I
Load Immediate	loadi	*wi,w2*	87	Immediate	w2 ← wi	. . . . .	0 0 * *	I
Load Quick	loadq	*wq,w2*	86	Quick	w2 ← wq	. . . . .	0 0 * 0	
Load Single Floating	loads	*sa,s2*	64,65	Address	s2 ← (sa)	. . . . .	. . . .	C,U,A,P,R,I
Load Word	loadw	*wa,w2*	60,61	Address	w2 ← (wa)	. . . . .	. . . .	C,U,A,P,R,I

Table 5 Instruction Operations (cont.)

Instruction Name	Syntax		Opcode	Format	Operation	FFFFF IVDUX	CVZN	Traps
Modulus Word	modw	*w1,w2*	9d	Register	w2 ← (w2) mod (w1)	· · · · ·	0 * 0 0	D
Modulus Word Unsigned	modwu	*w1,w2*	9f	Register	w2 ← (w2) mod (w1)	· · · · ·	0 0 0 0	D
Move Characters r0=length, r1=source, r2=dest	movc		b4 0d	Macro	while (r0) = 0, (r2) ← ((r1)), r0 ← (r0) – 1, r1 ← (r1) + 1, r2 ← (r2) + 1	· · · · ·	· · · ·	C,U,P,R,W
Move Double Floating	movd	*d1,d2*	26	Register	d2 ← (d1)	· · · · ·	· · · ·	
Move Double Floating to Longword	movdl	*d1,l2*	2e	Register	l2 ← (d1)	· · · · ·	· · · ·	
Move Longword to Double Floating	movld	*l1,d2*	2f	Register	d2 ← (l1)	· · · · ·	· · · ·	
Move Processor Register to Word	movpw	*p1,w2*	11	Register	w2 ← (p1)	· · · · ·	· · · ·	
Move Single Floating	movs	*s1,s2*	24	Register	s2 ← (s1)	· · · · ·	· · · ·	
Move Supervisor to User (privileged)	movsu	*w1,w2*	b6 01	Macro	w2 ← (w1)	· · · · ·	0 0 * *	S
Move Single Floating to Word	movsw	*s1,w2*	2c	Register	w2 ← (s1)	· · · · ·	· · · ·	
Move User to Supervisor (privileged)	movus	*w1,w2*	b6 00	Macro	w2 ← (w1)	· · · · ·	0 0 * *	S
Move Word	movw	*w1,w2*	84	Register	w2 ← (w1)	· · · · ·	0 0 * *	
Move Word to Processor Register	movwp	*w2,p1*	10	Register	p1 ← (w2)	· · · · ·	* * * *	
Move Word to Single Floating	movws	*w1,s2*	2d	Register	s2 ← (w1)	· · · · ·	· · · ·	
Multiply Double Floating	muld	*d1,d2*	2a	Register	d2 ← (d2) × (d1)	* * · * *	· · · ·	
Multiply Single Floating	muls	*s1,s2*	28	Register	s2 ← (s2) × (s1)	* * · * *	· · · ·	
Multiply Word	mulw	*w1,w2*	98	Register	w2 ← (w2) × (w1)	· · · · ·	0 * 0 0	
Multiply Word Unsigned	mulwu	*w1,w2*	9a	Register	w2 ← (w2) × (w1)	· · · · ·	0 * 0 0	
Multiply Word Unsigned Extended	mulwux	*w1,l2*	9b	Register	l2 ← (w2) × (w1)	· · · · ·	0 * 0 0	
Multiply Word Extended	mulwx	*w1,l2*	99	Register	l2 ← (w2) × (w1)	· · · · ·	0 * 0 0	
Negate Double Floating	negd	*d1,d2*	b4 3b	Macro	d2 ← –(d1)	· · · · ·	· · · ·	
Negate Single Floating	negs	*s1,s2*	b4 3a	Macro	s2 ← –(s1)	· · · · ·	· · · ·	
Negate Word	negw	*w1,w2*	93	Register	w2 ← –(w1)	· · · · ·	* * * *	
No Operation	noop	*bb*	00	Control	none	· · · · ·	· · · ·	
Not Quick	notq	*wq,w2*	ae	Quick	w2 ← ~wq	· · · · ·	0 0 0 1	
Not Word	notw	*w1,w2*	ac	Register	w2 ← ~(w1)	· · · · ·	0 0 * *	
Or Immediate	ori	*wi,w2*	8f	Immediate	w2 ← (w2) | wi	· · · · ·	0 0 * *	I
Or Word	orw	*w1,w2*	8c	Register	w2 ← (w2) | (w1)	· · · · ·	0 0 * *	
Pop Word	popw	*w1,w2*	16	Register	w1 ← (w1) + 4, w2 ← ((w1) – 4)	· · · · ·	· · · ·	C,U,A,P,R
Push Word	pushw	*w2,w1*	14	Register	w1 ← (w1) – 4, (w1) ← (w2)	· · · · ·	· · · ·	A,P,W
Restore Registers fn - f7 0 ≤ *n* ≤ 7	restdn		b4 28 .. b4 2F	Macro	f*n* .. f7 ← ((r15)) .. ((r15) + 8 × [7–*n*]), r15 ← (r15) + 8 × [8–*n*]	· · · · ·	· · · ·	C,U,A,P,R
Restore User Registers (privileged)	restur	*w1*	b6 03	Macro	r0 .. r15 ← ((w1)) .. ((w1) + 60)	· · · · ·	· · · ·	C,U,A,P,R,S
Restore Registers r*n* - r14 0 ≤ *n* ≤ 12	restw*n*		b4 10 .. b4 1C	Macro	r*n* .. r14 ← ((r15)) .. ((r15) + 4 × [14–*n*]), r15 ← (r15) + 4 × [15–*n*]	· · · · ·	· · · ·	C,U,A,P,R
Return From Routine	ret	*w2*	13	Register	PC ← ((w2)) w2 ← (w2) + 4	· · · · ·	· · · ·	C,U,A,P,R
Return From Interrupt (privileged)	reti	*w1*	b6 04	Macro	Restore SSW, PSW and PC	· · · · ·	· · · ·	S
Rotate Immediate	roti	*wi,w2*	3c	Immediate	w2 ← (w2) rot wi	· · · · ·	0 0 * *	I
Rotate Longword	rotl	*w1,l2*	35	Register	l2 ← (l2) rot (w1)	· · · · ·	0 0 * *	
Rotate Longword Immediate	rotli	*wi,l2*	3d	Immediate	l2 ← (l2) rot wi	· · · · ·	0 0 * *	I
Rotate Word	rotw	*w1,w2*	34	Register	w2 ← (w2) rot (w1)	· · · · ·	0 0 * *	
Save Registers f*n* - f7 0 ≤ *n* ≤ 7	saved*n*		b4 20 .. b4 27	Macro	(r15) – 8 × [8 – *n*] .. (r15) – 8 ← (f*n*) .. (f7), r15 ← (r15) – 8 × [8 – *n*]	· · · · ·	· · · ·	A,P,W

Table 5 Instruction Operations (cont.)

Instruction Name	Syntax		Opcode	Format	Operation	FFFFF IVDUX	CVZN	Traps
Save User Registers (privileged)	saveur	*w1*	b6 02	Macro	(w1) – 4 .. (w1) – 64 ← (r15) .. (r0)	· · · · ·	· · · ·	A,P,W,S
Save Registers *rn* - r14 $0 \le n \le 12$	savew*n*		b4 00 .. b4 0C	Macro	(r15) – 4 × [15 –*n*] .. (r15) – 4 ← (*rn*) .. (r14), r15 ← (r15) – 8 × [8 – *n*]	· · · · ·	· · · ·	A,P,W
Scale by, Double Floating	scalbd	*w1,d2*	b4 3d	Macro	d2 ← (d2) × $2^{(w1)}$	* * · * *	· · ·	
Scale by, Single Floating	scalbs	*w1,s2*	b4 3c	Macro	s2 ← (s2) × $2^{(w1)}$	* * · * *	· · · ·	
Shift Arithmetic Immediate	shai	*wi,w2*	38	Immediate	w2 ← (w2) sha wi	· · · · ·	0 * * *	I
Shift Arithmetic Longword	shal	*w1,l2*	31	Register	l2 ← (l2) sha (w1)	· · · · ·	0 * * *	
Shift Arithmetic Longword Immediate	shali	*wi,l2*	39	Immediate	l2 ← (l2) sha wi	· · · · ·	0 * * *	I
Shift Arithmetic Word	shaw	*w1,w2*	30	Register	w2 ← (w2) sha (w1)	· · · · ·	0 * * *	
Shift Logical Immediate	shli	*wi,w2*	3a	Immediate	w2 ← (w2) shl wi	· · · · ·	0 0 * *	I
Shift Logical Longword	shll	*w1,l2*	33	Register	l2 ← (l2) shl (w1)	· · · · ·	0 0 * *	
Shift Logical Longword Immediate	shlli	*wi,l2*	3b	Immediate	l2 ← (l2) shl wi	· · · · ·	0 0 * *	I
Shift Logical Word	shlw	*w1,w2*	32	Register	w2 ← (w2) shl (w1)	· · · · ·	0 0 * *	
Store Byte	storb	*w2,ba*	78,79	Address	ba ← (w2)	· · · · ·	· · · ·	A,P,W,I
Store Double Floating	stord	*d2,da*	76,77	Address	da ← (d2)	· · · · ·	· · · ·	A,P,W,I
Store Halfword	storh	w2,*ha*	7c,7d	Address	ha ← (w2)	· · · · ·	· · · ·	A,P,W,I
Store Single Floating	stors	*s2,sa*	74,75	Address	sa ← (s2)	· · · · ·	· · · ·	A,P,W,I
Store Word	storw	*w2,wa*	70,71	Address	wa ← (w2)	· · · · ·	· · · ·	P,W,I
Subtract Double Floating	subd	*d1,d2*	23	Register	d2 ← (d2) – (d1)	* * · * *	· · · ·	
Subtract Immediate	subi	*wi,w2*	a3	Immediate	w2 ← (w2) – wi	· · · · ·	* * * *	I
Subtract Quick	subq	*wq,w2*	a2	Quick	w2 ← (w2) – wq	· · · · ·	* * * *	
Subtract Single Floating	subs	*s1,s2*	21	Register	s2 ← (s2) – (s1)	* * · * *	· · · ·	
Subtract Word	subw	*w1,w2*	a0	Register	w2 ← (w2) – (w1)	· · · · ·	* * * *	
Subtract Word with Carry	subwc	*w1,w2*	91	Register	w2 ← (w2) – (w1) – C	· · · · ·	* * * *	
Test and Set	tsts	*wa,w2*	72,73	Address	w2 ← (wa), wa ← 1	· · · · ·	· · · ·	I
Trap on Floating Unordered	trapfn		b4 3e	Macro	IF PSW<ZN> indicates unordered, illegal instruction trap	· · · · ·	· · · ·	C,U,A,P, R,W,I
Wait for Interrupt (privileged)	wait		b6 05	Macro	Wait for interrupt	· · · · ·	· · · ·	S
Exclusive-OR Immediate	xori	*wi,w2*	ab	Immediate	w2 ← (w2) (+) wi	· · · · ·	0 0 * *	I
Exclusive-OR Word	xorw	*w1,w2*	a8	Register	w2 ← (w2) (+) (w1)	· · · · ·	0 0 * *	

Table 6A Integer Branch Conditions

cond	PSW Flags C	V	Z	N	Name	Condition
0	X	X	X	X	b	Branch always

cond	PSW Flags C	V	Z	N	Name	Compare R1:R2	Name	Result R2:0
1	X X	0 1	0 0	0 1	bclt	Less Than	brgt	Greater Than
2	X X	0 1	X 0	0 1	bcle	Less or Equal	brge	Greater or Equal
3	X	X	1	0	bceq	Equal	breq	Equal
4	X X	0 1	0 X	1 0	bcgt	Greater Than	brlt	Less Than
5	X X X	1 0 X	X 0 1	0 1 0	bcge	Greater or Equal	brle	Less or Equal
6	X X	X X	0 1	X 1	bcne	Not Equal	brne	Not Equal
7	0	X	0	X	bcltu	Less Than Unsigned	brgtu	Greater Than Unsigned
8	0	X	X	X	bcleu	Less or Equal Unsigned	brgeu	Greater or Equal Unsigned
9	1	X	X	X	bcgtu	Greater Than Unsigned	brltu	Less Than Unsigned
A	1 X	X X	X 1	X X	bcgeu	Greater or Equal Unsigned	brleu	Less or Equal Unsigned

cond	PSW Flags C	V	Z	N	Name	Condition
8	0	X	X	X	bnc	Not Carry
9	1	X	X	X	bc	Carry
B	X	1	X	X	bv	Overflow
C	X	0	X	X	bnv	Not Overflow
D	X	X	0	1	bn	Negative
E	X	X	X	0	bnn	Not Negative
F	X	X	1	1	bfn	Floating Unordered

The R2 field of the **branch on condition** instruction selects the conditions on which to branch. When a choice of mnemonics is shown, use the ones beginning with **bc** if the condition codes to be tested were set by a compare instruction. Use the mnemonics beginning with **br** is they were set by move or logical instructions (those instructions that set only N or Z).

Table 6B Floating Branch Conditions

cond	Name	Condition
0	bfany	Floating ANY exception
1	bfbad	Floating BAD result
2-F		Reserved

6. Exceptions

The CLIPPER architecture supports 402 exception conditions: 18 hardware traps, 128 programmable supervisor call traps, and 256 vectored interrupts.

Traps are exceptions recognized by the CPU during execution of single instructions (e.g., divide by zero, page fault). A trap causes all instructions in both the upper and lower pipelines to either be backed out or completed in a manner consistent with program restart.

Interrupts are events signalled by devices external to the CLIPPER Module and input to the module via the interrupt pins. Interrupts are taken when the following conditions are met:

— Interrupts are enabled.

— The Interrupt Level ($\overline{IVEC}$<7:4>) is less than or equal to the IL field in the SSW.

— All instructions in the lower pipeline have finished executing. String instructions have either completed execution or have detected the interrupt and saved sufficient state information for continuation.

— No traps are pending.

A flow chart showing the necessary conditions for interrupts is shown in *Figure 14*.

The address of the service routine for each trap, supervisor call, and interrupt is stored in an Exception Vector Table (see *Table 7*), located in the first real page of main memory. The Exception Vector Table (EVT) contains a two-word entry for each exception, consisting of the starting address of the exception's service routine and an SSW value associated with the routine. Unassigned EVT addresses are reserved for future use by Intergraphand must be initialized to point to a valid handler routine.

The priority of exceptions is the order shown in the EVT (in the order from highest to lowest priority), except that the trace trap has the lowest priority. The CLIPPER Module's internal priority logic ensures that exception service is always granted to the highest priority event.

Table 7 Exception 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 Divided by Zero
Instruction Memory Trap Group	
288	Corrected Memory Error
290	Uncorrectable Memory Error
2A8	Page Fault
2B0	Execute Protect Fault
Ilegal Operation Trap Group	
300	Illegal Operation
308	Privileged Instruction
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 Level 0 Number 1
810	Interrupt Level 0 Number 2
.	
.	
.	
878	Interrupt Level 0 Number 15
880	Interrupt Level 1 Number 0
888	Interrupt Level 1 Number 1
.	
.	
.	
FF8	Interrupt Level 15 Number 15

6.1. INTRAP and reti Sequences

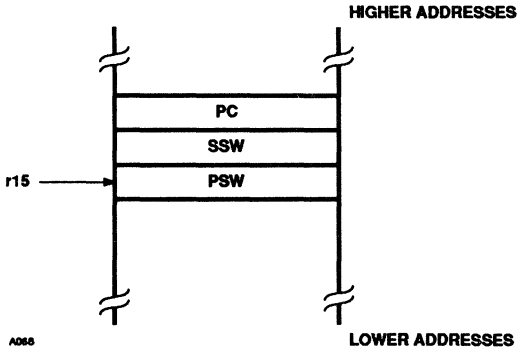
Two macro instruction sequences, INTRAP and **reti**, manage the entry to and exit from both traps and interrupts. The INTRAP sequence performs a non-interruptable context switch to supervisor mode, and then transfers control to the trap or interrupt handler. The **reti** sequence is an interrupt/trap return, also non-interruptible, which restores the system to the correct user or supervisor environment.

During the INTRAP and **reti** sequences, all interrupts are disabled; traps are not disabled, but only serious system faults can occur, as explained below.

The INTRAP sequence begins by saving the PC, SSW, and PSW on the supervisor stack as shown in *Figure 13*. The saved PSW will have MTS or CTS set to indicate the cause of the trap. INTRAP then copies the SSW's user mode flag (U) into the previous mode flag (P). In order to access the Vector Table, INTRAP sets the user mode flag to supervisor mode and clears the protect key (K), user data mode (UU), and user protect key (KU). The PSW is cleared.

The address of the required Exception Vector Table entry, V, is then obtained in one of three ways: 1) For traps and the non-maskable interrupt, the address is generated from internal trap logic. 2) For supervisor calls, the address is generated from the lower 7 bits of the instruction. This value is multiplied by 8 and 400H is added to it. 3) For priority interrupts, a number is read from the Interrupt Bus lines, $\overline{\text{IVEC}}$<7:0>. This value is inverted, multiplied by 8, and 800H is added to it.

Figure 13 Supervisor Stack After INTRAP



INTRAP uses V to obtain the new PC value and V + 4 to obtain the associated SSW value. The new SSW value is transferred to the SSW, overwriting the previous contents of SSW except for the previous mode flag (P), which is retained in order to indicate the mode of the interrupted program. INTRAP then exits, and control is passed to the trap or interrupt service routine.

After completing its service, the trap or interrupt handler executes the **reti** sequence. **reti** restores the PSW, SSW and PC to their contents prior to INTRAP.

6.1.1. Faults During INTRAP and reti

The occurrence of a trap during INTRAP or **reti** results in an Unrecoverable Fault (URF). The CLIPPER Module halts in a controlled suspended state, drives the $\overline{\text{URF}}$ signal low as an alarm, and waits until restarted by the $\overline{\text{RESET}}$ signal. (In the URF state, all inputs other than $\overline{\text{RESET}}$ are ignored.)

To avoid the occurrence of a page fault during INTRAP or **reti** (and the resulting URF condition), the supervisor stack must always have a valid Page Table entry that permits both reading and writing. This will prevent page faults from occurring during INTRAP or **reti**, because the supervisor stack is the only memory area referenced by these sequences.

6.2. Traps

Traps are signalled in the CPU chip or by either of the CAMMUs. There are 18 predefined traps, shown in *Table 7*.

Both conditional and unconditional traps are supported (see *Table 8*). Conditional traps are enabled by flags in the PSW and occur only when enabled. Conditional

Table 8 Conditional and Unconditional Traps

Conditional Traps	Unconditional Traps
Corrected Memory Error Floating-Point Arithmetic Trap Group Trace	Uncorrectable Memory Error Page Fault Protect Faults Privileged Instruction Illegal Operation Integer Divide by Zero Supervisor Call

Table 9 Trap Handler Environment Summary

Trap	When Trap Is Taken	Return Address (Saved In Supervisor Stack)
Data Memory Trap Group	During Execution	Faulting Instruction
Floating-Point Arithmetic Trap Group	After Execution	Next Instruction To Be Executed
Integer Arithmetic Trap Group	After Execution	Next Instruction To Be Executed
Instruction Memory Trap Group	Before Execution	Faulting Instruction
Illegal Operation Trap Group	Before Execution	Faulting Instruction
Diagnostic Trap Group	After Execution	Following Instruction
Supervisor Call	After Execution	Following Instruction

traps that are disabled can be detected and handled by the executing program.

Traps may be generated at various stages of instruction processing, as shown in *Table 9*. The CLIPPER Module's internal trap logic ensures that the saved program counter points to the instruction at which the trapped program may be correctly restarted.

6.2.1. Data Memory Trap Group

Data memory traps occur when the data cache interface reports a fault. These traps cause the faulted instruction, as well as subsequent instructions already in the upper pipeline, to be backed out.

Data memory traps are recorded in the PSW's memory trap status (MTS) field. The MTS field is also used by the instruction memory trap group for the same fault conditions. Interpretation is not ambiguous because instruction memory traps are deferred until data memory traps have been serviced, and they are serviced by different trap handlers.

In the case of the **pushw** and **popw** instructions, the stack pointer is decremented or incremented in parallel with the instruction's memory access. Thus, when a data memory trap occurs during a **pushw** or **popw** instruction, the operating system must, before restarting the program, restore the stack pointer to the value it had prior to the trapping instruction, i.e., decrement the stack pointer by 4 for **popw**, or increment the stack pointer by 4 for **pushw**.

Corrected/Uncorrectable Memory Errors

Corrected and uncorrectable data memory errors are detected by memory and communicated to the CAMMU via the two system bus signals, $\overline{\text{MSBE}}/\overline{\text{RETRY}}$ and $\overline{\text{MMBE}}$ respectively. It is the responsibility of memory to save the real memory address of the location that failed in a predetermined location, where it may be accessed for maintenance by the operating system.

The operating system may ignore indications of corrected memory errors ($\overline{\text{MSBE}}/\overline{\text{RETRY}}$) by clearing the ECM flag in the SSW.

Page Fault

A page fault occurs when a program attempts to access a page for which there is no valid entry in the currently assigned Page Directory or Page Tables. The operating system uses this fault to allocate pages to user or supervisor programs. The address saved on the supervisor stack is the program address of the instruction that caused the page fault. The virtual address of the data memory location that generated the fault is saved in the CAMMU's Fault register.

Read/Write Protection Faults

Read/write accesses to each page are validated by a comparison of the U, K, UU, and KU flags in the SSW with the protection code in the TLB or user page tables. When an access violation occurs, the address saved on the supervisor stack is the program address of the instruction that caused the fault. The virtual address of the data memory location that generated the fault is saved in the CAMMU's Fault register.

6.2.2. Floating-Point Arithmetic Trap Group

There are five distinct floating-point exceptions which are specified in the IEEE Standard 754. These exceptions are signalled by the FPU in the case of invalid operation, inexact result, overflow, underflow, or divide

by zero. For each exception, there corresponds a floating-point exception flag in the PSW. The corresponding bit is set on any occurrence of the exception.

In addition, for each exception there exists a floating-point trap enable flag. There is also a floating-point group trap enable flag. When an exception arises for which the individual trap enable flag is true and the group trap enable flag is true, then a floating-point trap is invoked and control is transferred to a user-specified trap handler. If the group trap enable is false, then the trap is not invoked. If the individual trap enable flag is false, then the trap is not invoked.

For the underflow and overflow exceptions, the behavior of the FPU is determined by the values of the floating-point trap enable flags as specified in the Standard. In particular, overflows with the overflow trap disabled deliver infinity or max_real, whereas the result with the trap enabled is the normalized result with the exponent distorted, as discussed below. Underflows are handled similarly.

The software knows which floating-point trap has occurred because each floating-point trap invokes a separate trap handler (each has its own entry in the Exception Vector Table). It is not sufficient to examine the floating-point exception flags, since the state of these immediately before executing the exceptional operation is generally unknown.

Floating Overflow
The floating overflow exception is signalled when the biased exponent of the result (after rounding) is greater than the largest finite representable exponent. With addition and subtraction, overflow occurs when two large numbers are added. At least one of them must have a biased exponent of +126 (single-precision) or +1022 (double-precision) and the fraction addition (or the subsequent rounding) has a carry out of the msb position. The overflow may coincide with the fraction sum being inexact, though this is not necessarily the case. With multiplication, overflow occurs if, after normalization and rounding, the product of two finite non-zero numbers has an exponent greater than +127 (single-precision) or +1023 (double-precision). Overflow for multiplication may be exact or inexact.

If the EFV flag is set, the computed result is delivered to the destination with the normalized rounded fraction of the true result (though the delivered exponent is usually wrong because of missing additional leading bits in the exponent field). For single-precision overflows, if the biased exponent of the true result is 255, then biased exponent 255 is delivered to the destination. If the true biased exponent of the result is 256 . . 408, then the true biased exponent minus 256 is delivered to the destination. Note that this is not the exponent wrapped result called for by the IEEE 754 specification; the wrap must be adjusted by system software before delivery to a user's trap handler. This is done to allow the user to provide software that handles traps in an application-specific way. For double-precision, the overflow exponents (biased) lie in the range 2047 . . 3120. These are mapped to 2047 and 0 . . 1072 respectively. These must be adjusted by $(3/4)\times 2^{11}$ (1536) to obtain the IEEE Standard wrapped exponent.

If the EFV flag is clear, then the computed result is discarded, and the properly signed default value (infinity or max_real, depending on rounding mode) is delivered to the destination. Max_real is the maximum representable value in the given floating-point format; single max_real = $2^{128} - 2^{104}$; double_max_real = $2^{1024} - 2^{971}$. The floating inexact exception is also signalled. If the rounding mode is round toward zero, the value delivered to the destination is the maximum finite representable number (max_real) with the appropriate sign. If the rounding mode is round toward + ∞, then a positive signed overflow is replaced with + ∞, while a negative signed overflow is replaced by minus max_real. For round toward - ∞, a positive overflow goes to plus max_real, while a negative overflow goes to - ∞.

Floating Inexact
The floating inexact exception is signalled when the result of an operation cannot be exactly represented in the precision of the destination. The result is rounded according to the rounding mode specified in the PSW

so that it has the precision of the destination, and then the rounded result is delivered to the destination.

Floating Underflow
The conditions under which the floating underflow exception condition is signalled differ according to the setting of the EFU flag. If EFU is set, the floating underflow exception is signalled when the result of an operation (before rounding) has a biased exponent less than the minimum representable biased exponent for a normalized number. If the true biased exponent of the result is zero, then biased exponent zero is delivered to the destination. If the true biased exponent is less than zero, then the exponent delivered to the destination is true biased exponent plus 256 (2048 for double.) The exponent must be adjusted by system software before delivery to the program's trap handler in order to conform to the IEEE 754 Specification. The range of underflowed biased exponents for single-precision is 0 . . -275; for double-precision the range is 0 . . -1125.

If the EFU flag is clear, then the underflowed fraction is right shifted as the exponent is incremented until the biased exponent equals one. At this point, the result has been restated as a denormal number. If this representation is exact, then no underflow exception is signalled. If the representation is inexact, then the result is rounded and delivered to the destination, and both the inexact and underflow exceptions are signalled.

Floating Divide by Zero
The floating divide by zero exception is signalled when the divisor is zero and the dividend is non-zero and finite. If the dividend is also zero, the result is the default quiet NaN (all ones in the fraction and exponent fields), and the FI flag is set. If the dividend is infinite, the result is infinite, and no condition flags are set. The default result is a correctly signed infinity.

Floating Invalid Operation
The floating invalid operation exception is signalled in the following cases:

1. One of the operands is a signalling NaN.
2. Add/Subtract, magnitude subtraction of infinities:
   $(+\infty) - (+\infty)$
   or $(+\infty) + (-\infty)$
   or $(+\infty) - (-\infty)$
   or $(+\infty) + (+\infty)$
3. Multiplication
   $0 \times \infty$
   or $\infty \times 0$
4. Division
   $0 \div 0$
   or $\infty \div \infty$

The value written to the destination is always a NaN. The NaN is either the NaN operand (the second operand if both are NaNs) made quiet if it were signalling (by setting the msb of the explicit fraction field), or the default NaN created by the hardware. The default NaN is quiet, and its fraction field is all ones.

6.2.3. Integer Arithmetic Trap Group
The CPU trap status field in the saved PSW indicates the cause of the integer arithmetic trap.

Integer Divide by Zero
The integer divide by zero exception is signalled when an integer divide or mod instruction is executed with zero divisor.

Integer divide by zero cannot be disabled. The result of the trapped instruction will not be written to the specified register.

6.2.4. Instruction Memory Trap Group
Instruction memory faults are detected and signalled by the instruction interface. These traps are not acted upon when first sensed, i.e., if a branch instruction or other sequence altering event occurs between the time that the instruction interface detects the trap condition and when that instruction arrives at the C stage, then the pending trap condition is cleared and the trap is deferred. A deferred trap will not trap until it is ready to be issued for execution. If pre-empted by another trap, it may trap later if the code is restarted.

The faulting instruction has not yet entered the lower pipeline when the trap is taken. The program address saved is that of the faulting instruction.

For instruction memory traps, the memory trap status (MTS) field in the saved PSW indicates the reported error.

Corrected/Uncorrectable Memory Error
Corrected and uncorrectable data memory errors are detected by memory and communicated to the CAMMU via the two system bus signals, $\overline{\text{MSBE}}/\overline{\text{RETRY}}$ and $\overline{\text{MMBE}}$ respectively. It is the responsibility of the memory system to save the real memory address of the location that failed in a predetermined location in memory, where it can be accessed for maintenance by the operating system.

The operating system may ignore indications of corrected memory errors by clearing the ECM flag in the SSW.

Page Fault
A page fault occurs when a program tries to access a page for which there is no valid entry in the currently assigned Page Directory or Page Tables. The operating system uses this fault to allocate pages to user and supervisor programs. The address saved on the supervisor stack is the program address of the instruction that caused the page fault. The virtual address of the memory location that caused the fault is saved in the CAMMU's Fault register. (The two addresses may differ for multiple-parcel instructions.)

Execute Protect Fault
Instruction fetches from each page are validated by a comparison of the U, K, UU and KU flags in the SSW with the protection level in the TLB or user's page tables. When an instruction fetch violation occurs, the address saved on the supervisor stack is the program address of the instruction that caused the fault. The virtual address of the memory location that caused the fault is saved in the CAMMU's Fault register.

6.2.5. Illegal Operation Trap Group

Illegal operation traps are taken before the instruction is executed. The program address saved on the supervisor stack is the address of the instruction which caused the trap. The CPU trap status field in the saved PSW indicates the type of trap.

Illegal Operation Fault
An illegal operation trap results from the attempted execution of any undefined instruction opcode or the occurrence of an addressing mode which is not specifically allowed.

Privileged Instruction Fault
A privileged instruction trap occurs when a privileged macro instruction is encountered in user mode.

6.2.6. Diagnostic Trap Group

Trace Trap
Unless pre-empted by another trap or interrupt, the trace trap occurs following the execution of an instruction whenever the PSW's T (trace trap enable) flag is set. For traced instructions which are interrupted or cause traps, the TP flag is set by hardware when the interrupt or trap occurs to ensure that the trace trap will occur immediately after the interrupt or other trap has been serviced. In the case of data page faults, TP must be cleared by the supervisor before restarting the faulting instruction to ensure that the instruction is traced exactly once.

MI ROM sequences are treated as a single instruction for trace purposes so that the entire sequence executes before the trace trap is taken.

At the time of the trap, the CPU trap status field in the saved PSW indicates that a trace trap has occurred. The saved PC is the address of the instruction following the instruction that caused the trace trap.

6.2.7. Supervisor Calls

A supervisor call is an instruction executed as a trap, and is made using the **calls** instruction. Its purpose is to provide controlled access to system-level functions. There are 128 supervisor call codes, with separate Vector Table entries for each. The PC value saved on the stack is the address of the instruction following the **calls** instruction.

6.2.8. Multiple Traps

Only traps in the data memory and floating-point groups can be signalled at the same time. CLIPPER internal trap logic permits correct recovery of both faulting instructions. INTRAP transfers control to the floating-point trap handler, and the **loadfs** instruction can be used to access the floating-point instruction that caused the trap. The MTS field in the saved PSW may be read by the floating-point trap handler to determine which data memory trap occurred.

6.3. Interrupts

The CLIPPER Module supports 16 prioritized interrupt levels, with each level containing interrupt numbers of equal priority. Level 0 (highest priority) contains 15 numbers; levels 1-15 each contain 16 numbers. In addition to the 16 interrupt levels, there is a non-maskable interrupt which has a higher priority than all interrupt levels and cannot be disabled by software. Level 0 Number 0 vectors to the NMI interrupt handler.

The CPU contains the logic necessary to arbitrate interrupt requests according to the priority of the interrupt level. The interrupt level currently being processed is stored in the Interrupt Level (IL) field of the SSW. The CPU accepts interrupts only for interrupt levels of equal or higher priority than the current interrupt level.

Interrupts are serviced between instructions, that is, interrupt requests are not acknowledged until instructions in the lower pipeline have finished executing, any resulting traps have been serviced, and memory transactions have concluded. Thus, interrupts are not normally permitted during a macro sequence, which is considered a single instruction. However, some macro sequences (for example, the string instructions) permit interrupts periodically during their execution.

6.3.1. Maskable Interrupt Request/Acknowledge Protocol

Priority interrupts are requested by the activation of the $\overline{\text{IRQ}}$ input line and the assertion of the vector number on $\overline{\text{IVEC}}$<7:0>. The vector number includes the interrupt level on $\overline{\text{IVEC}}$<7:4> and the interrupt number on $\overline{\text{IVEC}}$<3:0>.

An interrupt request will be acknowledged by the CPU if interrupts are enabled (the interrupt enable flag in SSW is set) and the interrupt level ($\overline{\text{IVEC}}$<7:4>) is of equal or higher priority than the interrupt level contained in the SSW's Interrupt Level (IL) field. To maximize interrupt responsiveness following the assertion of $\overline{\text{IRQ}}$ and $\overline{\text{IVEC}}$, the interrupt level can change to higher priority on any BCLK until $\overline{\text{IRQ}}$ is released. See *Figure 14.*

The CPU samples $\overline{\text{IRQ}}$ on the rising edge of every BCLK if interrupts are enabled and the priority condition is met. The CPU then enters the pre-trap state, in which the execution pipeline is emptied by withholding issue of the instruction in the issue and control phase. The instructions in the execution pipeline complete executing; if their execution causes a trap to be signalled, the interrupt is deferred and the (higher priority) trap is serviced. The CPU then asserts $\overline{\text{IACK}}$.

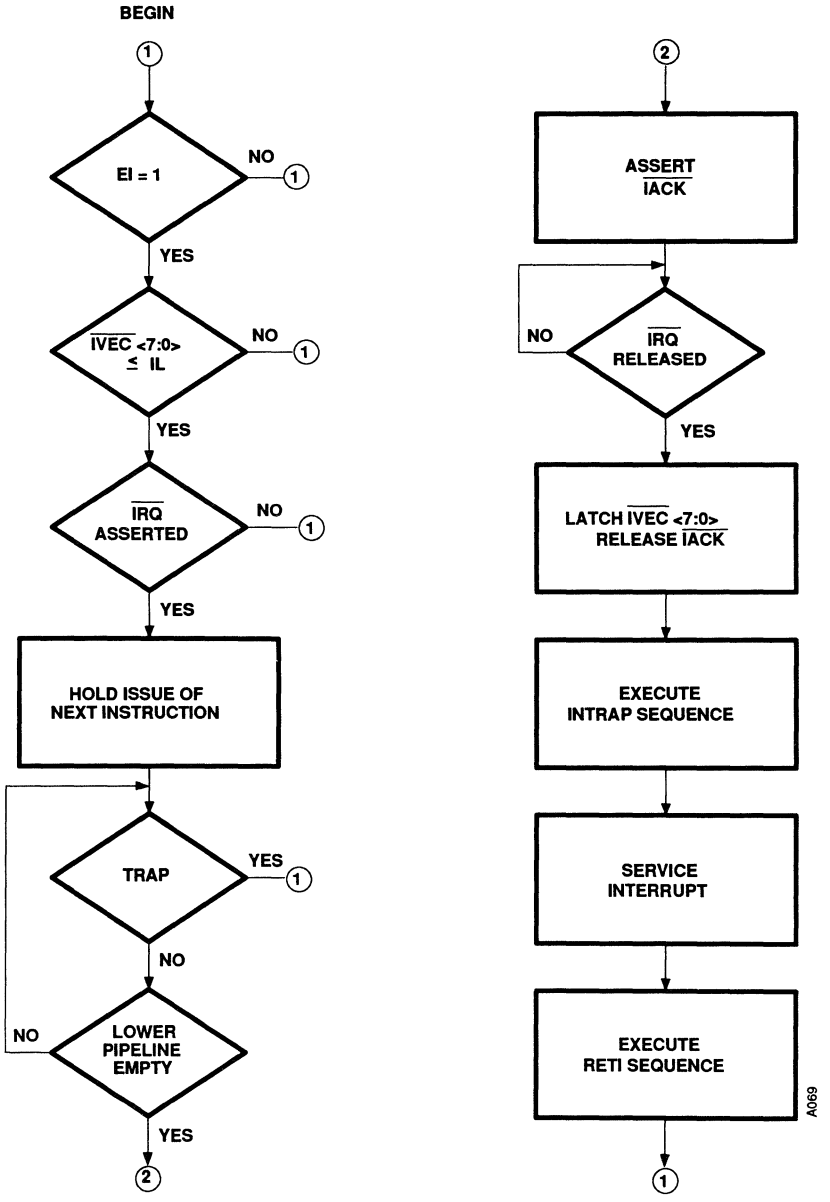
The CPU latches the interrupt number and level on the BCLK following the release of $\overline{\text{IRQ}}$, and releases $\overline{\text{IACK}}$ during the following BCLK.

The maskable interrupt request/acknowledge timing is shown in *Figure 59.* See also *Section 9.4.8, Interrupt Bus.*

6.3.2. Non-Maskable Interrupt

The non-maskable interrupt is signalled on the $\overline{\text{NMI}}$ input to the CPU which is sampled on the rising edge of every BCLK. An active low on $\overline{\text{NMI}}$ greater than the BCLK period will trigger this interrupt. $\overline{\text{NMI}}$ remains active until acknowledged by the CPU on $\overline{\text{NMIACK}}$. If $\overline{\text{NMI}}$ is asserted after another interrupt has already been acknowledged, the non-maskable interrupt is serviced after completion of the INTRAP sequence for the acknowledged interrupt. The $\overline{\text{NMI}}$ request/acknowledge timing is shown in *Figure 60.* See also *Section 9.4.8, Interrupt Bus.*

Figure 14 Interrupt Flow Diagram



7. Cache and MMU

The CLIPPER Module contains two Cache/Memory Management Unit (MMU) combination VLSI chips called CAMMUs which are designed to optimize CLIPPER performance.

Each CAMMU contains a 4 K-byte data cache, and a memory management unit which translates CPU 32-bit virtual addresses into 32-bit real addresses. One CAMMU is used for CPU instruction fetching and caching and is interfaced to the CPU Instruction Bus; the second CAMMU is used for CPU data transfers and caching and is interfaced to the CPU Data Bus. Both CAMMUs also interface to main memory and I/O devices via the CLIPPER Bus.

The two CAMMUs are functionally identical, but each is hardware programmed via an external chip pin for use as either an instruction CAMMU (I-CAMMU) or a data CAMMU (D-CAMMU).

The CAMMUs feature several caching policy and Bus Watch options which allow optimum performance tailored to specific applications. A prefetch option is available for the I-CAMMU; and fixed address translation is used in both the I-CAMMU and the D-CAMMU for guaranteed access of selected locations in main memory, Boot, and I/O spaces. In addition, CAMMU internal registers and register fields are easily accessed for efficient CAMMU configuration and control.

7.1. Functional Overview

The two main functional units of the CAMMU are the cache and the Memory Management Unit (MMU), with the MMU comprised of the Dynamic Translation Unit (DTU), the Translation Lookaside Buffer (TLB), and the Hardwired Translation Lookaside Buffer (HTLB) (see *Figure 15*). The CAMMU also utilizes a cache control unit which controls CAMMU data fetches from main memory.

Figure 15 CAMMU Interface

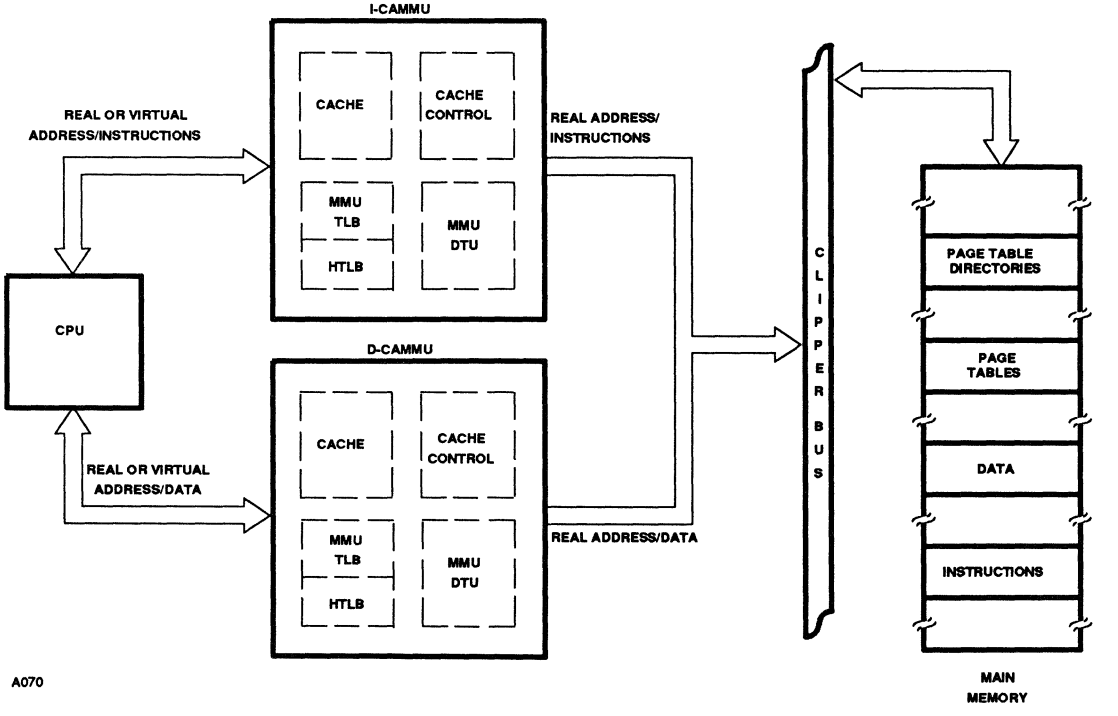
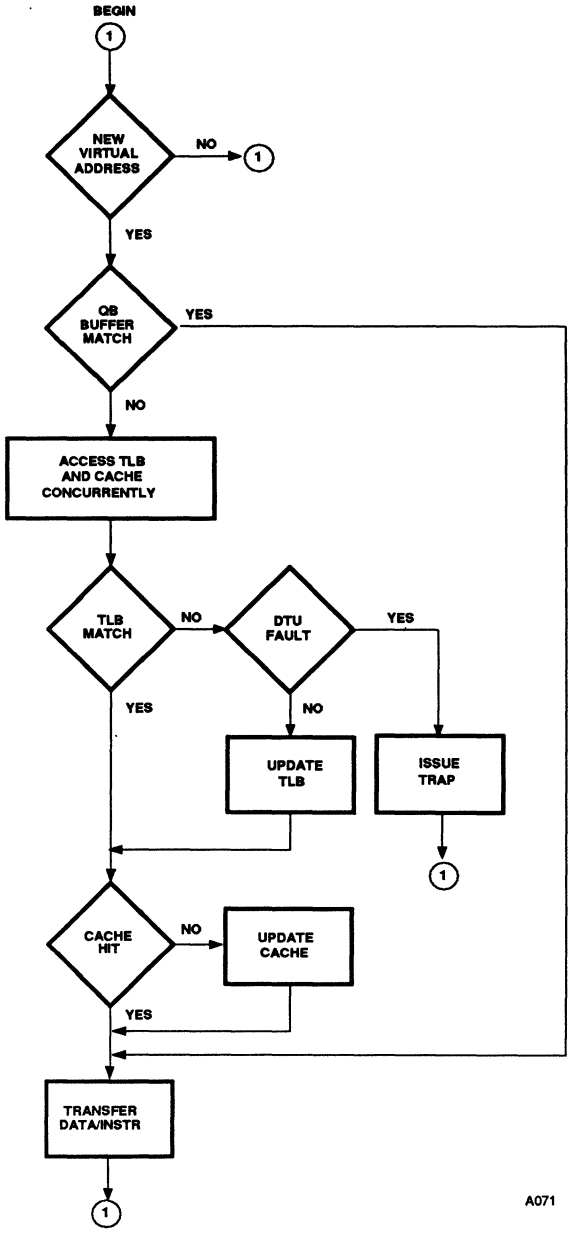


Figure 16 Basic CAMMU Functional Flow



CAMMU operation begins when the CPU asserts a virtual address on the CPU-CAMMU address/data bus. The task of the CAMMU is to translate the CPU virtual address (bits <31:12>) into a real address and to use the translated real address to find the data.

The CAMMU compares the virtual address with a virtual address of the data stored in a 16-byte (Quadword) Buffer containing the most recently accessed cache line. If there is a match, the data is fetched directly from the Quadword Buffer and no additional cache or TLB action is performed. If there is no Quadword Buffer match, the CAMMU attempts to translate the address by using the TLB, which is a look-up table containing Virtual Address Tags and associated Real Address fields which point to locations in the cache. If the CAMMU finds a Virtual Address Tag in the TLB which matches the CPU virtual address, it compares the associated Real Address field in the TLB with the Real Address fields of a cache line set, already selected by virtual address bits <10:4>, to determine whether the data is in the cache. If the data is not in the cache, the CAMMU accesses main memory for the data.

If the CAMMU cannot find a matching Virtual Address Tag in the TLB, it invokes the DTU to search declared blocks of main memory (Page Directory Tables and Page Tables) in an attempt to translate the virtual address.

The DTU, upon successful translation of the virtual address, updates the TLB with the new Virtual Address Tag/real address association. The CAMMU then continues with data access. If the DTU cannot find the valid translation in main memory, the CAMMU asserts a CPU page fault trap for resolution by the operating system.

Each CAMMU cache consists of 256 quadwords of data (4 K-bytes) with associated Real Address Tags in a configuration similar to the TLB. The CAMMU searches the Quadword Buffer and the onboard cache first, then main memory for addressed data locations if required.

A basic logic flow of CAMMU operation is shown in *Figure 16.*

Figure 17 CPU Virtual Address Format

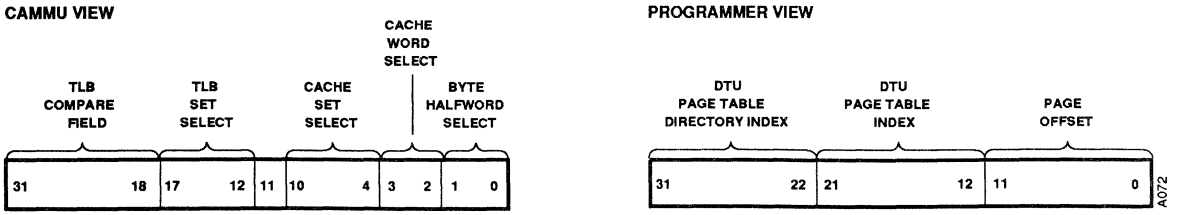


Figure 18 Simplified CAMMU Block Diagram

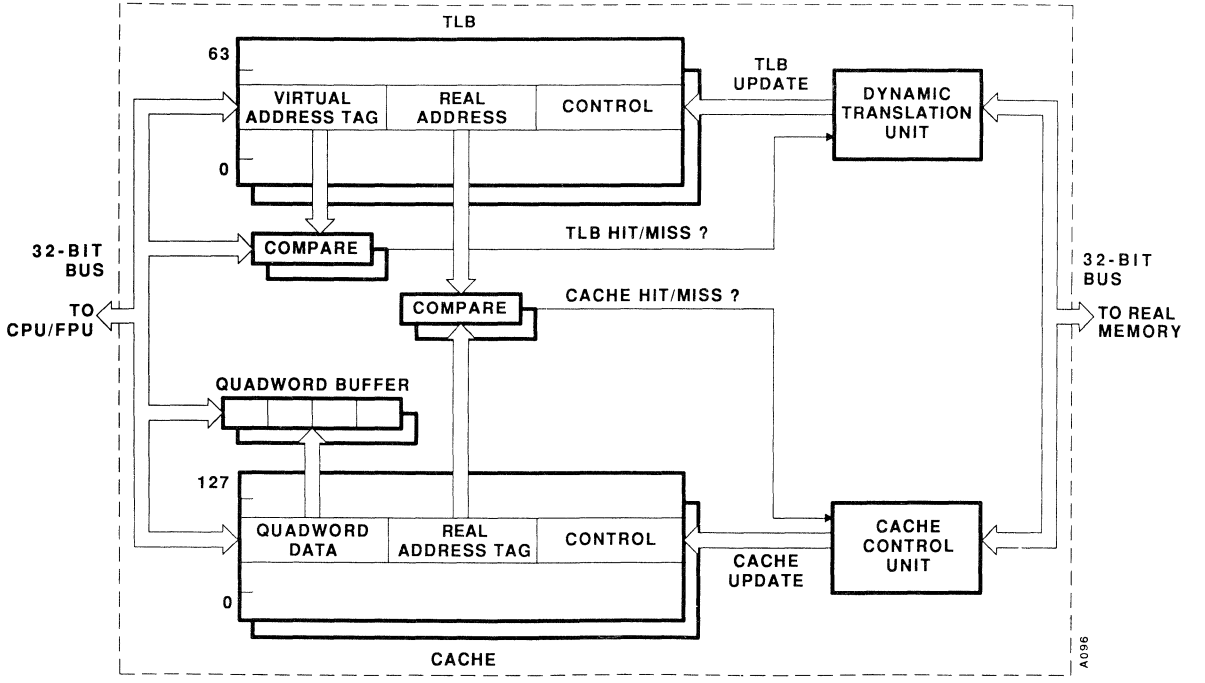


Figure 17 depicts the format of the CPU virtual address and indicates how the various virtual address fields are used by the CAMMU. *Figures 18* and *19* show CAMMU operation. These figures should be referred to while reading the following CAMMU descriptions.

7.2. Memory Management Unit (MMU)

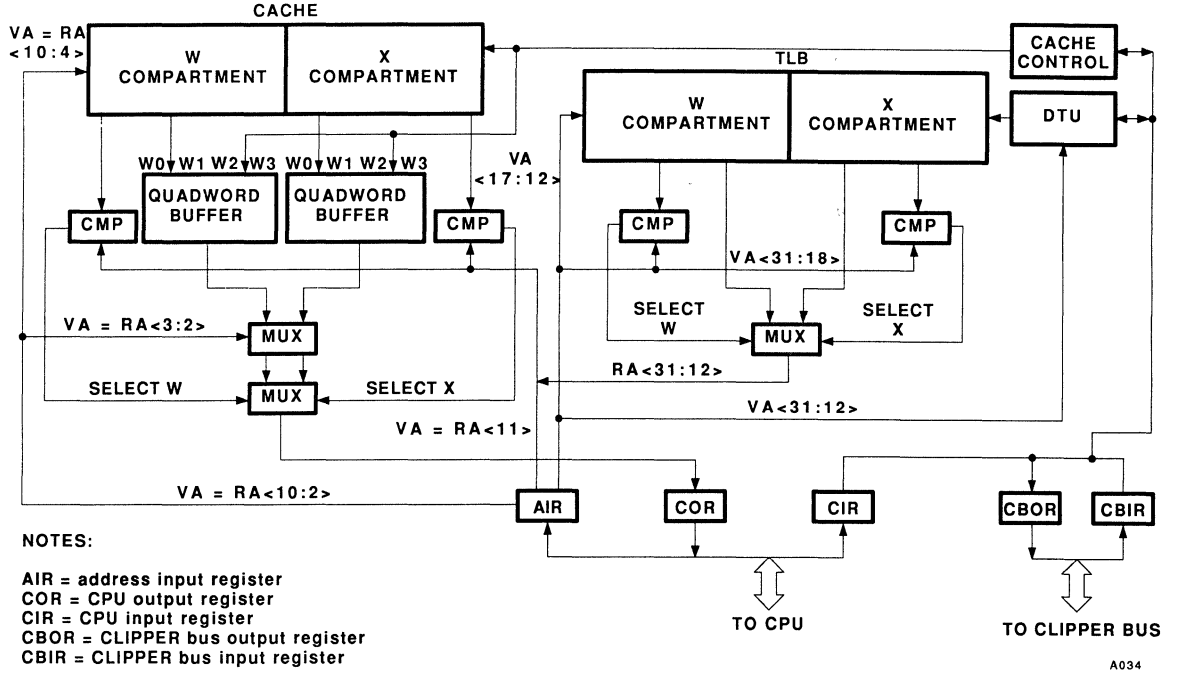
The Memory Management Unit translates CPU virtual addresses into real addresses and supports address space access protection by the operating system on a per-page basis.

Address translation is executed by three functional units within the MMU: the Translation Lookaside Buffer (TLB), the Hardwired TLB (HTLB), and the Dynamic Translation Unit (DTU). Address space access protection and memory management support are performed by logic within the MMU which utilizes system tags and protection codes associated with the virtual memory pages.

7.2.1. Translation Lookaside Buffer (TLB)

The TLB is a two-way set-associative memory array that is used by the CAMMU for fast, on-board virtual ad-

Figure 19 CAMMU Block Diagram



dress to real address translation. It consists of 64 sets of lines, with each set consisting of a W and an X compartment line, and an associated U flag (see *Figure 20*). The CAMMU uses bits <17:12> of the CPU virtual address to select a TLB line set, then compares bits <31:18> of the virtual address with the VA (Virtual Address) Tag of both the W and X Compartment lines of the selected set.

If there is a match, and if the appropriate access protection code allows the data/instruction access, the 20-bit RA (Real Address) field of the matching W or X line is multiplexed and transferred to the cache as real address bits <31:12> where they, along with virtual/real address bit 11 (this bit is not translated), are used to validate the data.

If there is no TLB match, the DTU attempts address translation, as explained in *Section 7.2.3, Dynamic Translation Unit.*

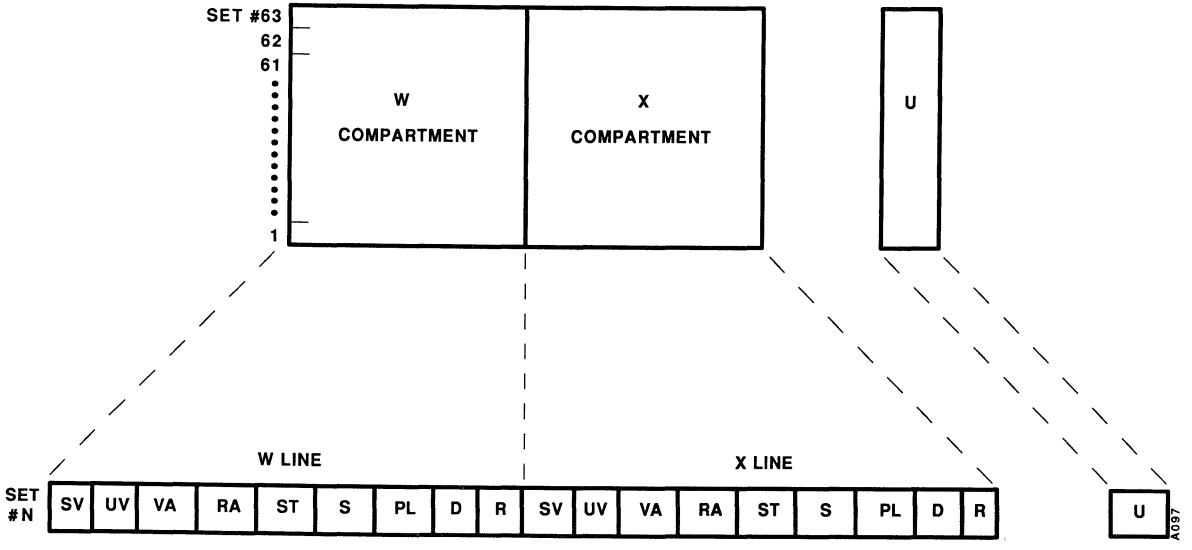
TLB Line Description
TLB line format is shown in *Figure 21.* Equivalent RA, ST, PL, D and R flags are located in the Page Tables. The CAMMU ensures that the D and R flags in the Page Table entries are updated with flag changes in the TLB.

The TLB line field definitions are as follows:

SV: Supervisor Valid
The SV flag when set indicates that the TLB line is used for address translation only during supervisor mode operation. All TLB SV flags can be cleared as a group by writing to the CAMMU Reset Register and by CLIPPER Module hardware reset.

UV: User Valid
The UV flag when set indicates that the TLB line is used for address translation only during user mode operation. All TLB UV flags can be cleared as a group

Figure 20 CLIPPER TLB



by writing to the CAMMU Reset Register and by CLIPPER Module hardware reset.

VA: Virtual Address Tag
This 14-bit field is used for W or X line selection once TLB set selection is complete. The VA Tag of each line is compared with CPU virtual address bits <31:18>. If there is a match, the matching line is used for the address translation.

RA:Real Address
This 20-bit field is used as real address bits <31:12> once the TLB line has been matched, and access protection and validation checks have been completed (see UV, SV, and PL descriptions).

ST:System Tag
This is a three-bit field which identifies the caching type, caching policy, and address space associated with the page referenced by the TLB line as follows:

ST<2:0>	Description
0	private, write-through, main memory space
1	shared, write-through, main memory space
2	private, copy-back, main memory space
3	noncacheable, main memory space
4	noncacheable, I/O space
5	noncacheable, Boot space
6	cache purge
7	noncacheable, main memory space, slave I/O mode

This field is used only in mapped mode. In unmapped mode, the UST (Unmapped System Tag) field in the CAMMU Control Register is used, as described in *Section 7.6.4.*

The System Tag is asserted on CLIPPER Bus lines TG<2:0> during CAMMU external accesses. Further information on the System Tag field is provided in *Section 7.4, System Tag.*

S: System Reserved
This is a general-purpose, two-bit field reserved for use by the operating system.

PL: Protection Level
Associated with each virtual address is a function code asserted by the CPU which identifies each memory reference as a read, write, or instruction fetch operation, and indicates the states of the U, UU, K, and KU flags in the CPU's SSW. These SSW flags indicate whether the memory access is by the supervisor or by a user, which protect key is to be used for access verification (K or KU), and the state of the key. The CAMMU compares the function code with the 4-bit PL field of a selected TLB line to determine whether read access, write access, and instruction fetching is allowed.

The Protection Level fields are used only for CPU mapped addresses (addresses asserted while the mapped mode bit of the SSW is set). Unmapped addresses invoke no access protection.

Figure 21 TLB Line Format and Description

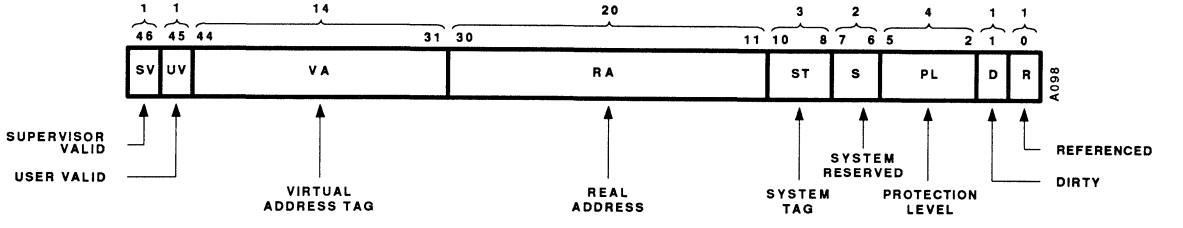


Table 10 shows allowed accesses according to the SSW's K, U, UU, and KU flags, and the PL field.

D: Dirty Flag
The Dirty flag is set by the CAMMU to indicate that the 4 K-byte page in main memory referenced by the TLB line has been altered. Typically the operating system uses this flag to determine whether the referenced data page must be copied to secondary storage (such as a hard disk) when the data in the page is replaced.

R: Referenced Flag
The CAMMU sets the R flag to indicate that the page associated with the line has been accessed. Typically the operating system uses this flag as part of a main memory page replacement algorithm by periodically clearing all the TLB R flags via the Reset Register (see *Section 7.6.5, Reset Register*), then allowing them to be set during normal program execution. When the operating system replaces a main memory page, it selects an "unreferenced" page for replacement based on the states of the R flags.

U: Used Flag
Associated with each TLB line set is a U (Used) flag which is set by the CAMMU to indicate that the W line of the set was last accessed and cleared to indicate that the X line was last accessed. When a new line has to be entered into the TLB as a result of a TLB miss, the least recently used line in the selected set is replaced based on the state of this flag.

Table 10 Page Access Encoding

	Supervisor Mode (U=0)				User Mode (U=1)	
	UU=0 D- and I-CAMMU		UU=1 D-CAMMU Only		D- and I-CAMMU	
PL	K=1	K=0	KU=1	KU=0	K=1	K=0
0	RW	-	-	-	-	-
1	RW	RW	-	-	-	-
2	RW	RW	RW	-	RW	-
3	RW	RW	RW	RW	RW	RW
4	RW	RW	RW	R	RW	R
5	RW	RW	R	R	R	R
6	RW	R	R	R	R	R
7	RWE	RWE	RWE	RWE	RWE	RWE
8	RE	-	-	-	-	-
9	R	RE	-	-	-	-
10	R	R	RE	-	RE	-
11	R	R	RE	RE	RE	RE
12	-	RE	-	RE	-	RE
13	-	-	RE	-	RE	-
14	-	-	-	RE	-	RE
15	-	-	-	-	-	-

Notes:
- No Access Allowed
R Read Only Allowed
RW Read and Write Allowed
RE Read and Execute Allowed
RWE Read, Write, and Execute Allowed

7.2.2. Fixed Address Translation

The CAMMUs feature hardwired TLB lines which ensure TLB hits of special memory pages by both mapped and unmapped addresses while the CPU is executing in supervisor mode. These hardwired entries eliminate page faults during INTRAP and **reti** sequences, and allow access of Boot and I/O space before the Page Table Directories and Page Tables have been initialized during system booting.

The hardwired TLB (HTLB), implemented in random logic and not visible to software, contains the functional equivalents of the VA, RA, ST and PL fields found in the writable TLB. However, equivalents to other TLB fields are not used in the HTLB. The HTLB can be accessed only during CPU supervisor mode, so UV and SV flags are not required. HTLB lines cannot be replaced, so no Used flags are required. Pages referenced by the HTLB are dedicated pages not subject to general replacement by the operating system, so no Referenced or Dirty flags are required.

HTLB Mapping
The hardwired TLB is invoked only when supervisor pages 0-7 are addressed by the CPU. With the exception of CAMMU I/O space (the upper half of page 0 of I/O space), these pages can also be mapped through the writable TLB. Virtual pages other than supervisor pages 0 - 7 must be mapped through the writable TLB.

When a CAMMU detects a supervisor page 0 - 7 virtual address, it selects the Real Address, System Tag, and Protection Level fields from the appropriate HTLB line. Pages 0 - 3 and 6 - 7 are protection free, allowing read, write and execution accesses. Pages 4 and 5 allow only read and write access, and attempted execution of Test and Set instructions in these pages results in a protection fault. The real address translation and system tag assigned to each page is shown in *Table 11*.

The address space assigned to each of the virtual pages is also shown in *Table 11*. These address space assignments are determined by the System Tag, which is asserted on CLIPPER Bus lines TG<2:0> during CLIPPER Bus access. These lines function as main memory, I/O and Boot space selects and must be decoded by system hardware for proper device selection.

Figure 22 contains a pictorial overview of Hardwired TLB mapping showing the three distinct real address spaces into which virtual pages 0 - 7 are mapped.

Virtual Page 0 - 7 Assignments
Three of the "hardwired" virtual pages are available for general use. The other five are dedicated for specific purposes. These page assignments are shown in *Table 11*.

The CPU fetches interrupt and trap vectors from supervisor virtual page 0. The CAMMUs translate this page

Figure 22 Hardwired TLB Mapping

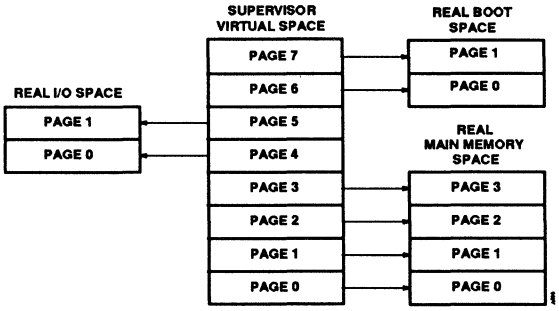


Table 11 Hardwired TLB Address Translations

Virtual Page Number	Real Page Number	Page Assignment	Protection Level (U=0)	System Tag	
				ST, TG	Description
0	0	General-Purpose and Interrupt and Trap Vectors	7	1	shared, write-through, main memory space
1	1	General-Purpose	7	2	private, copy-back, main memory space
2	2	General-Purpose	7	3	noncacheable, main memory space
3	3	General-Purpose	7	3	noncacheable, main memory space
4	0	I/O, CAMMUS and Reserved	3	4	noncacheable, I/O space
5	1	I/O	3	4	noncacheable, I/O space
6	0	General-Purpose	7	5	noncacheable, Boot space
7	1	General-Purpose	7	5	noncacheable, Boot space

Note: The ST field is decoded by the CAMMU during page access. The bits are transferred to the CLIPPER Bus lines TG<2:0> during CLIPPER Bus access.

into page 0 of main memory where the vectors must be located.

Similarly, following CLIPPER reset, the CPU fetches initial boot code from virtual page 6, which the CAMMUs translate into page 0 of Boot space. The CAMMUs also translate virtual page 7 into Boot space (page 1) to allow a total of 8 K-bytes of HTLB-translated Boot addresses.

The first Boot instructions must be located at supervisor virtual address 6000 Hex, which translates to address 0 of Boot space. The rest of the boot code should be located in pages 0 and 1 of Boot space as required.

Virtual page 4 is reserved by Intergraph for CAMMU internal register addressing and for future use (see *Section 7.6, Internal Registers*). Virtual page 5 is available for I/O. The D-CAMMU translates these virtual pages into pages 0 and 1 of real I/O space. Attempted access of virtual pages 4 or 5 for instruction fetch results in a protection fault.

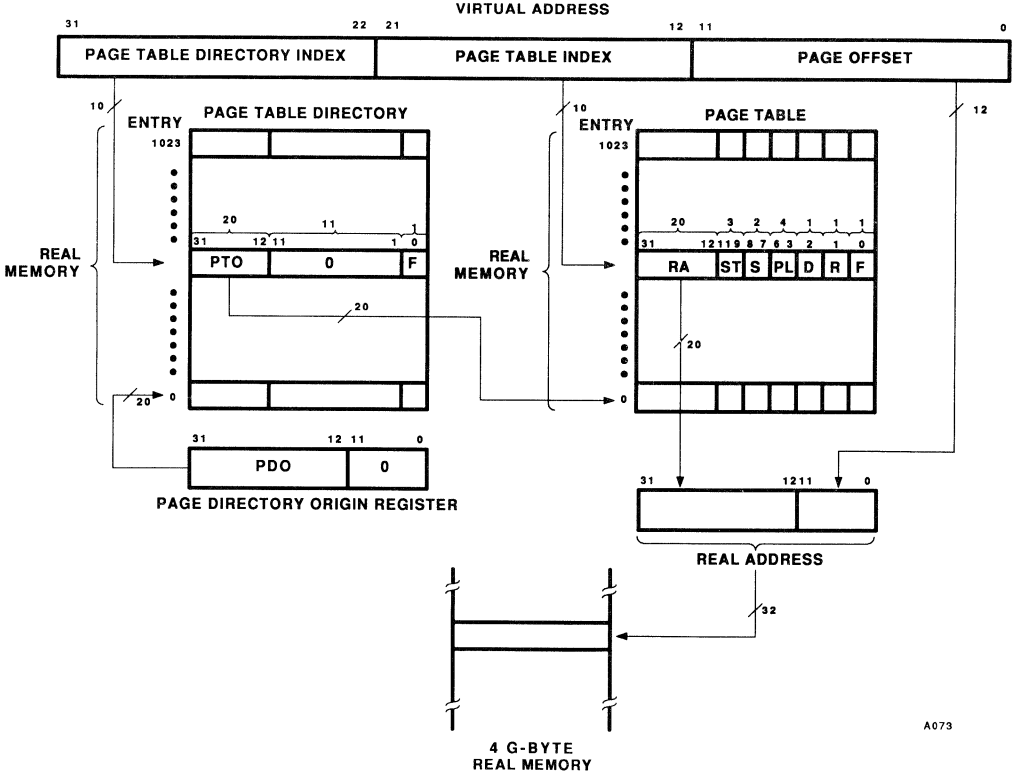
Additional pages can be created in Boot or I/O space by assigning the appropriate System Tag (5 or 4) to virtual pages other than 0 - 7. Translation of these pages, however, is by the writable TLB or the DTU, not by the Hardwired TLB.

Virtual pages 1- 3 are general-purpose pages which are translated into main memory pages 1- 3 by the HTLB.

7.2.3. Dynamic Translation Unit (DTU)

The DTU translates virtual address bits <31:12> into real address bits <31:12> in two steps. First it fetches a Page Table Origin from a Page Table Directory located

Figure 23 DTU Virtual Address Translation



in main memory; then it fetches an entry from a Page Table, also located in main memory. This sequence is depicted in *Figure 23.*

Once the DTU has completed address translation, the CAMMU updates the TLB and provides the real address to validate the cache data, then searches the cache for the data. If the data is not cached, the CAMMU accesses main memory for the data using the concatenation of translated address bits <31:12> and untranslated virtual address bits <11:0>.

Note that because the DTU accesses only main memory and not the cache during address translation, all Page Table Directories and Page Tables must be located in noncacheable pages (see *Section 7.4, System Tag*).

Page Table Directory Entry Selection
Two 20-bit PDO (Page Directory Origin) registers each contain the base address of a Page Table Directory. One PDO register is used by the CAMMU during supervisor mode operations; the other PDO register is used during user mode operations. The DTU concatenates bits <31:22> of the virtual address with the contents of the appropriate PDO register to form the most significant 30 bits of a Page Table Directory entry address. (Page Table Directory entries are word-aligned; therefore bits <1:0> are forced LOW.) In effect, the PDO register points to the Page Table Directory, and bits <31:22> of the virtual address select one of 1024 Page Table Directory entries.

Page Table Directory Format
Each Page Table Directory consists of 1024 32-bit words located in main memory. Page Table Directory entries (see *Figure 24*) are comprised of two fields.

PTO: Page Table Origin
This field is used by the DTU during address translation to locate the Page Table in main memory.

F: Page Fault
The F flag is set or cleared by the operating system to indicate the absence or presence of a valid Page Table pointed to by the PTO field in the entry. A set F flag indicates absence of a valid Page Table, and attempted DTU address translation with a Page Table directory entry having a set F flag forces a CPU page fault trap.

Page Table Entry Selection
The selected Page Table Directory entry contains a 20-bit PTO field (see *Figure 24*) which holds the base address of a Page Table that is to be used for address translation. The DTU concatenates bits <21:12> of the virtual address with the contents of the PTO field to form bits <31:2> of the Page Table entry address (bits <1:0> are forced LOW). In effect, the PTO field points

Figure 24 Page Table Format

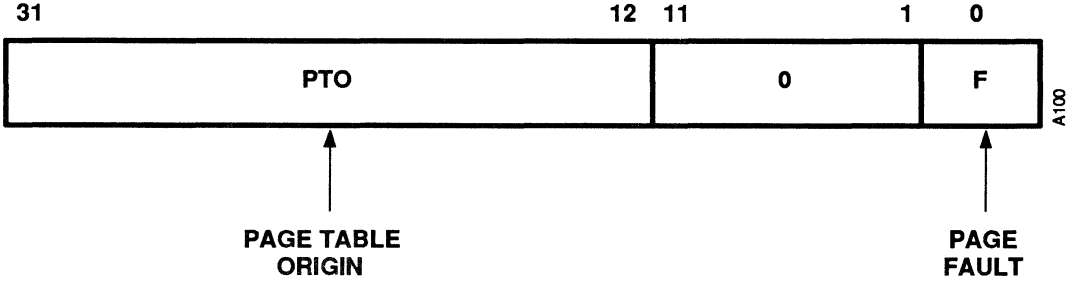


Figure 25 Page Table Entry Format

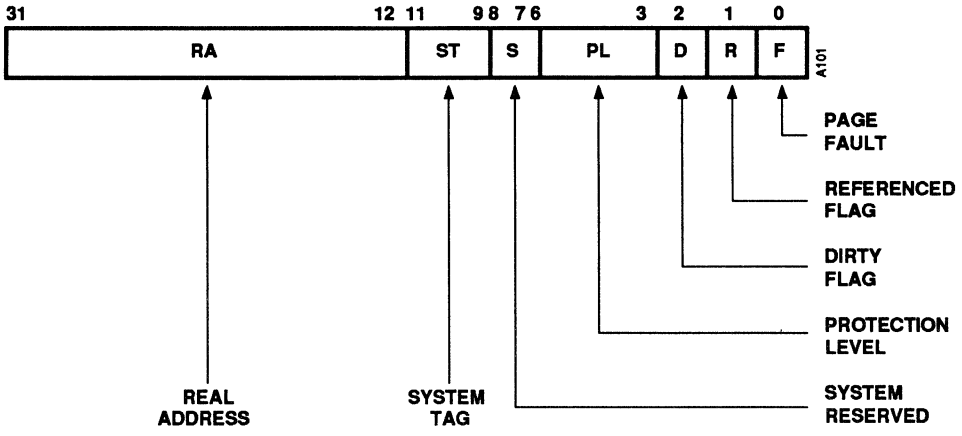
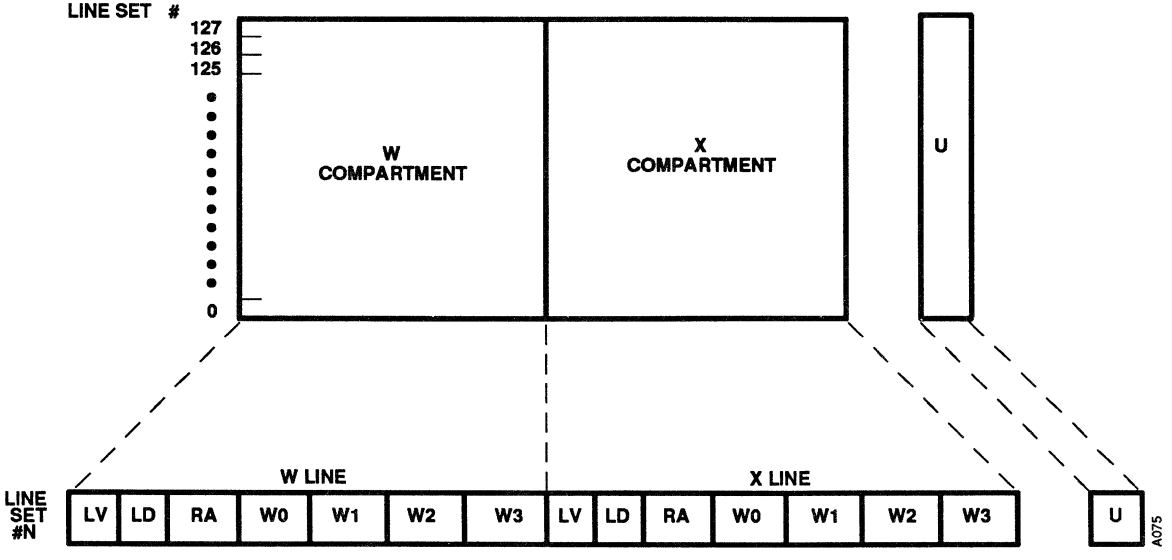


Figure 26 Cache Set-Associative Memory Array



to the appropriate Page Table, and bits <21:12> of the virtual address select one of 1024 Page Table entries.

The CAMMU then uses the 20-bit RA (Real Address) field in the Page Table entry, shown in *Figure 25*, as bits <31:12> of the real address.

Page Table Format
Each Page Table consists of 1024 32-bit words comprised of six fields, as shown in *Figure 25*. Equivalent ST, S, PL, D, and R fields are located in the TLB registers. See *Section 7.2.1, Translation Lookaside Buffer*.

The following are the Page Table entry field descriptions:

RA: Real Address
The 20-bits of the RA field are used as real address bits <31:12> following address translation. These bits constitute a 4 K-byte page address.

ST: System Tag
This field identifies the caching policy, caching type, and address space of the page (see *Section 7.4, System Tag*).

S: System Reserved
This is a general-purpose 2-bit field reserved for the operating system.

PL: Protection Level
The CAMMU uses this field to determine whether data read, data write, and instruction fetching are allowed to/from the page (see *Section 7.2.1, Translation Lookaside Buffer*).

D: Dirty Flag
The Dirty flag is set by the CAMMU to indicate that the page has been altered.

R: Referenced Flag
The CAMMU sets the R flag to indicate that the page has been accessed.

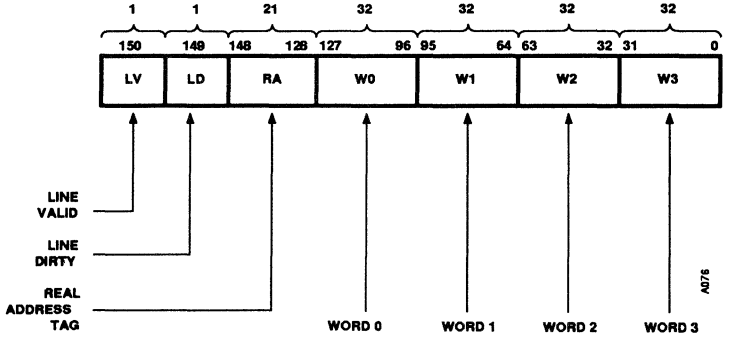
F: Page Fault
The F flag is set/cleared by the operating system to indicate the absence/presence of a valid page. A set F flag indicates absence of a valid page, and attempted DTU address translation with a Page Table entry having a set F flag forces a CPU page fault trap.

7.3. Cache
The cache architecture is similar to that of the TLB, as shown in *Figure 26*. It is a 4 K-byte cache composed of 128 sets of lines, with each set consisting of a W compartment line and an X compartment line.

Associated with each cache line set is a U bit which is set to indicate that the W line of the entry was last accessed, and cleared to indicate that the X line was last accessed. When, as a result of cache miss, a new data quadword has to be cached, the least recently used line in the selected line set is replaced based on the state of this bit.

7.3.1. Cache Line Description
Figure 27 shows the cache W and X compartment line format. Each line consists of four 32-bit data words and LV, LD, and RA fields defined as follows:

Figure 27 CLIPPER Cache Line Format



LV: Line Valid
The LV bit, when set, indicates that the data in the associated line is valid; when clear, it indicates that the data is invalid.

A line LV bit is set by the CAMMU when it loads new data into the cache line. The bit is cleared by the CAMMU, operating as a slave, in response to CLIPPER Bus activity when its Bus Watch modes are enabled, or by a cache purge operation (TG = 6) as described in *Section 7.4.2, System Tag 6.* Individual LV flags are also cleared by hardware when the CV (Clear Valid) bit in the CAMMU Control Register is set, and the cache provides more current (dirty) data for a quadword I/O Read (see *Section 7.5, Bus Watch Modes, Watch I/O Reads*).

In the case of Bus Watch, LV is cleared during a quadword write to a main memory address that matches the particular cache location. During the Bus Watch operation, the CAMMU asserts the $\overline{\text{CBSY}}$ signal on the CLIPPER Bus. The $\overline{\text{CBSY}}$ signal prevents another bus transaction from beginning until Bus Watch operation has completed. If the CPU has addressed this same cache line prior to the CLIPPER Bus's write operation, the CPU has priority and the bit is not cleared until the CPU access is completed. This is described in more detail in *Section 7.5, Bus Watch Modes.*

All cache LV flags can be cleared as a group by writing to the CAMMU Reset Register (see *Section 7.6, Internal Registers*), and by CLIPPER hardware reset.

LD: Line Dirty
The LD bit is set by the CAMMU to indicate that data in the cache line has not been updated in main memory. This occurs when the CAMMU is operating in the copy-back mode (see *Section 7.4, System Tag*), and a CPU write to memory results in a cache hit. In this case, the data is written to the cache but not to main memory, resulting in "dirty" cache data, i.e., data which is more current than main memory data. This bit is cleared by the CAMMU when the copy-back data is copied back to memory.

RA: Real Address Tag
The RA tag is used for cache line selection. The RA tags of both the W and X compartment lines are compared with translated address bits <31:12> and bit 11 of the virtual address. Accessed data is located in a matching line.

W0-W3: Word 0 - 3
Words 0 - 3 are the four 32-bit data words in the cache line.

7.3.2. Cache Data Selection

Virtual address bits <11:2> are used directly by the CAMMU as real address bits <11:2> to access cache data (refer to *Figure 19*).

The CAMMU uses address bits <10:4> to select one of the 128 cache line sets. The CAMMU compares the concatenation of translated address bits <31:12> and address bit 11 with the RA field of both the W line and the X line of the selected line set. If there is no match, a cache miss has occurred, and the CAMMU accesses main memory for the data transfer. If there is a match, the CAMMU uses address bits <3:2> to select one of the four data words in the matching line for the data transfer.

7.3.3. Prefetch

The D-CAMMU implements a "demand" data fetching algorithm. Data fetching for the cache is "on demand" by the CPU; that is, a new data quadword is fetched from main memory only as a result of a cache miss.

The I-CAMMU also supports demand fetching which functions identically to D-CAMMU data fetching, but features an optional prefetching algorithm not available in the D-CAMMU. This algorithm prefetches the next four words of instructions from main memory for line N + 1 of the cache when line N has been accessed by the CPU.
I-CAMMU prefetching is controlled by bit 0 of the CAMMU Control Register. When bit 0 is clear, prefetch is disabled; when set, prefetch is enabled.

I-CAMMU prefetch enable/disable should be based on the general structure of the code being executed. If the instructions are in general executed sequentially as stored in main memory, the probability of cache hits of prefetched instructions is high; therefore, prefetch should be enabled for increased CPU throughput. If the instructions are branch intensive, the probability of

cache hits of prefetched instructions is reduced; therefore, prefetch may be disabled to reduce system bus traffic.

7.3.4. Quadword Data Transfers

The CAMMU cache lines each contain four words. The CAMMUs transfer data/instructions between the caches and main memory four words (one quadword) at a time. (Single-word transfers are used for data/instructions in noncacheable pages.)

7.4. System Tag

Associated with each virtual page is a three-bit ST (System Tag) field which determines the caching policy that applies to the page, the page caching type (private or shared), and the page's address space (I/O, Boot, or main memory). In addition, the System Tag can be used to identify two special operations: Cache Purge and Slave I/O mode. This field is found in the Page Table entries (see *Figure 25*) and in the TLB (see *Figure 21*).

The System Tags for supervisor pages 0 - 7 are hardware-selected by the CAMMU (see *Section 7.2.2, Fixed Address Translation* and *Table 11*). All other virtual address page System Tags are assigned by the operating system according to system requirements when it creates the Page Tables. The CAMMUs decode the ST fields during address translation and transfer the bits to the CLIPPER Bus lines TG<2:0> during CLIPPER Bus access. (If the system is being operated in unmapped mode, the UST field (Unmapped System Tag) in the CAMMU Control Register determines the System Tag.)

7.4.1. System Tags 0 - 5

System Tags 0 - 5 encode the following information about a virtual page:

(1) Address space— main memory, Boot, or I/O
(2) Caching type—private or shared.
(3) Caching policy—cacheable or noncacheable; write-through or copy-back.

Address Space

"Address space" identifies the real address space of the page as either main memory space, Boot space or I/O space. The CAMMUs map all virtual addresses into one of these spaces.

Caching Type

Two types of page caching are recognized by the CAMMU: private and shared. Private caching pages are accessed and cached by one CAMMU only. Shared caching pages are accessed and cached by more than one CAMMU. (Pages that are cached by both the I-CAMMU and D-CAMMU of a CLIPPER Module are shared pages.)

Note that the terms "shared" and "private" relate only to CAMMU access. In fact, a page may be private to a CAMMU but accessible by non-CAMMU devices. This page, though private, is common to one or more devices other than the CAMMU, and should therefore receive special consideration when assigning System Tags.

Caching Policy

Caching policy is a set of attributes assigned to a page by the System Tag which identifies the page as cacheable or noncacheable, and, if cacheable, defines the caching mode which applies to the page as copy-back or write-through. Combinations of write-through or copy-back caching to private pages, and write-through caching to shared pages are possible.

Cacheable data can be entered into a cache; noncacheable data cannot be entered into a cache. Pages can be tagged as cacheable or noncacheable according to system requirements.

Write-through and copy-back are two schemes for updating main memory with data in the D-CAMMU cache. Selection of these modes is based on system requirements and page caching type. Private pages may be write-through or copy-back; shared pages must be write-through.

During a CPU write, the CAMMU searches the cache for the accessed location. If the location is not in the cache (a cache miss), the CAMMU operates according to the caching mode as follows:

(1) Write-through—the CAMMU updates main memory with the CPU data but does not update the cache because the data is not cached.

(2) Copy-back—the CAMMU reads the quadword containing the addressed data from main memory into the cache, then updates the cache, but does not update main memory.

If the data is in the cache (a cache hit), the CAMMU operates according to caching mode as follows:

(1) Write-through—the CAMMU updates both the cache and the main memory.

(2) Copy-back—the CAMMU updates only the cache, but does not update main memory.

Write-through mode forces the D-CAMMU to access the CLIPPER Bus and update main memory immediately following a cache write. Write-through mode thus ensures that main memory data is current with the cache at all times. Shared pages (those that can appear in more than one cache) must be write-through.

Copy-back mode inhibits updating of main memory with the new data until the cache line written into is replaced. When a copy-back write hit to the cache occurs, the LD (Line Dirty) flag in the hit line is set to indicate that the line data must be written to main memory when the line is replaced. Since copy-back mode does not assure data consistency between main memory and the cache at all times, copy-back mode cannot be used for pages that are shared by another CAMMU.

Write-through mode eases the task of data management because main memory is always "up to date" but increases CLIPPER Bus traffic because the CAMMU must access the bus following each cache write. Copy-back requires more careful data management consideration but decreases system bus traffic, thereby significantly improving system performance. These factors should be considered when assigning System Tags.

7.4.2. System Tag 6—Cache Purge

System Tag 6, Cache Purge, forces invalidation (purging) of cache lines that are hit during CPU write operations that are tagged TG = 6. The lines are invalidated by clearing of the Line Valid (LV) bits.

A CPU write with the ST field in the TLB set to a 6 will purge hit cache lines of its own caches. The write proceeds as normal with TG lines = 6 on the CLIPPER Bus, causing hit lines in other module caches (having Watch CPU writes enabled) to be purged. Any cache with Watch CPU writes enabled will purge hit lines (regardless of the state of the TLB system tag field) when a write is detected on the CLIPPER Bus with TG = 6.

The Cache Purge feature facilitates the re-use of pages by allowing invalidation of data belonging to a replaced page which is left in a cache. In multiple CLIPPER Module applications, for example, a page might be replaced in main memory which may leave unpurged data in a cache of the module not initiating the page replacement. The CAMMU initiating the page replacement can invalidate that cache data by writing to the cached data locations using the Cache Purge tag.

7.4.3. System Tag 7—Slave I/O

System Tag 7, Slave I/O Mode, in effect allows the module to act as a DMA controller. It supports transfer of data between I/O and main memory in DMA-like fashion, but is not intended to replace DMA controllers.

During Slave I/O operation, the CLIPPER Module accesses an individual word, halfword, or byte from a source (such as main memory) which is simultaneously read by a destination device. Both read and write operations can be tagged Slave I/O mode.

Slave I/O read operations are used to transfer data from main memory to an I/O device. The CLIPPER Module executes a read from memory with TG = 7, and the memory responds with the data which is read by the I/O but can be ignored by the CPU. The I/O must recognize TG = 7 as Slave I/O mode and assert RDYi to terminate the operation. Timing for the Slave I/O read operation is the same as for a standard read.

Slave I/O write operations are used to transfer data from I/O to main memory. The CLIPPER Module executes a write to memory with TG = 7 using arbitrary data that will not be asserted on the bus. The I/O must recognize TG = 7 as slave I/O mode and assert data on the bus. The main memory asserts RDYi to terminate the operation.

Timing for the Slave I/O write operation is the same as for a standard write, with the exception that $\overline{DIR}$ transitions after the address phase of the operation as if for a read. This prevents the CLIPPER Module bus transceivers from driving the bus during the data transfer phase of the operation, allowing the I/O to send data without bus contention.

7.5. Bus Watch Modes

Bus Watch modes are used by the CAMMU to ensure data consistency between the cache and main memory, and to transfer the "latest" data to an external device reading main memory.

When Bus Watch is enabled in a CAMMU, it monitors main memory accesses by other bus masters. Depending on the Bus Watch mode enabled and the type of main memory access (identified by decoding the CLIPPER Bus TG<2:0> and CT<5:0> lines, as described in *Section 9, CLIPPER Bus*), the CAMMU intervenes to update the cache with data that is written to main memory, to invalidate cache data, or to transfer updated data from the cache to a bus master that is reading main memory. This Bus Watch monitoring occurs in parallel with the memory access in order to eliminate or minimize the Bus Watch operation effect on CLIPPER Bus throughput. Each CLIPPER Bus master must generate the appropriate TG<2:0> tag when accessing the CLIPPER Bus.

The three Bus Watch modes featured by the CAMMU are:

(1) Watch CPU Writes (CPU writes to shared cacheable pages)
(2) Watch I/O Writes (I/O writes to cacheable pages)
(3) Watch I/O Reads (I/O reads from private copy-back pages)

These Bus Watch modes are controlled by bits <3:1> of the CAMMU Control Register, as explained in *Section 7.6, Internal Registers.*

Watch CPU Writes

Watch CPU Writes is enabled in a CAMMU to ensure that data in the CAMMU cache is updated with new data written by another CAMMU into its shared main memory pages, or to invalidate cache lines (cache data) in the case of quadword writes.

The CPU transfers data to/from main memory via the D-CAMMU. The D-CAMMU transfers the data using either single-word (byte, halfword, or word) transfers, or quadword transfers. When, with Watch CPU Writes enabled, a slave CAMMU (a CAMMU that is not accessing the bus) detects a CPU write to one of its shared main memory pages by a master CAMMU (a D-CAMMU that is accessing the bus), the slave CAMMU determines whether the accessed location is in its cache, and whether the write involves one word for four words. If a single word write, the CAMMU updates the matched cache line. If a quadword write, the CAMMU does not update the matched cache line but instead invalidates the line by clearing the LV bit.

Watch I/O Writes

Watch I/O Writes, when invoked, functions identically to Watch CPU Writes. The two modes differ in that Watch I/O Writes responds to I/O writes rather than to CPU writes.

When a CAMMU with Watch I/O Writes enabled detects an I/O write to one of its cacheable pages, the CAMMU determines whether the accessed data is cached in the CAMMU, and whether the write involves one word or four words. If a single-word write, the CAMMU updates the matched cache line. If a quadword write, the CAMMU does not update the matched line but instead invalidates the line by clearing the LV bit.

Watch I/O Reads

Watch I/O Reads is enabled to ensure that data read by I/O devices from private, copy-back pages is always current data. This mode functions identically for both single-word and quadword I/O reads.

When this mode is enabled in a D-CAMMU, the D-CAMMU monitors the system bus for reads by I/O of private, copy-back pages. When the D-CAMMU detects such a read, it searches its cache for the data. If the data is not cached or the cached data LD bit is clear, the I/O device reads the data directly from main

memory. If the data is cached and the LD bit is set, the D-CAMMU aborts the assertion of data by main memory and asserts the current cache data on the CLIPPER Bus. The D-CAMMU thus intervenes in the I/O read operation to provide the more current cached data. If the Clear Valid bit in the Control Register is set, then the CAMMU will also clear the Line Valid bit in the cache line used to supply the data; the memory interface can use the cache data to update its own contents, as described in *Section 9.4.7.*

Note that because the copy-back caching mode applies only to private pages not shared by CAMMUs, this Bus Watch mode is invoked only during I/O reads of copy-back pages.

7.6. Internal Registers

Each CAMMU contains five software-accessible registers used for initialization and control. Two of these registers, the Supervisor PDO and User PDO, are used in address translation; they contain the base addresses of the supervisor and user Page Table Directories. The Fault register is loaded with the virtual address associated with certain fault conditions and is used by the operating system to implement virtual memory. The Control and Reset registers are used to control various aspects of CAMMU operation. These registers are discussed in the following sections.

7.6.1. Supervisor PDO Register

The Supervisor PDO (Page Directory Origin) Register is a 20-bit read/write register that holds the base address of a Page Table Directory address which is used by the DTU during supervisor mode address translation (see *Section 7.2.3, Dynamic Translation Unit*).

7.6.2. User PDO Register

The User PDO (Page Directory Origin) Register is a 20-bit read/write register that holds the base address of a Page Table Directory address which is used by the DTU during user mode address translation (see *Section 7.2.3, Dynamic Translation Unit*).

7.6.3. Fault Register

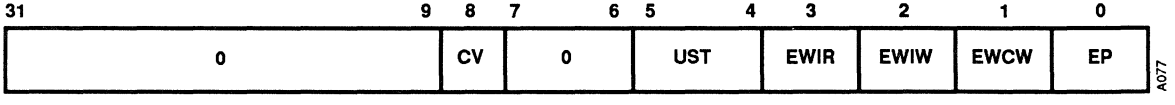
The Fault Register is a 32-bit read-only register which holds the virtual address of the data or instruction memory location that generated a page fault. It is intended for use by trap handling routines for fault recovery.

7.6.4. Control Register

The Control Register is a 9-bit read/write register used to enable prefetching in the I-CAMMU, to selectively enable the Bus Watch modes, to assign a system tag to unmapped memory addresses, and to enable the clearing of cache line LV bits during Bus Watch of I/O Reads. The Control Register is shown in *Figure 28* and is described below.

CV: Clear Valid
When this bit is set, the LV (Line Valid) bit in a copy-back cache line is cleared by hardware when the more current (dirty) data contained within that line is supplied by the CAMMU for an I/O quadword read (as a result of Bus Watch of I/O Reads). This permits pages that are swapped by I/O back to disk to be simultaneously purged from the cache. Use of this option requires the memory interface to use the data sent to the I/O device to update its own contents, except in cases where the data will not be read by another I/O device (see *Section*

Figure 28 CAMMU Control Register



NOTE: BITS <6:7> AND <31:9> MUST ALWAYS BE PROGRAMMED 0 OR UNDEFINED WILL OCCUR.

9.4.7). The Clear Valid option is disabled by clearing this bit. On reset, this bit is cleared by hardware.

Note: The Clear Valid bit was called the "Clear Dirty" bit in previous documents.

UST: Unmapped System Tag
When the Mapped Mode bit in the CPU System Status Word is clear, all CPU addresses, except supervisor virtual addresses 0 - 7FFF Hex which are mapped by the HTLB, are treated by the CAMMUs as real addresses requiring no translation. These unmapped addresses therefore have no TLB or Page Table source of system tags. The CAMMUs, therefore, use the two-bit UST field to tag pages referenced with unmapped addresses as follows:

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

The UST bits map to TG<1:0> CLIPPER Bus lines. TG<2> is forced to 0.

UST is set to 3 by CLIPPER Module reset.

EWIR: Enable Watch I/O Reads
EWIR, when set, enables Watch I/O Reads operation. This bit is ignored by the I-CAMMU. EWIR is set by CLIPPER Module reset.

EWIW: Enable Watch I/O Writes
EWIW, when set, enables Watch I/O Writes operation. EWIW is set by CLIPPER Module reset.

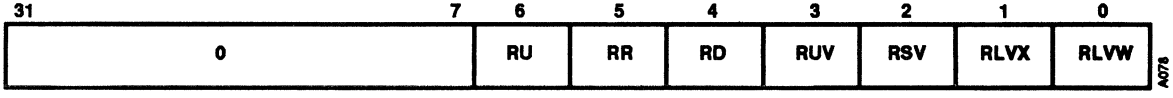
EWCW: Enable Watch CPU Writes
EWCW, when set, enables Watch CPU Writes operation. EWCW is set by CLIPPER Module reset.

EP: Enable Prefetch
EP, when set, enables I-CAMMU prefetching. When EP is clear, I-CAMMU prefetching is disabled, and the I-CAMMU fetches instructions "on demand." The state of this bit is ignored by the D-CAMMU, which always fetches on demand. EP is set by CLIPPER Module reset.

7.6.5. Reset Register

The Reset Register is a 7-bit, write-only register that allows selective resetting of the CAMMU cache and TLB (see *Figure 29*). The cache LV (Line Valid) and U (Used) flags, and the TLB SV (Supervisor Valid), UV (User Valid), D (Dirty) and R (Referenced) flags can be cleared by setting individual Reset Register bits.

Figure 29 CAMMU Reset Register



NOTE: BITS <31:7> MUST ALWAYS BE PROGRAMMED 0 OR UNDEFINED RESULTS WILL OCCUR.

Figure 30 CAMMU Access Map

	VIRTUAL ADDRESS	DESCRIPTION
RESERVED	0x00004E00 — 0x00004FFF	Reserved.
GLOBAL CAMMU	0x00004D81 — 0x00004DFF	Reserved
	0x00004D80	Global CAMMU, Reset register
	0x00004D41 — 0x00004D7F	Reserved.
	0x00004D40	Global CAMMU, Control register
	0x00004D11 — 0x00004D3F	Reserved
	0x00004D10	Global CAMMU, Fault register
	0x00004D09 — 0x00004D0F	Reserved.
	0x00004D08	Global CAMMU, User Page Directory Offset register
	0x00004D05 — 0x00004D07	Reserved
	0x00004D04	Global CAMMU, Supervisor Page Directory Offset register.
	0x00004D00 — 0x00004D03	Reserved
	0x00004CFF	Global CAMMU, TLB Line Set 63, X Line, VA Field.
	0x00004CFE	Global CAMMU, TLB Line Set 63, X Line, RA Field.
	0x00004CFD	Global CAMMU, TLB Line Set 63, W Line, VA Field.
	0x00004CFC	Global CAMMU, TLB Line Set 63, W Line, RA Field.
	0x00004C04 — 0x00004CFB	Global CAMMU, Line Sets 1 through 62.
	0x00004C03	Global CAMMU, TLB Line Set 0, X Line, VA Field
	0x00004C02	Global CAMMU, TLB Line Set 0, X Line, RA Field.
	0x00004C01	Global CAMMU, TLB Line Set 0, W Line, VA Field.
	0x00004C00	Global CAMMU, TLB Line Set 0, W Line, RA Field
I-CAMMU	0x00004B81 — 0x00004BFF	Reserved.
	0x00004B80	I-CAMMU, Reset register.
	0x00004B41 — 0x00004B7F	Reserved.
	0x00004B40	I-CAMMU, Control register.
	0x00004B11 — 0x00004B3F	Reserved.
	0x00004B10	I-CAMMU, Fault register.
	0x00004B09 — 0x00004B0F	Reserved
	0x00004B08	I-CAMMU, User Page Directory Offset register
	0x00004B05 — 0x00004B07	Reserved
	0x00004B04	I-CAMMU, Supervisor Page Directory Offset register
	0x00004B00 — 0x00004B03	Reserved.
	0x00004AFF	I-CAMMU, TLB Line Set 63, X Line, VA Field
	0x00004AFE	I-CAMMU, TLB Line Set 63, X Line, RA Field
	0x00004AFD	I-CAMMU, TLB Line Set 63, W Line, VA Field.
	0x00004AFC	I-CAMMU, TLB Line Set 63, W Line, RA Field.
	0x00004A04 — 0x00004AFB	I-CAMMU, Line Sets 1 through 62.
	0x00004A03	I-CAMMU, TLB Line Set 0, X Line, VA Field
	0x00004A02	I-CAMMU, TLB Line Set 0, X Line, RA Field.
	0x00004A01	I-CAMMU, TLB Line Set 0, W Line, VA Field
	0x00004A00	I-CAMMU, TLB Line Set 0, W Line, RA Field
D-CAMMU	0x00004981 — 0x000049FF	Reserved.
	0x00004980	D-CAMMU, Reset register
	0x00004941 — 0x0000497F	Reserved
	0x00004940	D-CAMMU, Control register.
	0x00004911 — 0x0000493F	Reserved
	0x00004910	D-CAMMU, Fault register.
	0x00004909 — 0x0000490F	Reserved.
	0x00004908	D-CAMMU, User Page Directory Offset register
	0x00004905 — 0x00004907	Reserved
	0x00004904	D-CAMMU, Supervisor Page Directory Offset register
	0x00004900 — 0x00004903	Reserved.
	0x000048FF	D-CAMMU, TLB Line Set 63, X Line, VA Field.
	0x000048FE	D-CAMMU, TLB Line Set 63, X Line, RA Field
	0x000048FD	D-CAMMU, TLB Line Set 63, W Line, VA Field.
	0x000048FC	D-CAMMU, TLB Line Set 63, W Line, RA Field
	0x00004804 — 0x000048FB	D-CAMMU, Line Sets 1 through 62.
	0x00004803	D-CAMMU, TLB Line Set 0, X Line, VA Field
	0x00004802	D-CAMMU, TLB Line Set 0, X Line, RA Field
	0x00004801	D-CAMMU, TLB Line Set 0, W Line, VA Field
	0x00004800	D-CAMMU, TLB Line Set 0, W Line, RA Field

A079

The Reset Register bits and their associated reset operations are as follows:

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

The reset operations shown are performed by writing to the Reset Register with the appropriate data pattern.

These CAMMU registers, as well as the CAMMU TLBs, are located in virtual page 4, which is translated by the Hardwired TLB into Page 0 of I/O space. A map of CAMMU I/O space is shown in *Figure 30.*

The CPU accesses the D-CAMMU I/O space directly. The CPU accesses the I-CAMMU I/O space indirectly via the D-CAMMU, because the I-CAMMU/CPU Instruction Bus is tied to CPU instruction buffers which only transfer instructions.

7.6.6. CAMMU Register Access

The D-CAMMU registers are located in virtual address 4800-49FF (Hex). The I-CAMMU registers are located in virtual address 4A00-4BFF. These addresses are used to access registers in individual CAMMUs.

The CAMMUs can also be addressed as a group using global addresses for TLB writes, register writes, and TLB/cache reset. In systems utilizing multiple CLIPPER Modules, for example, a CPU can execute global writes to CAMMUs other than its companion D-CAMMU by accessing virtual address locations 4Cnn (Hex, TLB write), and 4Dnn (Hex, register write, and TLB/cache reset). I/O devices can execute the global writes by accessing Cnn and Dnn (Hex).

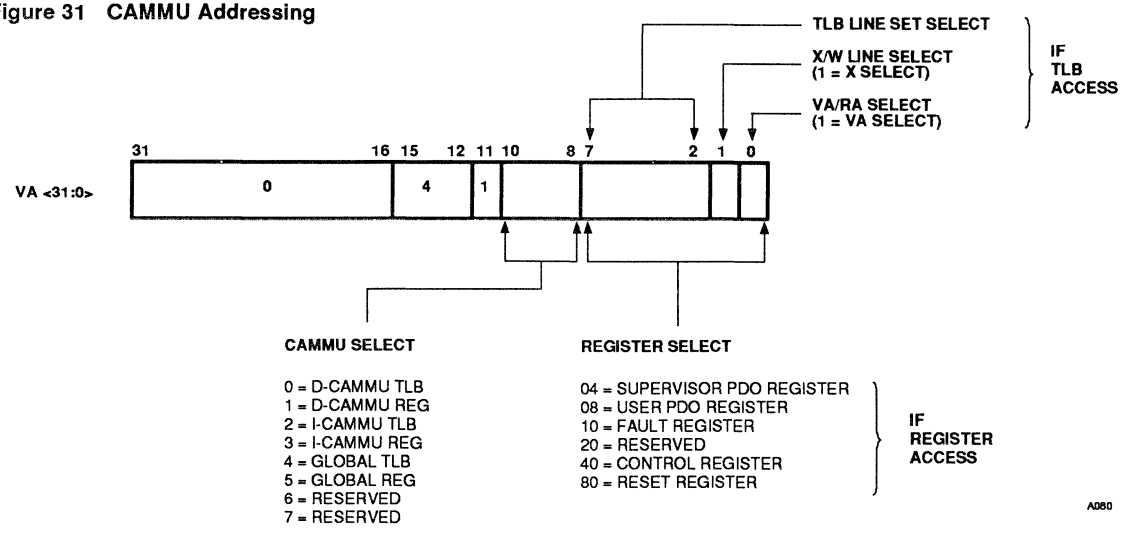
Global writes are typically used in multi-CPU systems when main memory pages that are shared by more than one CLIPPER Module are replaced. If the virtual address of a page being replaced is identical for all modules sharing the page, a single global write to CAMMU I/O space can be used to invalidate the TLB entry corresponding to the outgoing page in all CAMMUs.

Register Addressing

CAMMU I/O space addresses are shown in *Figure 31.* Virtual address bits <31:11> comprise the CAMMU Base Address field, which must point to the upper half of virtual page 4 for CAMMU access.

Virtual address bits <10:8> comprise the CAMMU Select field. This field identifies the following:

Figure 31 CAMMU Addressing



Bit No. 10	9	8	CAMMU Selected	Operation/Access
0	0	0	D-CAMMU	R/W TLB
0	0	1	D-CAMMU	R/W Registers; Reset TLB/Cache
0	1	0	I-CAMMU	R/W TLB
0	1	1	I-CAMMU	R/W Registers; Reset TLB/Cache
1	0	0	Global	Write TLB
1	0	1	Global	Write Registers; Reset TLB/Cache

Note:
The TLBs and caches are reset by writing to the Reset Register.

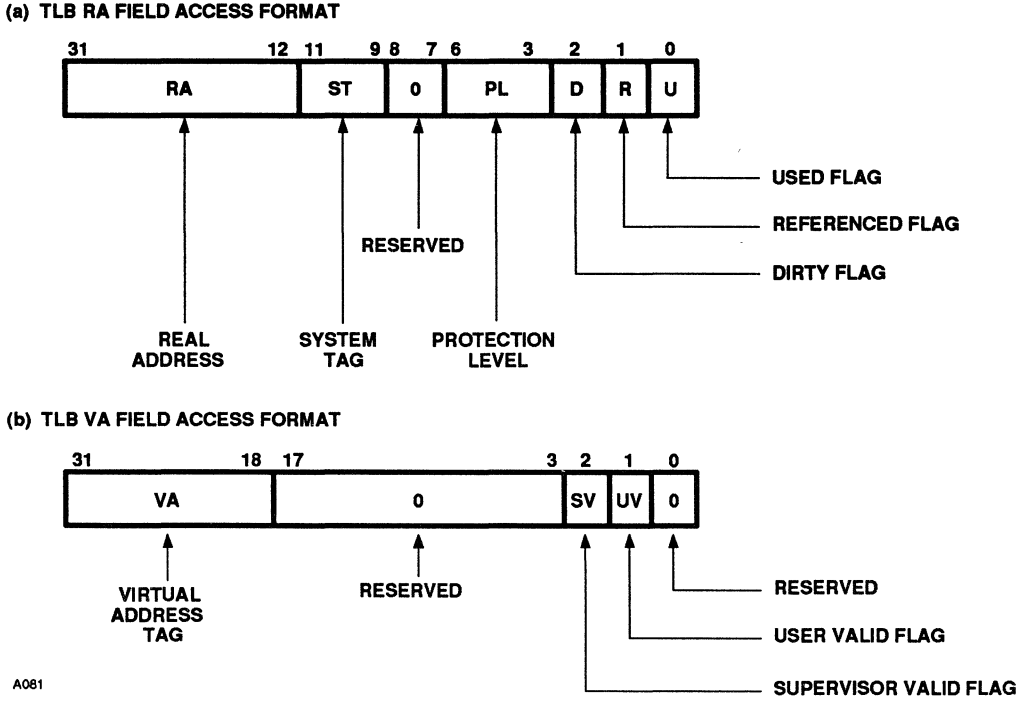
The first four entries show individual I- and D-CAMMU addressing. The last two entries show global addressing, intended for use in systems utilizing more than one CLIPPER Module. A CPU uses global addressing in such a system to access a specific register or to reset the TLB and cache in all CAMMUs in the system other than its own D-CAMMU.

Bits <7:0> of the virtual address comprise the Register Select field. This field identifies the register or the TLB field being accessed. All five CAMMU registers, and individual VA (Virtual Address) and RA (Real Address) fields of the TLB can be addressed.

If the operation is a TLB access, virtual address bits <7:2> address one of the 64 TLB entries, bit <1> addresses the W or X line of the TLB entry, and bit <0> addresses the VA or RA field of the addressed TLB line.

If the operation is a register access, virtual address bits <7:0> address the registers as follows:

Figure 32 TLB Access Data Formats



Bits <7:0>	Register Addressed
0000 0000	Reserved, Must Be Zero
0000 0001	Reserved, Must Be Zero
0000 0010	Reserved, Must Be Zero
0000 0100	Supervisor PDO (read/write)
0000 1000	User PDO (read/write)
0001 0000	Fault (read only)
0010 0000	Reserved, Must Be Zero
0100 0000	Control (read/write)
1000 0000	Reset (write only)

CAMMU Data Format
The format of data written to and read from the CAMMUs varies according to the register or TLB field addressed. Both the fields and the number of data bits used in the 32-bit data words written to the CAMMUs differ to accommodate individual CAMMU registers and register types.

TLB Access Data Format
TLB access data formats are shown in *Figure 32*. Two formats are used. One format is used when accessing a TLB RA field; the second is used when accessing a TLB VA field.

Figure 32A shows the data format used when accessing a TLB RA field. When accessing an RA field, the System Tag and Protection Level fields and the R and D flags of the addressed TLB line are also accessed, as well as the U flag of the TLB set containing the TLB line.

Figure 32B shows the data format used when accessing a TLB VA field. The UV and SV flags of the TLB line are also accessed. Note that several data bits are not used. These bits are reserved by Intergraph and must be zero.

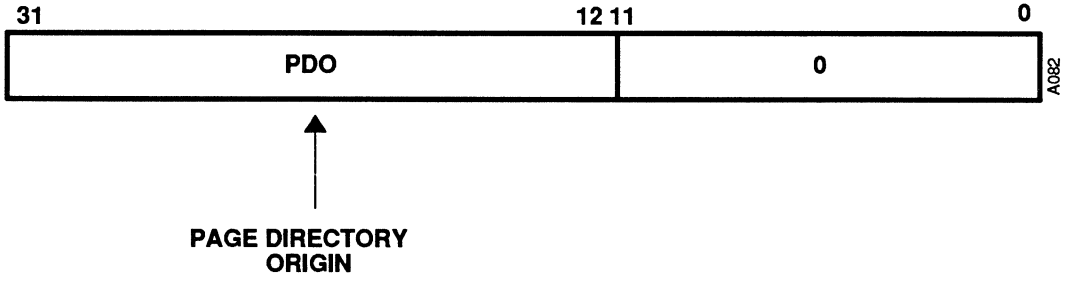
PDO Register Access Data Format
Figure 33 shows the data format used when accessing either the supervisor or the user PDO register. Bits <31:12> are used to transfer the 20-bit PDO data; bits <11:0> are reserved by Fairchild and must be zero.

Fault Register Data Format
The 32-bit address in the Fault Register is read as a 32-bit data word.

Control Register Access Data Format
The least-significant nine bits are used when accessing the Control Register; bits <31:9> are reserved by Intergraph and must be zero.

Reset Register Access Data Format
The seven least-significant bits are used when accessing the Reset Register; bits <31:7> are reserved by Intergraph and must be zero.

Figure 33 PDO Register Access Format



NOTE:
This format is used for both the user and supervisor PDO register access.

8. CLIPPER Hardware Reset

The CLIPPER Module is reset when power is initially applied to the module (power-on reset), and when $\overline{\text{RESET}}$ is asserted LOW during operation.

The response of the CPU to a hardware reset is as follows:

(1) The T flag in the PSW is cleared; the remaining flags in the PSW are undefined.
(2) The following SSW flags are cleared: EI, TP, M, U, K, KU, UU, and P; the remaining SSW flags are undefined.

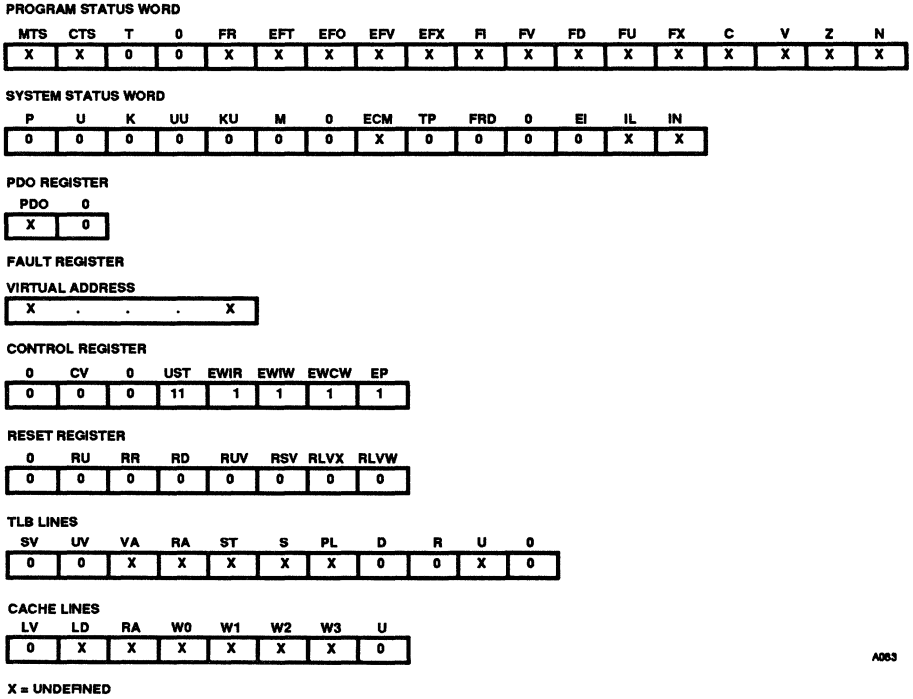
The response of each CAMMU to reset is as follows:

(1) All LV (Line Valid) flags in the cache are cleared.
(2) All U (Used) flags in the cache are cleared.
(3) All UV (User Valid) flags in the TLB are cleared.
(4) All SV (Supervisor Valid) flags in the TLB are cleared.
(5) All D (Dirty) flags in the TLB are cleared.
(6) All R (Referenced) flags in the TLB are cleared.
(7) Bits <8:0> of the Control Register are set to 3F.
(8) The Reset Register is cleared.

Reset therefore places the CLIPPER Module in unmapped supervisor mode with all traps and conditional interrupts disabled and with Bus Watch and prefetching enabled. *Figure 34* shows the state of the CLIPPER Module's CPU control registers, and the CAMMU's registers, TLB, and cache lines following reset. While $\overline{\text{RESET}}$ is asserted, all CLIPPER Module Bus active LOW signals are pulled HIGH (via pull-up resistors), and all active HIGH signals are forced LOW. BCLK continues clocking normally.

$\overline{\text{RESET}}$ must be held low for a minimum of 100 BCLK cycles after V_{DD} reaches V_{DD} min when power is initially applied to the CLIPPER Module (see *Figure 35*). This ensures adequate module reset time. It must be released in synchronization with BCLK. $\overline{\text{RESET}}$ must be

Figure 34 CLIPPER Module Following Reset



held LOW for a minimum of 100 BCLK cycles when asserted during operation, and both the assertion and release of $\overline{\text{RESET}}$ must be synchronized with BCLK.

The CLIPPER Module executes diagnostic routines following release of $\overline{\text{RESET}}$ if $\overline{\text{URDIAG}}$ is asserted during the two BCLK cycles following the release of $\overline{\text{RESET}}$ (see *Section 9.4.9, Diagnostics Control*). Then it begins execution at supervisor virtual address 6000H, which is mapped by the HTLB to real address 0 of Boot space.

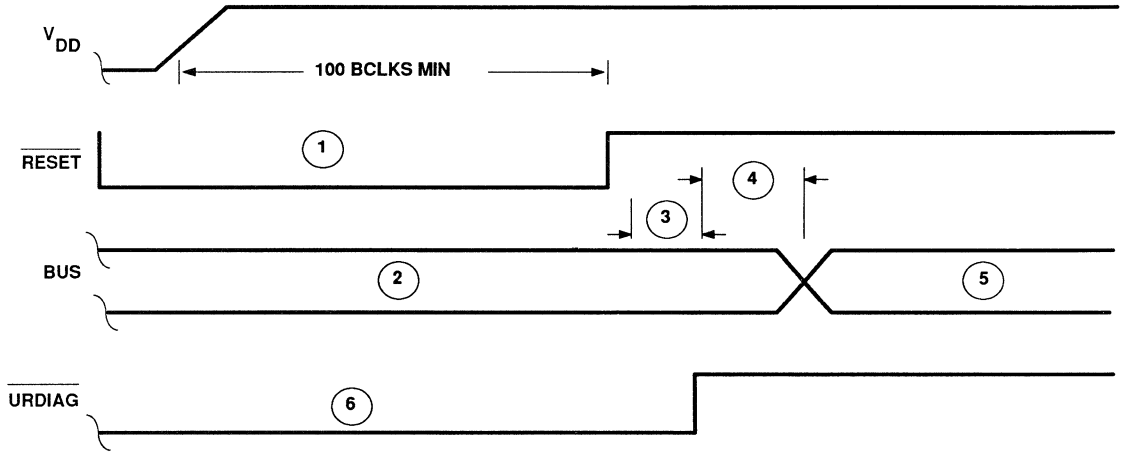
9. CLIPPER Bus

The CLIPPER Module interfaces to external system devices and functional units such as main memory, I/O devices and peripherals, and other CLIPPER Modules via the CLIPPER Bus—a high-speed, synchronous bus designed to support multiple bus masters.

The bus includes 32 bidirectional, multiplexed address/data lines which support byte, halfword, word, and quadword transfers. A separate interrupt bus allows fast interrupt management by the CLIPPER Module with no address/data line loading or contention, thereby increasing the effective bus bandwidth. The bus protocol allows devices that are clocked at different rates to interface to the CLIPPER Module through the use of wait states as required, and bus arbitration to be centralized in a simple, fast bus arbiter. The bus supports Bus Watch, which monitors the bus and takes corrective action to ensure data consistency between the CAMMUs and main memory.

The bus utilizes a single clock (BCLK), generated by the Clock Control Unit, for system clocking. All CLIPPER Module signal sampling and signal assertion are

Figure 35 Reset Timing



Notes:

1. $\overline{\text{RESET}}$ transitions must be synchronized with BCLK rising edges.
2. CLIPPER Bus is inactive until first instruction fetch.
3. Internal CPU startup time.
4. CPU diagnostics execution if $\overline{\text{URDIAG}}$ was asserted during the 2 BCLK cycles following release of $\overline{\text{RESET}}$.
5. Fetch from boot space.
6. $\overline{\text{URDIAG}}$ is asserted during $\overline{\text{RESET}}$ if CPU diagnostics execution prior to instruction execution is desired. $\overline{\text{URDIAG}}$ must remain asserted for at least 2 BCLK periods following release of $\overline{\text{RESET}}$ to assure recognition and can then either remain asserted or be released with no further effect on CLIPPER Module operation.

gated on the rising edge of this clock. All module outputs except BCLK are open drain and are tied to pullup resistors inside the module. These signals are tied to a 96-pin connector for interfacing to user-designed systems, where they may be buffered.

The signals tied to the CLIPPER Module connector constitute the CLIPPER Bus, shown in *Figure 36*. These signals are interfaced through buffers and logic devices as shown in *Figure 37*. Note that this interface includes ORed logic and address/data signal transceiver control ($\overline{DIR}$).

The CLIPPER Module Bus consists of the following groups of bus lines and signals:

— Address/Data multiplexed lines used for address and data transfer
— Cycle Type signals used to identify the number of bytes or words transferred during a bus operation, to identify the operation as a read, write, or global write, and to identify the bus master executing the operation as a CPU or an I/O device

Figure 36 CLIPPER Bus Signals

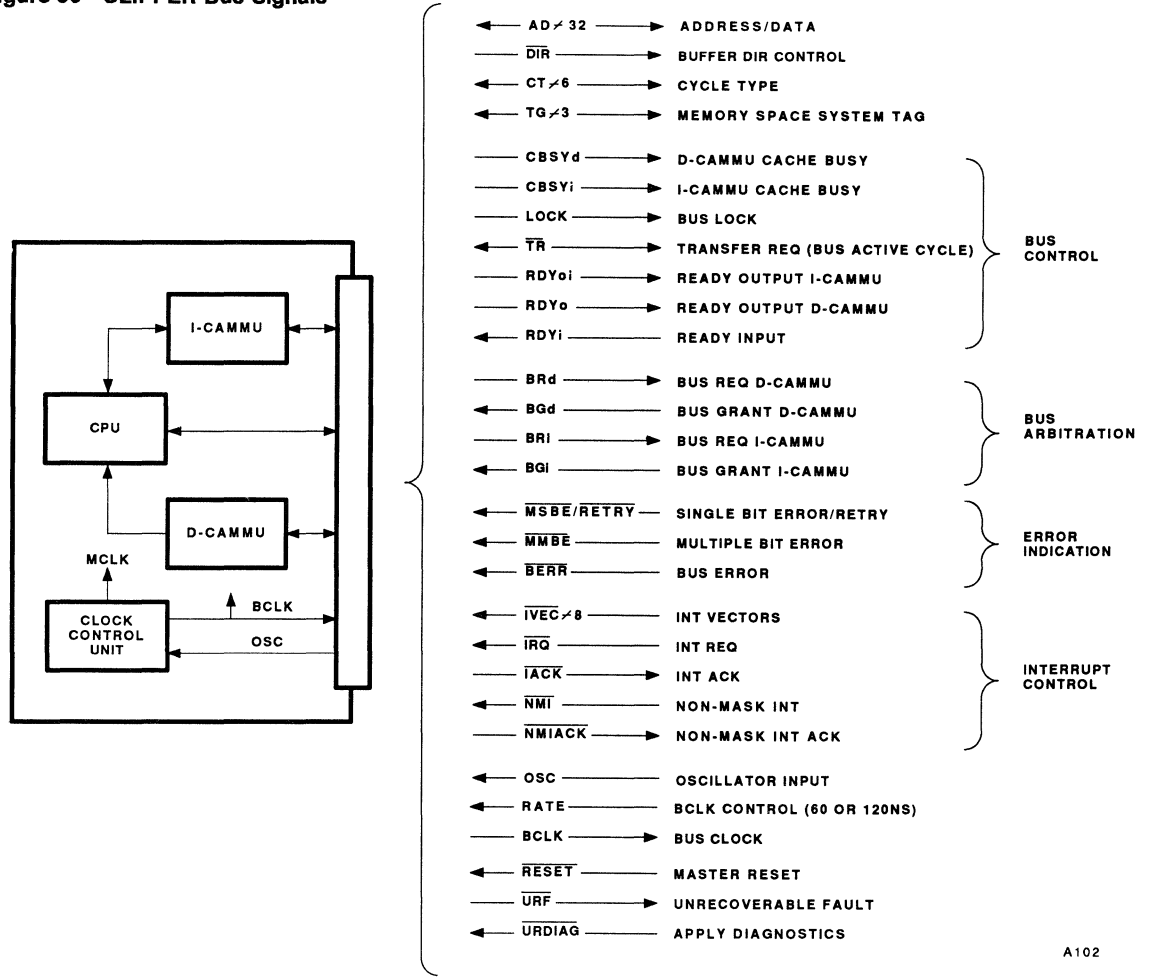
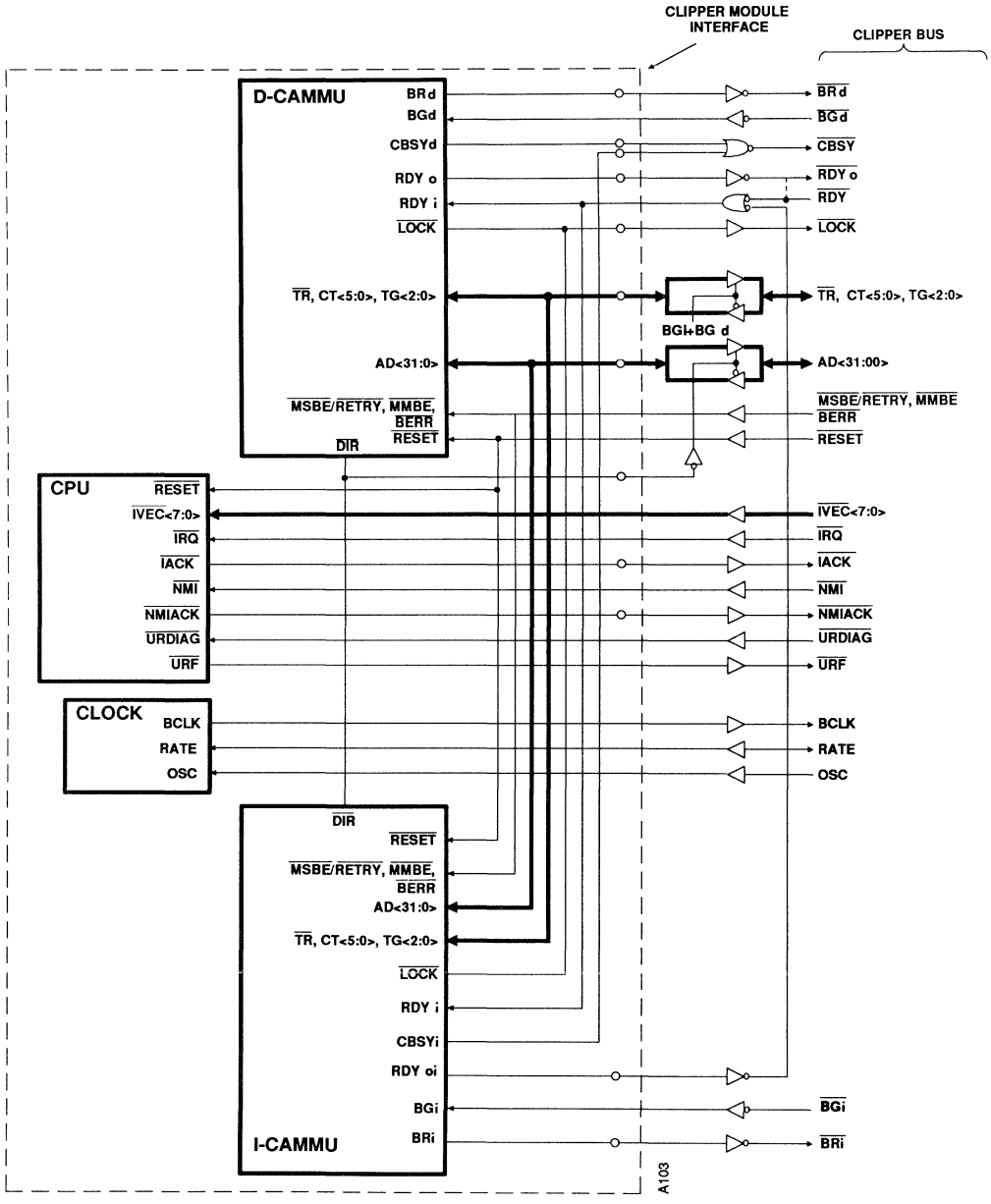


Figure 37 Module to CLIPPER Bus Interface



Notes:
1 RDY and RDYo can be connected together on the CLIPPER Bus
2 —o— =pullup resister

— Memory Space System Tags used to identify address space being accessed and the caching policy which applies to the accessed data
— Error signals used to report memory errors and bus errors
— Bus arbitration handshaking signals
— Interrupt control lines
— Interrupt vector bus
— Bus protocol control lines

An example CLIPPER Module system implementation showing these signals is depicted in *Figure 38. Table 12* contains detailed descriptions of the CLIPPER Bus

Figure 38 Example CLIPPER System (Block Diagram)

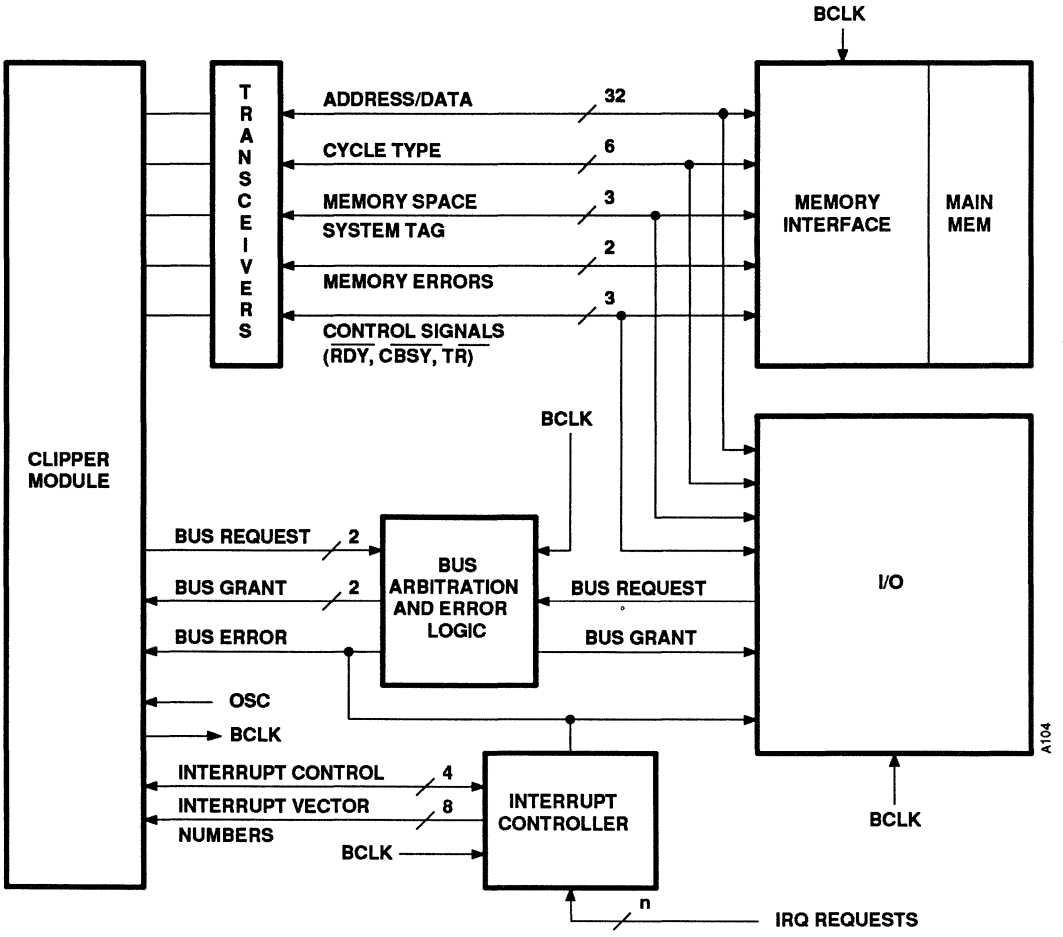


Table 12 CLIPPER Bus Signal Descriptions

Signal	Type	Description
AD <31:0>	I/O	ADDRESS/DATA. This is a positive logic (HIGH = logic 1) multiplexed address and data bus which is tied to the CAMMUs.
$\overline{DIR}$	O	DIRECTION CONTROL. This control signal can be used to control the drive direction of TTL tranceivers interfacing AD <31:0> to the CLIPPER Bus. A master CAMMU asserts $\overline{DIR}$ during an entire write operation and during the first two cycles of a read operation. A slave D-CAMMU asserts this signal when transferring data during an I/O read; a slave I-CAMMU asserts this signal when transferring data to a companion D-CAMMU. Drive direction is from CAMMU to the CLIPPER Bus when $\overline{DIR}$ is low.
TG <2:0>	I/O	MEMORY SPACE SYSTEM TAGS. These three CAMMU signals identify the address space being accessed, the page type, and the caching policy which applies to the accessed page. In addition, they signal two special operations, Cache Purge and Slave I/O mode. System tags are derived from one of four sources. In mapped mode, they are selected during address translation from the TLB, the HTLB, or from page tables in main memory. In unmapped mode, TG<2> is zero and TG<1:0> is selected by the UST bits in the CAMMU Control Register.TG<2:0> tag encoding is as follows:

TG2	TG1	TG0		Encoding
0	0	0	=	main memory/private cacheable/ write-through
0	0	1	=	main memory/shared cacheable/write-through
0	1	0	=	main memory/private cacheable/copy-back
0	1	1	=	main memory/noncacheable
1	0	0	=	I/O space/noncacheable
1	0	1	=	boot space/noncacheable
1	1	0	=	cache purge
1	1	1	=	slave I/O mode/main memory/noncacheable

Note: The slave CAMMU can continually monitor the Memory Space System Tag and check for cache/main memory data consistency when the Bus Watch modes are enabled. The Bus Watch modes, when enabled, are invoked during the following CLIPPER Bus operations:

(1) I/O writes to shared or private pages.
(2) CPU writes to shared pages.
(3) I/O reads from private copy-back pages.

Frequently systems require the transfer of data between main memory and an I/O device. This type of data transfer is normally implemented by a CPU as a read operation into a CPU register, followed by a CPU write operation to the I/O device. This type of operation is accelerated by the CLIPPER Module through the use of slave I/O mode identified by the Memory Space System Tag. The slave I/O mode allows an I/O device to capture data being read by the CPU during the read portion of the operation. The I/O device must be able to interpret TG = 7 as slave I/O mode, then read the transferred data as it is being read by the CPU. The data read by the CPU is discarded.

Table 12 CLIPPER Bus Signal Descriptions (cont.)

Signal	Type	Description
CT <5:0>	I/O	CYCLE TYPE. These six CAMMU signals indicate the type of CLIPPER Bus operation in progress. CT<5:0> signal encoding is as follows: Signal / State / Operation CT5: 0 CPU master; 1 I/O master. CT4: 0 Write operation; 1 Read operation. CT<3:2>: 00 Word/halfword/byte transfer; 01 Quadword transfer; 10 Reserved; 11 Global CAMMU write. CT<1:0>: 00 Whole word transfer; AD<1:0> must be 0. 01 Reserved. 10 Byte transfer; AD<1:0> define the byte position. 11 Halfword transfer; AD<1> defines the halfword position; AD<0> must be 0. Notes: (a) CT<1:0> have meaning only for single word transfers, with AD<1:0> pointing to transferred word/bytes. (b) In halfword/byte mode, the data must be written in the location specified by AD<1:0>. (c) During a quadword transfer, the master must assert AD<3:0> all 0 to point to a quadword boundary.
CBSYi, CBSYd	O	CACHE BUSY (I-CAMMU, D-CAMMU). A CAMMU asserts CBSY to indicate execution of internal operations associated with Bus Watch operations. Main memory data must not be asserted on the CLIPPER Bus while CBSYi or CBSYd is asserted. If a D-CAMMU asserts RDYo while asserting CBSY, indicating that it is asserting more recent cache data on the CLIPPER Bus, main memory must abort the data transfer operation.
$\overline{\text{LOCK}}$	O	BUS LOCK. $\overline{\text{LOCK}}$ is asserted by a bus master when it requires the CLIPPER Bus for more than one operation. $\overline{\text{LOCK}}$ is asserted by the CAMMUs during the following operations: (1) DTU Page Table Directory and Page Table accesses. (2) DTU R or D bit modifications in Page Tables. (3) Read-modify-write (test and set) operations. (4) Cache line replace and fetch on cache miss (quadword write followed by quadword read).
$\overline{\text{TR}}$	I/O	TRANSFER REQUEST. $\overline{\text{TR}}$ is asserted by a bus master to indicate that a CLIPPER Bus operation is in progress.

Table 12 CLIPPER Bus Signal Descriptions (cont.)

Signal	Type	Description
RDYi	I	READY INPUT. RDYi is tied to both CAMMUs. During read operations, the slave with the addressed data asserts RDYi to indicate that it has asserted the data on the AD bus. During single word, byte or halfword write operations, the slave asserts RDYi to indicate that it has latched (read) the data. During quadword write operations, the slave asserts RDYi to indicate that it has latched the data word currently on the AD bus, and is ready to latch the next word.
RDYo	O	READY OUTPUT. RDYo is asserted by the D-CAMMU during Watch I/O Reads operations to indicate to the I/O device that it has asserted data on the AD bus for reading. This occurs when the data location being accessed in main memory is cached, and the cache data is more "recent" than the main memory data. RDYo can be tied to $\overline{\text{RDY}}$ on the CLIPPER Bus.
RDYoi	O	READY OUTPUT. RDYoi is asserted by the I-CAMMU when it is being accessed by its companion D-CAMMU. RDYoi is not interfaced to the CLIPPER Bus.
BRi,BRd	O	BUS REQUEST (I-CAMMU, D-CAMMU). These signals are asserted by the respective CAMMUs to obtain control of the CLIPPER Bus.
BGi,BGd	I	BUS GRANT (I-CAMMU, D-CAMMU). Bus Grant is asserted by the CLIPPER Bus arbitration logic in response to a Bus Request by a CAMMU, and indicates that the requesting CAMMU has control of the bus.
$\overline{\text{MSBE}}$/ $\overline{\text{RETRY}}$	I	MEMORY SINGLE BIT ERROR/RETRY. The main memory interface asserts $\overline{\text{MSBE}}$/$\overline{\text{RETRY}}$ when it detects a corrected error in main memory during a read operation. (Typically, in systems utilizing error correction, only single-bit errors are corrected.) $\overline{\text{MSBE}}$/$\overline{\text{RETRY}}$ is tied to both CAMMUs, and is sampled by the CAMMUs when RDYi is sampled. A master CAMMU issues a trap to the CPU when it detects $\overline{\text{MSBE}}$/$\overline{\text{RETRY}}$ asserted. The main memory interface must not assert an interrupt when it detects a corrected data error. The $\overline{\text{MSBE}}$/$\overline{\text{RETRY}}$ signal must be presented to the CAMMU by the memory interface along with (during the same BCLK as) RDYi and the data to indicate a corrected error. $\overline{\text{MSBE}}$/$\overline{\text{RETRY}}$ may not be asserted when RDYi is inactive during main memory accesses. The $\overline{\text{MSBE}}$/$\overline{\text{RETRY}}$ signal is also used to abort and retry CLIPPER Bus operations. If the signal is asserted during access of I/O space (TG=4) while RDYi is inactive, the current bus operation is aborted by the master CAMMU with no trap assertion to the CPU. Thus, if this pin is active during the same BCLK that RDYi is HIGH, an $\overline{\text{MSBE}}$ condition is recognized; if this pin is active during a BCLK when RDYi is LOW (for an I/O space access), a $\overline{\text{RETRY}}$ condition is recognized.

Table 12 CLIPPER Bus Signal Descriptions (cont.)

Signal	Type	Description
$\overline{\text{MMBE}}$	I	MEMORY MULTIPLE BIT ERROR. The main memory interface asserts $\overline{\text{MMBE}}$ when it detects an uncorrectable error in main memory during a read operation. (Typically, these will be multiple-bit errors, because in systems using error correction, only single-bit errors are corrected.) This signal must be asserted during the same BCLK cycle that nRDY is asserted. $\overline{\text{MMBE}}$ is tied to both CAMMUs, and is sampled by the CAMMUs when RDYi is sampled. A master CAMMU issues a trap to the CPU when it detects nMMBE asserted. The main memory interface must not assert an interrupt when it detects an uncorrectable data error.
$\overline{\text{BERR}}$	I	BUS ERROR. $\overline{\text{BERR}}$ should be asserted by user-designed logic to indicate a CLIPPER Bus error condition such as a bus timeout. $\overline{\text{BERR}}$ is tied to both CAMMUs. Upon assertion of $\overline{\text{BERR}}$, the master CAMMU terminates the bus operation and indicates to the CPU that the operation is completed. (If the bus error occurs during a read operation, the data asserted on the AD bus at the time $\overline{\text{BERR}}$ is asserted is transferred by the CAMMU to the CPU). The CAMMU does not issue a trap in response to a bus error; the bus error logic should assert an interrupt.
$\overline{\text{IVEC}}$ <7:0>	I	INTERRUPT VECTORS. This is an inverted logic (LOW=logic 1) bus, tied directly to the CPU. It transfers interrupt vector numbers associated with interrupt requests.
$\overline{\text{IRQ}}$	I	INTERRUPT REQUEST. This signal, tied directly to the CPU, is asserted by system devices for interrupt service requests. Once asserted, $\overline{\text{IRQ}}$ must remain asserted until $\overline{\text{IACK}}$ is asserted by the CPU. An interrupt level and number must be asserted on $\overline{\text{IVEC}}$<7:0> with each interrupt request. $\overline{\text{IRQ}}$ is maskable.
$\overline{\text{IACK}}$	O	INTERRUPT ACKNOWLEDGE. $\overline{\text{IACK}}$ is asserted by the CPU in response to an interrupt request ($\overline{\text{IRQ}}$) to acknowledge that servicing of the interrupt is in progress.
$\overline{\text{NMI}}$	I	NON-MASKABLE INTERRUPT. This signal, tied directly to the CPU, is asserted by system devices for non-maskable interrupt service requests. Once asserted, $\overline{\text{NMI}}$ must remain asserted until $\overline{\text{NMIACK}}$ is asserted by the CPU.
$\overline{\text{NMIACK}}$	O	NON-MASKABLE INTERRUPT ACKNOWLEDGE. This signal is asserted by the CPU in response to an $\overline{\text{NMI}}$ request to acknowledge that servicing of the interrupt is in progress.
RATE	I	BCLK RATE CONTROL. This input to the CLIPPER Module controls the CLIPPER Bus BCLK frequency. When RATE is tied to GND, BCLK frequency is 1/2 MCLK frequency. When RATE is tied to V_{DD}, BCLK frequency is 1/4 MCLK frequency. If the standard 66.7 MHz crystal is used in the CLIPPER Module, BCLK frequency is 16.7 MHz if RATE is tied to GND, and 8.3 MHz if RATE is tied to V_{DD}.

Table 12 CLIPPER Bus Signal Descriptions (cont.)

Signal	Type	Description
BCLK	O	BUS CLOCK. BCLK clocks all devices on the CLIPPER Bus. All signals must be clocked onto the CLIPPER Bus on the rising edge of BCLK; all signals must be latched/sampled from the CLIPPER Bus on the rising edge of BCLK. The propagation delay of signals asserted on the system bus must be less than one BCLK period and more than the BCLK skew between devices in order to ensure proper operation of the synchronous CLIPPER Bus.
$\overline{\text{RESET}}$	I	RESET. This is the master reset signal which is asserted by system logic to reset the CLIPPER Module and other devices on the CLIPPER Bus. Upon release of $\overline{\text{RESET}}$, the CPU begins instruction fetching at Boot space address 0.
$\overline{\text{URF}}$	O	UNRECOVERABLE FAULT. This signal is asserted by the CPU to indicate that it has stopped program execution as a result of an unrecoverable fault condition. An unrecoverable fault occurs when the CPU encounters an error condition during execution of on-chip diagnostic routines, or when a trap occurs during the execution of INTRAP or **retl**.
$\overline{\text{URDIAG}}$	I	APPLY DIAGNOSTICS. This input to the CPU is asserted to force the CPU to execute on-chip diagnostic routines resulting in the following: (1) The diagnostics detected no error conditions. The CPU begins program execution at Boot space address 0 (supervisor virtual address 6000 hex). (2) The diagnostics detected an error condition. The CPU asserts $\overline{\text{URF}}$ and stops execution. $\overline{\text{RESET}}$ must be asserted when $\overline{\text{URDIAG}}$ is asserted.
OSC	I	OSCILLATOR INPUT. This signal is used by the CLIPPER Clock Control Unit to derive MCLK and BCLK. MCLK is the CLIPPER internal clock; BCLK is the CLIPPER Bus clock.

Table 13 Signal Summary

Signal	Mnemonic	Input/Output[1]	Active State
Address/Data Bus	AD	I/O	HIGH
Direction Control	$\overline{DIR}$	O	HIGH = input LOW = output
Memory Space System Tag	TG	I/O	HIGH
Cycle Type	CT	I/O	HIGH
Cache Busy	CBSYi, CBSYd	O	HIGH
Bus Lock	$\overline{LOCK}$	O	LOW
Transfer Request	$\overline{TR}$	I/O	LOW
Ready	RDYi	I	HIGH
	RDYo, RDYoi	O	HIGH
Bus Request	BRi, BRd	O	HIGH
Bus Grant	BGi, BGd	I	HIGH
Memory Single Bit Error/Retry	$\overline{MSBE/RETRY}$	I	LOW
Memory Multiple Bit Error	$\overline{MMBE}$	I	LOW
Bus Error	$\overline{BERR}$	I	LOW
Interrupt Vector Bus	$\overline{IVEC}$	I	LOW
Interrupt Request	$\overline{IRQ}$	I	LOW
Interrupt Acknowledge	$\overline{IACK}$	O	LOW
Non-Maskable Interrupt	$\overline{NMI}$	I	LOW
Non-Maskable Interrupt Acknowledge	$\overline{NMIACK}$	O	LOW
BCLK Rate Select[2]	RATE	I	HIGH = 120 ns LOW = 60 ns
Bus Clock	BCLK	O	—
Master Reset	$\overline{RESET}$	I	LOW
Unrecoverable Fault	$\overline{URF}$	O	LOW
Apply Diagnostics	$\overline{URDIAG}$	I	LOW
Oscillator Input	OSC	I	—

Notes:

1. Inputs are designed with a nominal switching threshold of 1.3 V and are therefore referred to as TTL compatible. All outputs (excluding BCLK, and $\overline{URF}$) are open drain structures with pull-up resistors (220 Ohms) to V_{CC} on the module. BCLK and $\overline{URF}$ are standard CMOS output signals. If an external pull-up is used for $\overline{URF}$, it must be at least 220 Ohms. Timing parameters are referenced to standard TTL levels.

2. The BCLK period values shown are for an OSC frequency of 66.7 MHz.

signals, and *Table 13* contains a summary of the bus signals.

9.1. System Clock

The CLIPPER Module is clocked by an external oscillator signal, OSC. A Clock Control Unit derives two clocks from OSC: MCLK and BCLK.

MCLK (Module Clock) is the internal CLIPPER master clock, used to drive the CPU, the CAMMUs, and associated module logic. The frequency of MCLK is one half the frequency of OSC. Therefore, if the typical MHz OSC frequency is used, the MCLK frequency is 33.3 MHz.

BCLK (Bus Clock) is the CLIPPER Module system clock, used to clock devices interfaced to the CLIPPER Bus. The CLIPPER Bus is synchronous: all data and control signals are asserted and sampled on the rising edge of BCLK. BCLK frequency is either one half or one fourth the frequency of MCLK, depending on the state of the CLIPPER Module Rate control pin. If RATE is tied to GND, BCLK frequency is one half the frequency of MCLK; if RATE is tied to VCC, BCLK frequency is one fourth the MCLK frequency. Therefore, assuming an OSC frequency of 66.7 MHz, BCLK frequency is either 16.7 MHz (60ns) or 8.3 MHz (120ns). BCLK is in phase with MCLK, with the LOW to HIGH transitions of the clocks occurring in phase with a skew of less than ± 5ns.

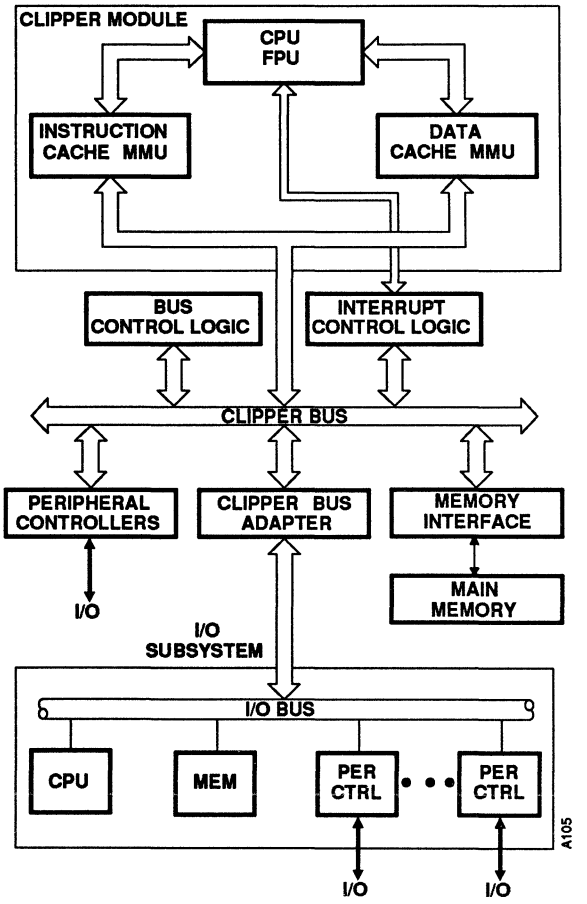
9.2. System Configuration

Any device (or functional unit) which meets the CLIPPER Bus protocol and electrical requirements (timing, threshold, and loading) can be interfaced to the CLIPPER Bus. Such devices include memory, I/O devices, and subsystems as well as the CLIPPER Module. A typical CLIPPER system configuration is shown in *Figure 39*.

Up to 4 G-bytes of memory can be addressed by the CLIPPER Module via its 32-bit address bus. This memory can be interfaced directly to the CLIPPER Bus if required.

I/O devices such as disk controllers, bus translators, data communications devices, and associated control logic such as bus arbitration units and interrupt control-

Figure 39 CLIPPER System



lers, can also be interfaced to the bus. Such devices may be "off-the-shelf" or user-designed. I/O devices are typically located in I/O space, but can also be located in main memory space.

9.3. Definitions

Several terms are used in the following text which may not have universally accepted meanings. These terms and their defintions as used in this text are as follows:

Master:
A device which has control of the CLIPPER Bus. A master gains control of the bus by asserting BR (Bus Request), then receiving BG (Bus Grant) from bus arbitration logic.

Slave:
A device that is being addressed via the CLIPPER Bus. A slave is addressed by a master.

Memory interface:
Logic which controls data transfer to and from main memory.

I/O Write:
Write by an I/O device.

I/O Read:
Read by an I/O device.

9.4. Bus Protocol

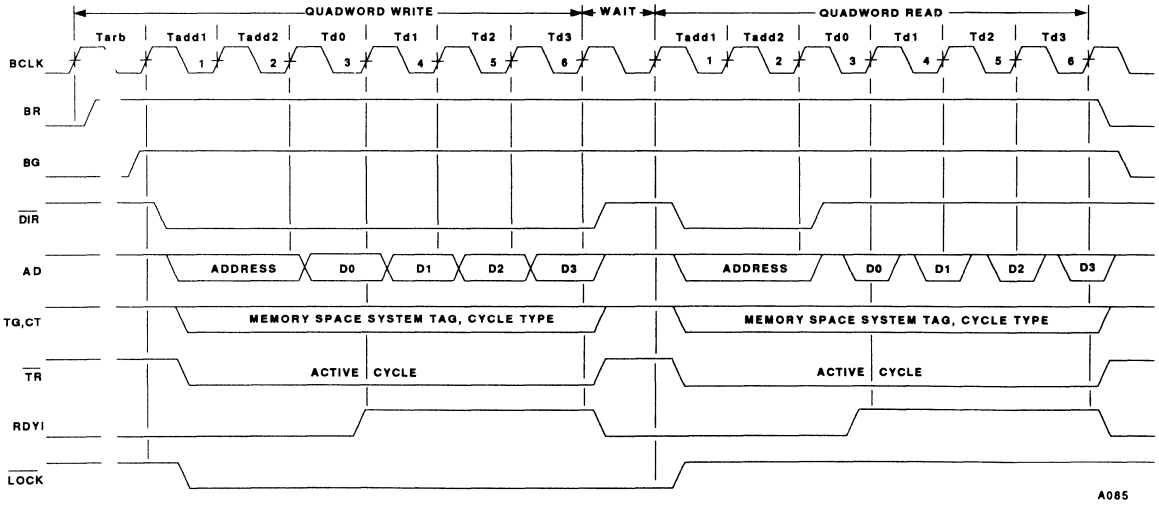
CLIPPER Bus operations are governed by the following rules:

(1) A bus master cannot introduce wait states. This requires that a bus master be able to transfer data at the maximum rate allowed by the bus protocol.
(2) Slaves may introduce wait states by delaying the assertion of $\overline{RDY}$ (Ready) on the CLIPPER Bus. Wait states can be introduced between the address and data cycles of an operation by delaying $\overline{RDY}$ and between data words in a quadword transfer by toggling $\overline{RDY}$.
(3) All CLIPPER Bus signals must be sampled on the positive edge of BCLK.
(4) All signals must meet required set-up and hold times with respect to the positive transition edge of BCLK.Signals must not transition within the Tsu set-up time of BCLK rising edge or undefined states within the CLIPPER Module can result.
(5) If $\overline{RDYo}$ (Ready out) is asserted on the CLIPPER Bus by a CAMMU while $\overline{CBSY}$ is active, the memory interface must abort its data transfer.

9.4.1. Bus Arbitration

A bus arbitration unit which arbitrates control of the CLIPPER Bus must be implemented in systems utilizing the CLIPPER Module. The unit must be capable of receiving bus requests from each of the possible bus

Figure 40 Cache Line Replacement



masters via Bus Request lines ($\overline{BR}$x, where "x" identifies a particular bus master), and must be able to assert a Bus Grant ($\overline{BG}$x) for each bus master. The unit may support priority assignment such that in cases of multiple requests for the bus, the bus arbitration unit grants the bus to the highest-priority requesting device.

A bus master should hold $\overline{BR}$x asserted during its entire access of the CLIPPER Bus, then should release $\overline{BR}$x as soon as possible after completion of its data transfer in order to maintain high system throughput. The bus arbitration unit should hold $\overline{BG}$x asserted until the bus master has released $\overline{BR}$x.

Multiple Bus Operations
A bus master can execute multiple bus operations by holding its $\overline{BR}$x signal asserted until it has completed all its data transfers. Read-modify-write operations, for example, require that the bus masters executing the operations maintain control of the bus during the reads and following writes, and the bus master maintains this control by holding $\overline{BR}$x asserted until after completion of the write. Another example of a multiple-bus operation is the replacement of a cache line as a result of a cache miss. As shown in *Figure 40*, the operation consists of a quadword write of the cache line to memory if the line is dirty, followed by a quadword read of the replacement line into the cache.

9.4.2. Bus Control

The bus control signals indicate CLIPPER Bus operation status which is used to implement bus protocol and control, support Bus Watch, and give a bus master the means to secure the bus indefinitely in order to complete multiple bus operations.

A Ready Input (RDYi) tied to each CAMMU is used to synchronize data transfers between CLIPPER, I/O, and memory. When a CAMMU reads data, it holds the bus in a read state until the responding device asserts RDYi, indicating that the data to be read is on the bus. When a CAMMU writes data, it provides data on the clock following the address phase of the operation until the device being written to asserts RDYi, indicating that it has latched the data. RDYi is thus used to accommodate various response times of devices on the bus. This eliminates the need to introduce for all data transfers the number of wait states necessary to accommodate the slowest device on the bus.

Ready Out (RDYo) is asserted by the D-CAMMU during Bus Watch operation in response to an I/O read of memory data that is cached. RDYo is active only during this operation. If the memory page being read is tagged as a copy-back page, then changes to the page data in the cache are not copied to the page in main memory until the page is replaced by the operating system. If an I/O device reads data from memory that is cached, and if the cache has updated data that has not been copied to the memory location being read, the D-CAMMU asserts RDYo while asserting CBSYd. This aborts assertion of memory data. (The memory interface must be designed to abort the memory operation when both CBSYd and RDYo are asserted.) The D-CAMMU instead asserts the updated cache data on the CLIPPER Bus, which is read by the I/O device, and RDYo. In this way, transfer of valid data to I/O devices is assured.

Ready Out I-CAMMU (RDYoi) is asserted by the I-CAMMU to indicate assertion or latching of data in response to access by the D-CAMMU. Since only the D-CAMMU can access the I-CAMMU, this signal is tied only to the D-CAMMU.

Two Cache Busy signals, one for the I-CAMMU (CBSYi) and one for the D-CAMMU (CBSYd), are used to indicate CAMMU internal operations associated with Bus Watch. CBSYi and CBSYd may be ORed to form a single Cache Busy ($\overline{CBSY}$) signal on the CLIPPER Bus as shown in *Figure 37*. When a CAMMU Bus Watch mode is invoked during a memory access, the affected CAMMU asserts Cache Busy to indicate that it is checking whether the accessed data location is cached (see *Figures 55* and *56*).

If the bus operation is a write, the memory interface must not assert $\overline{RDYi}$ until $\overline{CBSY}$ is released by the CAMMU. This ensures that the CAMMU has time to update data in its cache before the bus operation is completed. If the operation is a read, the memory interface must not drive the drive the Address/Data bus until $\overline{CBSY}$ is released. This allows the CAMMU to abort assertion of data by the memory interface, and to provide cached data if required (see Ready Out description).

Transfer Request ($\overline{TR}$) is asserted by bus masters to indicate that CLIPPER Bus operations are in progress. While Transfer Request is asserted, no bus master other than the one controlling the bus can gain bus access.

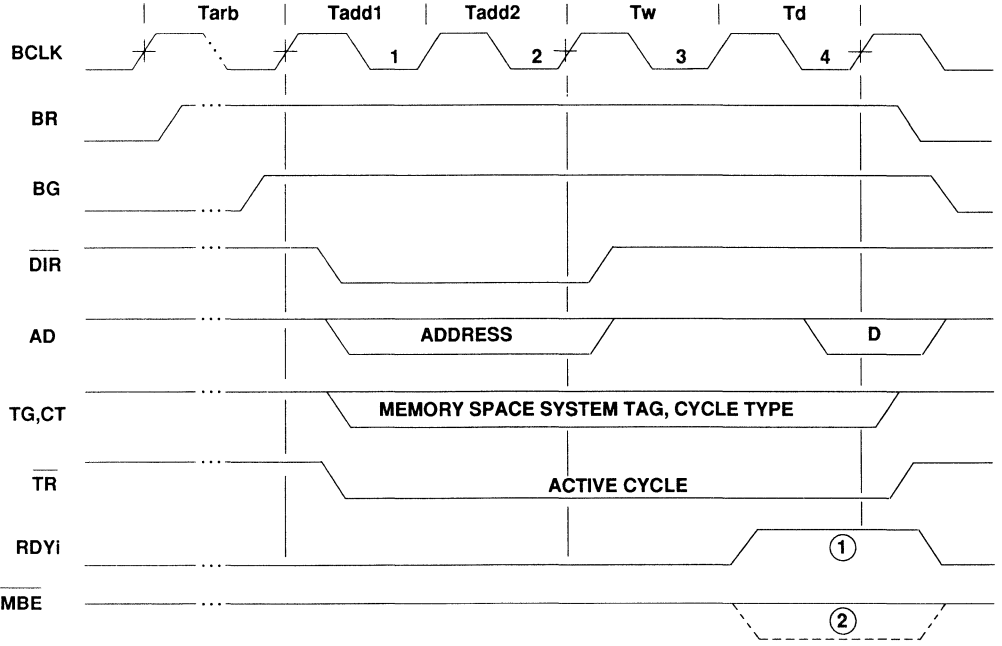
Lock ($\overline{LOCK}$) is used in dual-bus applications in which the CLIPPER Bus is interfaced to a separate I/O bus through a bus adapter or a dual port memory, this provides CLIPPER Bus masters with a means of maintaining control of the I/O bus or dual port memory throughout successive bus operations. $\overline{LOCK}$ becomes active during DTU page table access, cache line replacement, and read-modify-write operations.

Direction Control ($\overline{DIR}$) is used to control drive direction of TTL transceivers buffering the CLIPPER Module Address/Data bus. This signal controls the transceivers with proper timing for all bus operations, eliminating the need for such logic in the system.

9.4.3. Memory Errors

Memory data errors are reported with the Memory Single Bit Error/Retry ($\overline{MSBE}/\overline{RETRY}$) and the Memory Multiple Bit Error ($\overline{MMBE}$) signals. These signals are asserted by error detection and correction logic within the memory interface to indicate that a single-bit error has been detected and corrected ($\overline{MSBE}/\overline{RETRY}$), or that an uncorrectable multiple-bit error has occured ($\overline{MMBE}$). The signals, tied directly to the CAMMUs for fast response, force traps to error-handling routines. Timing for $\overline{MSBE}/\overline{RETRY}$ and $\overline{MMBE}$ is shown in *Figure 41*. Note that the signals are asserted during the same cycle that RDYi is active.

Figure 41 Single Word Read (1 Wait State)



NOTES:

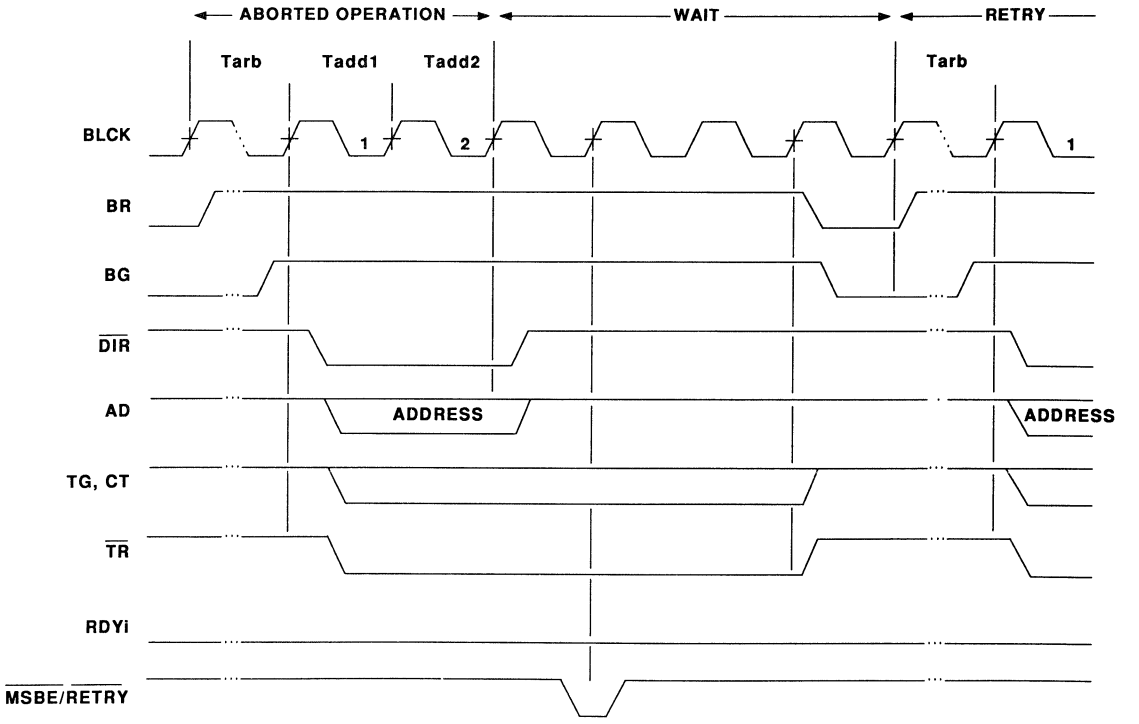
1. Timing for SINGLE WORD READ with NO WAIT STATES is shown in Figure 53.
2. These signals are asserted here by the memory interface to indicate memory data errors.

A086

In cases of multiple-bit and single-bit errors, the CAMMU Fault Register does not capture the addresses causing the errors. It is therefore necessary to design an address snapshot register into the system to capture addresses for use by the trap routine servicing single-bit errors if analysis of the errors is required.

The $\overline{\text{MSBE}}/\overline{\text{RETRY}}$ signal is also used to abort and retry CLIPPER Bus operations. If the signal is asserted during access of I/O space (TG = 4) while RDYi is inactive, the current bus operation is aborted and retried with no trap assertion. This feature is intended to resolve Bus Lockout in dual-bus systems, which occurs when a CLIPPER Bus master and an I/O processor (IOP) bus master simultaneously request access to each other's buses. For example, if CLIPPER has control of the CLIPPER Bus for attempted access of the I/O bus at the same time that an I/O bus master has control of the I/O bus for attempted access of the CLIPPER Bus, each bus master waits for the other to release its bus. Each bus master is therefore "locked out" from the other bus until one of the masters is forced to release its bus. Simple logic in the interbus interface logic can be used to assert $\overline{\text{MSBE}}/\overline{\text{RETRY}}$ whenever a CLIPPER request for the I/O bus occurs simultaneously with an IOP request for the CLIPPER Bus. This forces the CLIPPER to abort its bus operation and release the CLIPPER Bus, then re-arbitrate access to the CLIPPER Bus for a retry of the aborted operation. The IOP can gain access to the CLIPPER Bus after the abort by the Module but before the retry, thus eliminating the Bus Lockout condition. The CLIPPER then waits for completion of the I/O operation before gaining access to the I/O bus for the retry. Timing for bus retry is shown in *Figure 42*.

Figure 42 Bus Retry (Single Word Read Example)



NOTE:
Retry occurs when $\overline{\text{MSBE}}/\overline{\text{RETRY}}$ is asserted while RDYi is released during access of I/O space only (TG = 4).

A087

In summary, the Memory Single Bit Error/Retry signal operates as follows:

1. If the signal is asserted during any time other than access to I/O space and during the same clock cycle that RDYi is active, the signal reports a corrected memory single-bit error. This causes the CAMMU to generate a trap to the CPU.

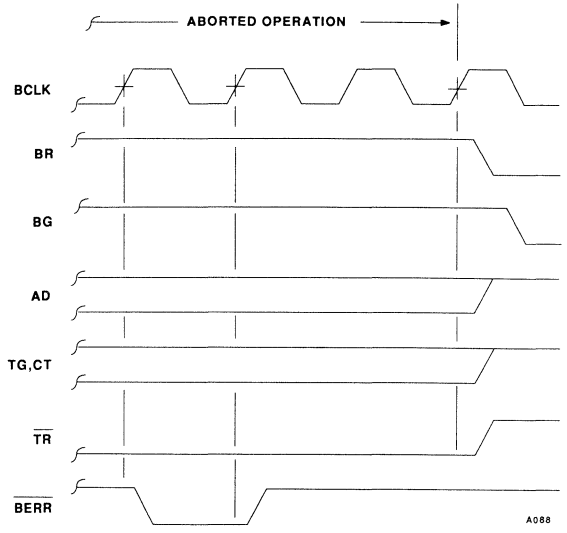
2. If the signal is asserted during access of I/O space (TG = 4) while RDYi is inactive, the current bus operation is aborted and retried by the master CAMMU with no trap assertion to the CPU.

9.4.4. Bus Error

A bus operation can be aborted by the assertion of $\overline{\text{BERR}}$ (Bus Error) by user-designed logic implemented in the CLIPPER system (see *Figure 43*). Bus error conditions should be detected by the bus error logic, which should then assert $\overline{\text{BERR}}$ and an interrupt request (via the interrupt logic). The CAMMU terminates the system bus access and releases the bus when it detects the assertion of $\overline{\text{BERR}}$. The CPU should use the interrupt request to vector to a routine designed to resolve the bus error condition.

$\overline{\text{BERR}}$ must be asserted by the bus error logic for one BCLK cycle. The state of the CLIPPER Bus associated with the bus error may be stored by the bus error logic for use by the bus error interrupt service routine.

Figure 43 Bus Error



9.4.5. Unrecoverable Fault

Some errors allow no clean means of recovery for continuation of program execution. These errors include the occurrence of a trap during execution of INTRAP or **reti**, and the detection of a fault during self-test (see *Section 9.4.9*). A trap during execution of INTRAP or **reti** can be avoided by ensuring that the Exception Vector Table is set up prior to the occurrence of a trap condition, and that the supervisor stack pointer always points to a valid page. No other conditions generate an unrecoverable fault.

Were the CPU to ignore these error conditions and continue execution, effects on the system could be catastrophic. A faulty or "lost" CPU could execute random writes to memory and I/O, for instance, corrupting data in both main memory and secondary storage. The CLIPPER CPU offers protection from catastrophic failure by stopping program execution immediately upon detection of one of the unrecoverable fault conditions, before the system is corrupted. It then asserts the Unrecoverable Fault signal ($\overline{\text{URF}}$) as a hardware indication that the CPU is halted due to an unrecoverable error, and that human intervention is required to correct the problem.

9.4.6. Wait States

Slow devices can introduce wait states by delaying assertion of $\overline{\text{RDY}}$ on the CLIPPER Bus during bus operations. Wait states consist of an integral number of BCLK periods during which time the master device remains in a "waiting" state until the slave device asserts $\overline{\text{RDY}}$ to indicate that it has asserted data on the bus (if a read operation by the master), or has read data from the bus (if a write). Wait states are further explained in the following descriptions of bus operations.

9.4.7. CLIPPER Bus Operations

Unless otherwise noted, the signal nomenclature used in this section describe the signals as shown in *Figure 37.*

— $\overline{RDY}$ is tied to $\overline{RDY}$o to form a single ready signal ($\overline{RDY}$). $\overline{RDY}$ is gated with RDYoi on the CLIPPER Module Interface.
— CBSYd and CBSYi on the CLIPPER Module Interface are gated to form a single ORed $\overline{CBSY}$ signal on the CLIPPER Bus.

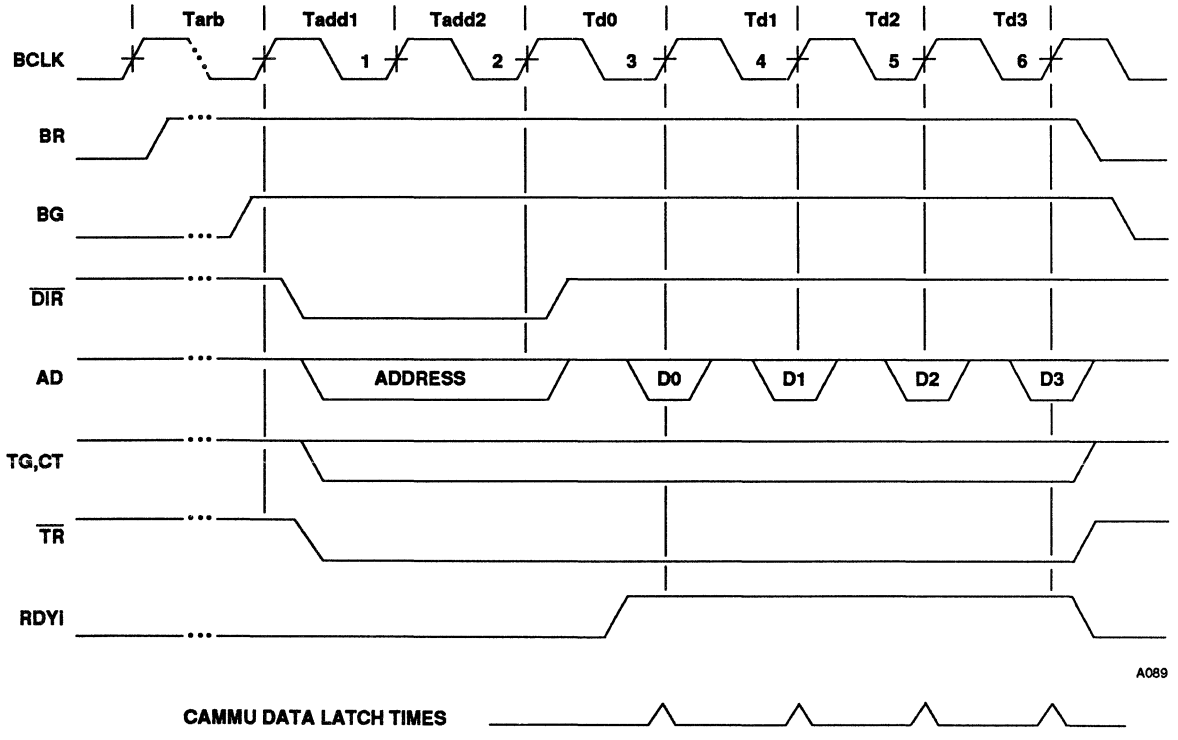
A CLIPPER Bus operation begins when a bus master requests the bus by asserting its Bus Request signal and receives Bus Grant from the bus arbitration unit. The bus master can then execute one of four bus operations: a read operation, a write operation, a global write operation, or a multiple memory access operation.

Read Operation

Upon receiving Bus Grant ($\overline{BGx}$), the master (possibly a CAMMU) asserts $\overline{TR}$, Memory Space System Tag (TG<2:0>), Cycle Type (CT<5:0>), and a real address (AD<31:0>) on the bus. The Memory Space System Tag, Cycle Type, and $\overline{TR}$ signals remain asserted during the entire operation. However, the bus master three-states the multiplexed address/data lines (AD<31:0>) after two BLCK cycles to make the lines available for data transfer by the slave device (see *Figures 41* and *44*).

The bus master then waits for the slave device to assert $\overline{RDY}$ (CAMMU RDYi signal), indicating that the data is on the bus. The master latches the data on the same positive transition of BCLK that it detects assertion of $\overline{RDY}$. The slave device can respond with data immediately after the address/data lines are three-stated

Figure 44 Quadword Read (No Wait States)



by the bus master, or can respond later as required by delaying assertion of RDY thereby introducing wait states.

The minimum number of BCLK cycles required for a read operation is three, excluding bus arbitration requirements: two cycles are required for assertion of address, and at least one cycle is required for the data transfer.

Bus Watch During Read Operations

During I/O reads of private, copy-back main memory space (i.e., TG=2), each CAMMU with Watch I/O Reads enabled asserts CBSY, indicating to other bus devices that it is checking for dirty cached data (cached data not yet written to main memory) corresponding to the main memory location being accessed by the I/O device. If it finds dirty data, it asserts the data on the AD bus, asserts RDYo, and releases CBSY. The I/O master must then latch the data on the positive BCLK transition following assertion of RDYo. If the data is not cached or the cached data is not dirty, the affected CAMMU releases CBSY, allowing transfer of main memory data to the bus master. This Bus Watch sequence applies to both single-word reads and quadword reads. If Bus Watch intervention occurs during quadword reads, however, the affected CAMMU will return all four data words if one or more data words is dirty.

The main memory interface must monitor the CLIPPER Bus CBSY and RDYo (which can be tied to the RDY signal) lines and allow main memory data response on AD<31:0> only if CBSY (CBSYi or CBSYd on the CLIPPER Module Interface) is not asserted, indicating that there will be no CAMMU intervention resulting from CAMMU Bus Watch. If RDYo is asserted by the CAMMU while CBSY is asserted, the memory interface must abort the memory read because the CAMMU is responding with more recent cache data. Memory monitoring of CBSY is summarized in *Figure 45.*

Figure 45 Memory Interface CBSY Monitoring

	CPU				I/O				
	READ		WRITE		READ		WRITE		
	SINGLE	QUADW	SINGLE	QUADW	SINGLE	QUADW	SINGLE	QUADW	TG<2:0>:
PRIVATE W.T.	-	-	-	-	-	-	m(1)	-(3)	000
PRIVATE C.B.	-	-	-	-	m(2)	m(2)	m(1)	-(3)	010
SHARED (W.T.)	-	-	m(1)	-(3)	-	-	m(1)	-(3)	001
NONCACHEABLE	-	-	-	-	-	-	-	-	011
CT <5:2>	0100	0101	0000	0001	1100	1101	1000	1001	

A090

m=monitor CBSY

NOTES:

(1) Single Word Writes: CAMMU updates cache on hit; memory interface must not assert RDY until after CBSY is released.

(2) I/O Reads: CAMMU provides data on cache hit; memory interface must not assert RDY until after CBSY is released, and may enter into memory data that is supplied by the CAMMU (indicated by assertion of RDY and CBSY by the CAMMU) in order to support Clear Dirty operation if required.

(3) Quadword Writes: The memory interface proceeds normally (doesn't monitor CBSY) if the bus arbiter inhibits granting of the bus again while CBSY is asserted; otherwise the memory interface must not assert RDY until after CBSY is released.

The Clear Valid option, if enabled, requires memory to update its contents with the more current (dirty) data supplied by the cache for an I/O quadword read, unless the data will not be read by another I/O device (see *Section 7.6.4*). Use of this option saves a write of the dirty data to memory when the cache line is replaced. The Clear Valid option is enabled by setting the Clear Valid flag in the Control Register. Memory support for this option requires that the memory transition from a memory read operation to a memory write operation when both $\overline{CBSY}$ and $\overline{RDY}$ are asserted by the D-CAMMU, and that the memory not be allowed any wait states between the individual quadwords supplied by the CAMMU.

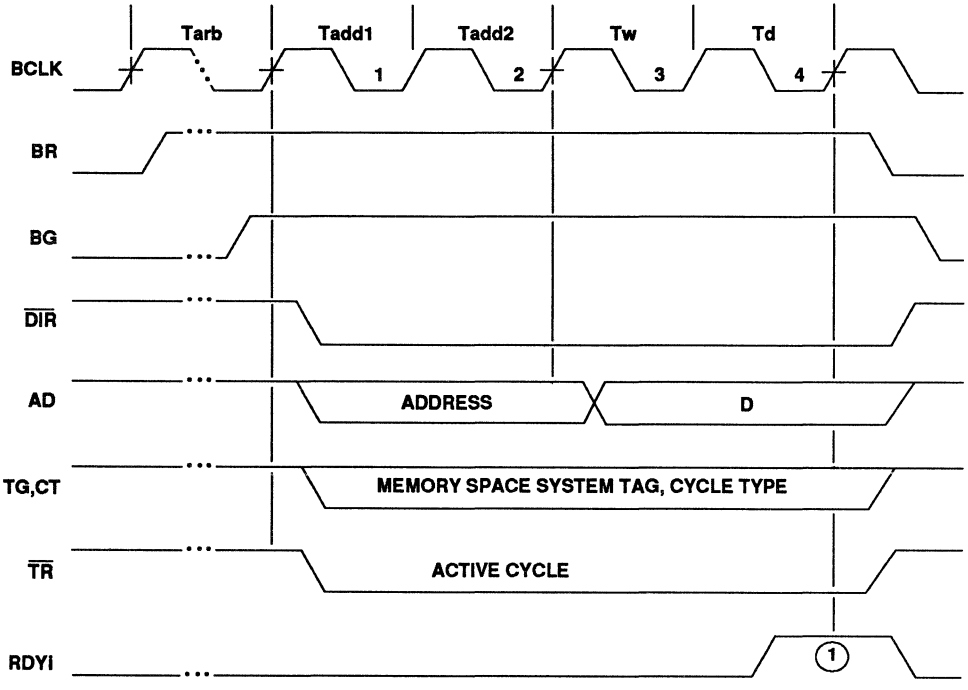
CAMMUs normally require 4 MCLK (120ns @ 66.7 MHz OSC frequency) cycles for Bus Watch checking. During this time the memory interface can proceed with the read operation without delay up to, but not including, assertion of $\overline{RDY}$. As a result of this parallelism, CAMMU Bus Watch operation results in little impact on CLIPPER Bus utilization.

Write Operation

Signal assertion and timing associated with a write operation are similar to those associated with a read operation. Upon receiving Bus Grant ($\overline{BGx}$), the master (possibly a CAMMU) asserts $\overline{TR}$, Memory Space System Tag (TG<2:0>), Cycle Type (CT<5:0>), and a real address (AD<31:0>) on the CLIPPER Bus. The Memory Space System Tag, Cycle Type, and $\overline{TR}$ signals remain asserted during the entire operation (see *Figures 46-48*).

After two BCLK cycles, however, the bus master replaces the address on the AD lines with the data to be written, and holds the data on the lines until the

Figure 46 Single Word Write (1 Wait State)



NOTE:
Timing for SINGLE WORD WRITE with NO WAIT STATES is shown in Figure 54.

A091

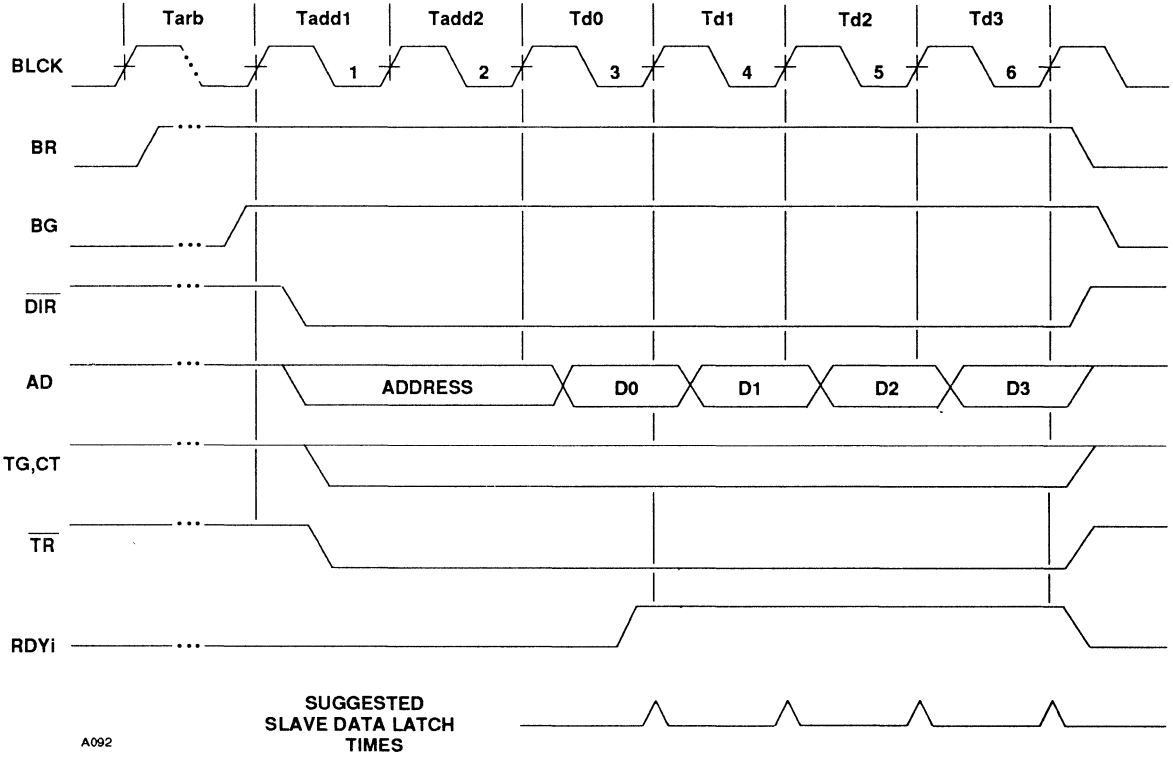
slave acknowledges latching of the data by asserting $\overline{RDY}$ (CAMMU RDYi signal) on the CLIPPER Bus. The slave can assert $\overline{RDY}$ when ready, allowing wait states as required.

If the operation is a single-word write, the bus master releases the bus immediately following detection of asserted $\overline{RDY}$ (CAMMU RDYi signal). If the operation is a quadword write, the bus master asserts the second, third, and fourth data words of the quadword data transfer during successive BCLK cycles following detection of asserted $\overline{RDY}$ The slave device can introduce wait states between assertion of the quadword address by the master and latching of the first data word, and between latching of the individual data words

The minimum number of BCLK cycles required for a write operation is three, excluding bus arbitration requirements. Two cycles are required for assertion of the address, and at least one cycle is required for the data transfer.

Bus Watch During Write Operations
A CAMMU invokes Watch I/O Writes, if enabled, when an I/O device writes to its cacheable main memory space; and invokes Watch CPU Writes, if enabled, when a CPU (via a D-CAMMU) writes to its shared cacheable main memory space. Both Bus Watch modes, when invoked, function identically. If the write operation invoking one of these modes is a single-word write operation, the affected CAMMU updates the cache with the data written to the main memory if the main memory data has been cached. If the write operation is a quadword write, the affected CAMMU invalidates the cache line containing the data addressed in main memory.

Figure 47 Quadword Write (No Wait States)



A CAMMU normally requires 4 MCLK (120ns @ 66.7 MHz OSC frequency) cycles to complete one of these Bus Watch operations. However, if the Bus Watch operation occurs while the CPU is accessing the CAMMU, the CAMMU may require more time to complete the operation and will keep asserting $\overline{CBSY}$ to inhibit further bus operations until it has completed the task. The bus master, however, can complete the write operation while the CAMMU executes its Bus Watch operation, so Bus Watch impact on CLIPPER Bus utilization is minimal.

If $\overline{CBSY}$ is asserted during a byte, halfword, or word memory write operation, the memory interface must not assert $\overline{RDY}$ until after $\overline{CBSY}$ is released. This ensures that the data remains on the bus long enough for entry by a CAMMU into its cache in case of a hit

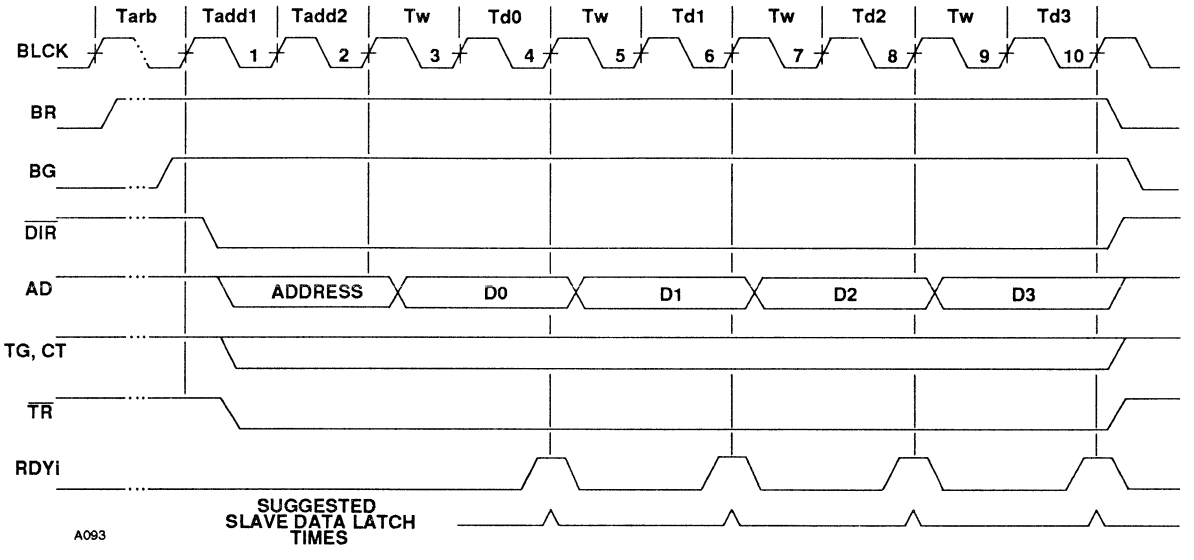
If $\overline{CBSY}$ is asserted during a quadword write, the memory interface can assert $\overline{RDY}$ normally without regard to the state of $\overline{CBSY}$ because in case of a cache hit, the affected CAMMU invalidates the hit line and does not require data to be present on the bus. However, the system bus arbiter must not grant the bus to a new bus master until $\overline{CBSY}$ is released, indicating that all CAMMUs are ready for a new operation. Alternatively, the memory interface can delay assertion of $\overline{RDY}$ until $\overline{CBSY}$ is released as in the byte/halfword/word write case, eliminating the need for the bus arbiter to monitor $\overline{CBSY}$. In any case, a new CLIPPER Bus operation should not be allowed to begin while $\overline{CBSY}$ is asserted. Memory monitoring of $\overline{CBSY}$ is summarized in *Figure 45*.

Global Write Operation

A global write is used in a system utilizing multiple CLIPPER Modules to reset the TLBs or caches, or to write to specific TLB lines or registers in all CAMMUs in the system except the companion D-CAMMU of the CLIPPER CPU executing the global write. CLIPPER Module global addressing is explained in *Section 7.6.6, CAMMU Register Access*. Non-CLIPPER bus masters can execute global writes by setting CT<3:2> HIGH during the otherwise normal write operations. Note, however, that CAMMUs respond only to global writes to CAMMU I/O space real addresses Cnn and Dnn (Hex).

Each CAMMU being written to by a global write asserts CBSYi (if an I-CAMMU) or CBSYd (if a D-CAMMU)

Figure 48 Quadword Write (4 Wait States)



during the write to inhibit further bus activity until it has completed internal tasks associated with the write. System logic is required which detects global writes and asserts $\overline{RDY}$ when $\overline{CBSY}$ is released.

Read-Modify-Write Operation

A read-modify-write bus operation is a combination of a read operation, followed by a write operation. Timing and protocol associated with the read and the write phases of a read-modify-write operation are the same as for single reads and writes; however, Bus Request ($\overline{BR}$) must be asserted by the bus master during the entire operation.

Read-modify-write operations are performed during execution of the **tsts** (test-and-set instruction). However, the write part of the read-modify-write is performed only if the bit to be tested is zero; if the bit has already been set (AD<31> = 1), the bus operation is terminated. Timing for this operation is shown in *Figure 49.*

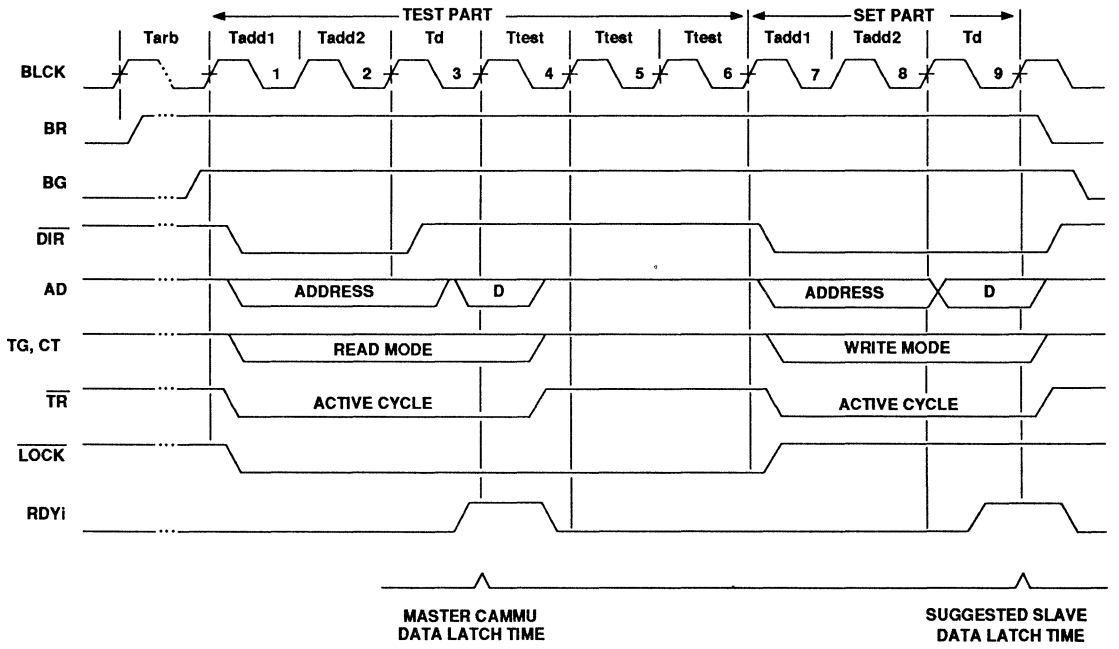
Read-modify-write operations are also performed during address translation when the DTU accesses main memory twice to read the Page Table Directory and Page Table in main memory, and follows with a write to the Page Table if the Referenced or Dirty flags must be modified. Timing for this operation is shown in *Figure 50.*

9.4.8. Interrupt Bus

The CLIPPER Bus includes a separate interrupt bus, $\overline{IVEC}$<7:0>, tied directly to the CPU. This bus allows interrupt levels and numbers to be transferred to the CPU without regard to CLIPPER Bus activity, thereby reducing CPU interrupt response time and increasing effective CLIPPER Bus bandwidth. (See *Section 6.3, Interrupts.*)

An interrupt controller must be implemented in a CLIPPER system. In cases of multiple interrupt requests, it must select between the interrupts, asserting

Figure 49 Read-Modify-Write (Test and Set)



NOTE:
Test Part - CAMMU determines if the set part of the operation is needed.

A094

the interrupt with highest priority. The interrupt controller must assert an interrupt request and its associated interrupt vector number together on the same positive transition edge of BCLK. The interrupt vector number can change to a higher priority on any BCLK. The CPU uses the interrupt level and number present on the $\overline{IVEC}$ bus when it detects $\overline{IRQ}$ release on a rising edge of BCLK, then releases $\overline{IACK}$ during the following BCLK period.

9.4.9. Diagnostics Control

The CLIPPER Module executes diagnostic routines following release of $\overline{RESET}$ if Apply Diagnostics ($\overline{URDIAG}$) is asserted during the two BCLK cycles following the release of $\overline{RESET}$. Then it begins execution at supervisor virtual address 6000H, which is mapped by the HTLB to real address 0 of Boot space.

The state of $\overline{URDIAG}$ during the two BCLK cycles following release of $\overline{RESET}$ determines whether the CLIPPER Module CPU executes internal diagnostics before executing from boot code (see *Figure 35*). This is a powerful feature of the module which allows self test of major functions of the CPU without test equipment, and without removal of the chip. Failure during diagnostics is reported by assertion of the Unrecoverable Fault ($\overline{URF}$) CLIPPER Bus signal.

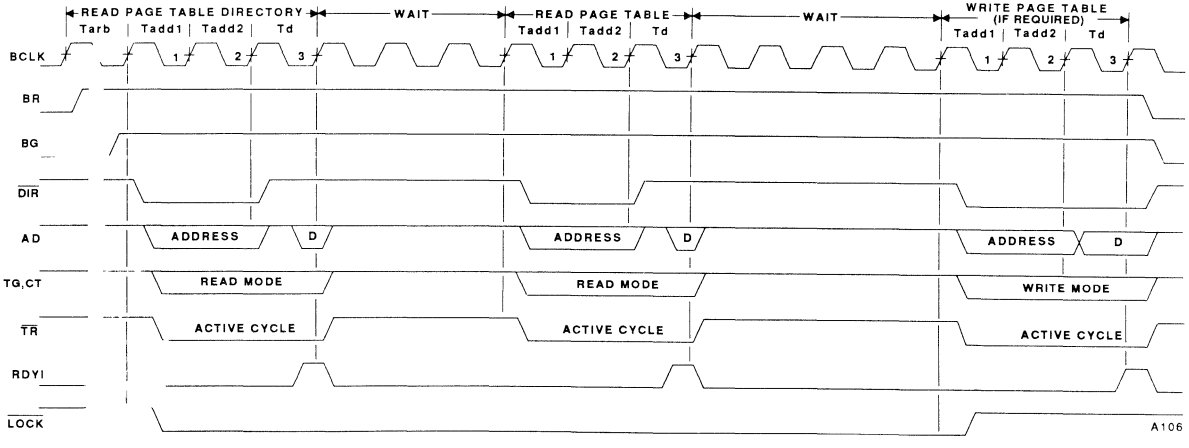
The CLIPPER Module self test checks most, but not all, of the major functions of the CPU. It is intended to be a first-level check of the CPU, and in fact is used to initially test individual CPU die during fabrication. The test executes approximately 700 instructions in about 4500 MCLK periods, using operands which test the CPU under worst-case conditions where possible. For example, worst-case carries, overflows, and sign extensions are tested.

The following CPU operations and functions are tested:

- Pipeline resource management
- Integer and floating-point execution units
- General-purpose register files
- Integer bypass mechanism
- Transition between supervisor and user modes
- Temporary (hidden) registers
- Macro branches
- All addressing mode computations
- Arithmetic shift, logical shift, and rotate instructions
- Integer multiply and divide
- Single- and double-precision floating-point instructions
- Floating-point status bits

CPU operations which require external response to instruction execution are generally not tested. These include exception conditions, branches, loads, stores, pushes and pops, and I-CAMMU and D-CAMMU interfaces.

Figure 50 Read-Modify-Write (DTU Operation)



9.4.10. Bus Timing

Figures 51-66 show CLIPPER Bus signal timing and test loads. Values for parameters indicated in the figures are listed in *Tables 15* and *17*.

BCLK is CMOS-compatible. All timing relationships in the timing figures are referenced to the 1.5 V midpoints of BCLK positive transitions.

The following are definitions of terms used in the figures:

Tarb (arbitration time)
BCLK cycle used for bus arbitration

Tadd1 (address time 1)
First BCLK cycle during which address is asserted on the bus

Tadd2 (address time 2)
Second BCLK cycle during which address is asserted on the bus

Td (data time)
BCLK non-wait state cycle during which data is asserted on the bus. For a quadword transfer, a numerical subscript (e.g., Td2) indicates which data word is asserted.

Tw (wait state time)
BCLK cycle during which the CLIPPER Module is in a wait state.

9.4.11. CLIPPER C100 Module Configurations

There are three CLIPPER C100 Module configurations, shown in *Figures 67 - 72*.

The C100 Module C100C1MLX is shown in *Figure 67*. Its connector mates with a user-supplied type BIC-Vero 905-72178F, or equivalent, male connector. Note that the numbering on the male connector is reversed relative to the CLIPPER Module connector.

The C100 Module C100C1BLX (*Figure 69*) mates with a user-supplied Samtec SD-125-T-18, or a McKenzie SBU-2X25-STGT-D131-VLI female socket connector, or the equivalent.

The C100 Module C100C1DLX (*Figure 71*) mates with a user-supplied McKenzie PH1-225/100 - 32G male connector or the equivalent.

9.4.12. Oscillator Connection

An external oscillator must be provided by the user to drive the clock control chip on the CLIPPER Module. The oscillator frequency must be twice the required MCLK frequency, with a duty cycle between 60/40 and 40/60. The oscillator should be placed as close to the connector as possible.

Table 15 AC Characteristics V_{CC} = 5.0 V ± 5%, T_A = 0 to +55°C

Symbol	Characteristic	33 MHz C100		Unit
		Min	Max	
t_{SU}	Setup Time	15.0		ns
t_H	Hold Time	0		ns
t_{CO1}	Clock to Transition Time[1]	0	20	ns
t_{CO2}		0	17	ns
t_{CO3}		0	15	ns
t_R	Output Rise Time[1,2]		11	ns
t_F	Output Fall Time[1,2]		7	ns

Notes:
1.Transition, rise, and fall times are for a 50pF external capacitive load (see *Figure 64*).
2.All outputs except BCLK.
3.To guarantee setup times, the input signals must have rise and fall times ≤ 4ns.

Table 16 DC Characteristics V_{CC} = 5.0 V ± 5%, T_A = 0 to +55°C

Symbol	Characteristic	Conditions	Min	Max	Unit
V_{IH}	Input HIGH Voltage		2.0	V_{CC}	V
V_{IL}	Input LOW Voltage		-0.5	0.8	V
V_{OH}	Output HIGH Voltage[1]	V_{CC} = 4.75 V I_{OH} = -20µA	4.7		V
		V_{CC} = 4.35 V I_{OH} = -2mA	4.3		V
V_{OL}	Output LOW Voltage[1]	V_{CC} = 5.25 V I_{OL} = +20µA	.2	0.4	V
		V_{CC} = 5.25 V I_{OL} = +2mA		0.45	V
I_{IN}	Input Leakage Current	V_{IN} = 0 to 5.25 V Inputs Only		±10	µA
I_{IH}	Input HIGH Current R_P = 220 ohms	Bidirectional I/O Only		±10	µA
I_{IL}	Input LOW Current R_P = 220 ohms	Bidirectional I/O Only V_{IL} = 0.55 V		-22	mA
C_{IN}	Input Capacitance	Inputs		18	pF
		Bidirectional I/O		28	
I_{CC}	Supply Current	T_A = 0°C, V_{CC} = 5.25 V		1.2	A
P_D	Power Dissipation	f_{OSC} = 66.7 MHz BCLK load = 100pF		6.0	W

1. I_{OH}, I_{OL} and I_{IL} parameters are a function of the value of Module pull-up resistor R_P.

Table 17 AC Characteristics - OSC, BCLK V_{CC} = 5.0 V ± 5%, C_L = 200pF, T_A = 0 to +55°C

Symbol	Characteristic	Conditions	33 MHz C100		Unit
			Min	Max	
f_{OSC}	Oscillator Frequency		2.0	66.7	MHz
t_{OSC}	Oscillator Cycle Time		15	500	ns
t_{OSCH} t_{OSCL}	Oscillator Pulse Width	t_{OSC} = Min	6	9	ns
t_{OSCR} t_{OSCF}	Oscillator Rise and Fall Time			6.0	ns
t_C	BCLK Cycle Time	RATE = LOW	60		ns
		RATE = HIGH	120		ns
t_{CH}	BCLK Pulse Width	t_C = Min (RATE = LOW)	27	33	ns
		t_C = Min (RATE = HIGH)	54	66	ns
t_R t_F	BCLK Rise and Fall Time (BCLK)			5.0	ns

Note
BCLK rise and fall times are for a 100pF capacitive load (see *Figure 65*). This load should not be exceeded to ensure proper operation.

Table 18 DC Characteristics - BCLK V_{CC} = 5.0 V ± 5%, T_A = 0 to +55°C

Symbol	Characteristic	Conditions	Min	Max	Unit
V_{OH}	Output HIGH Voltage[1]	I_{OH} = +100mA	4.3		V
V_{OL}	Output LOW Voltage[2]	I_{OL} = +100mA		0.45	V

Notes:
1. V_{OH} worst case occurs with V_{CC} = 4.75 V.
2. V_{OL} worst case occurs with V_{CC} = 5.25 V.

Table 19 DC Characteristics - OSC V_{CC} = 5.0 V ± 5%, T_A = 0 to +55°C

Symbol	Characteristic	Conditions	Min	Max	Unit
V_{IH}	Input HIGH Voltage		4.0	V_{CC} +0.5	V
V_{IL}	Input LOW Voltage		GND -0.5	0.5	V

Figure 51 AC Measurement Points

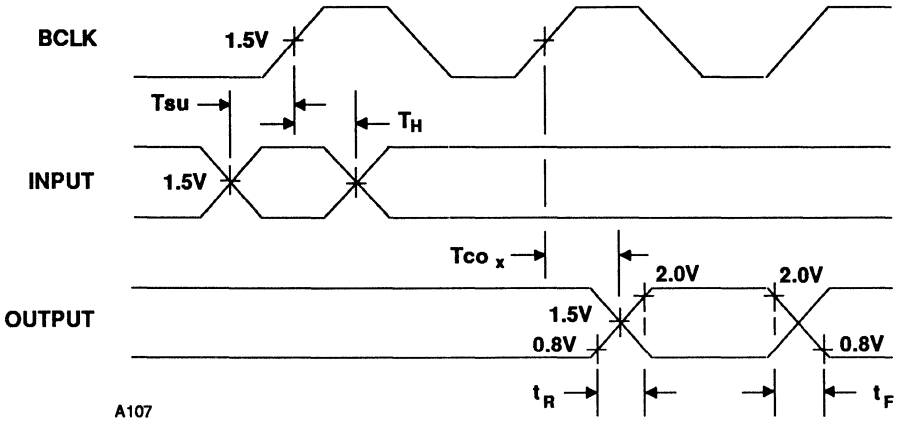


Figure 52 AC Measurement Points, OSC and BCLK

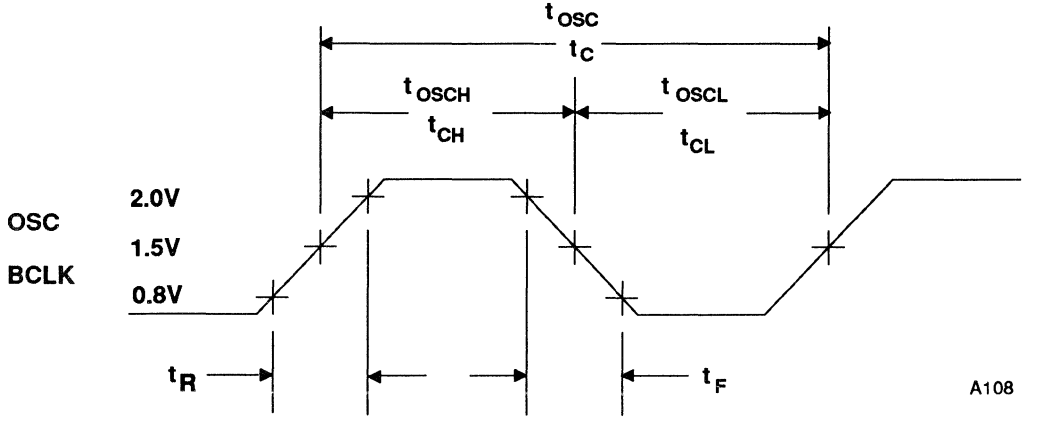
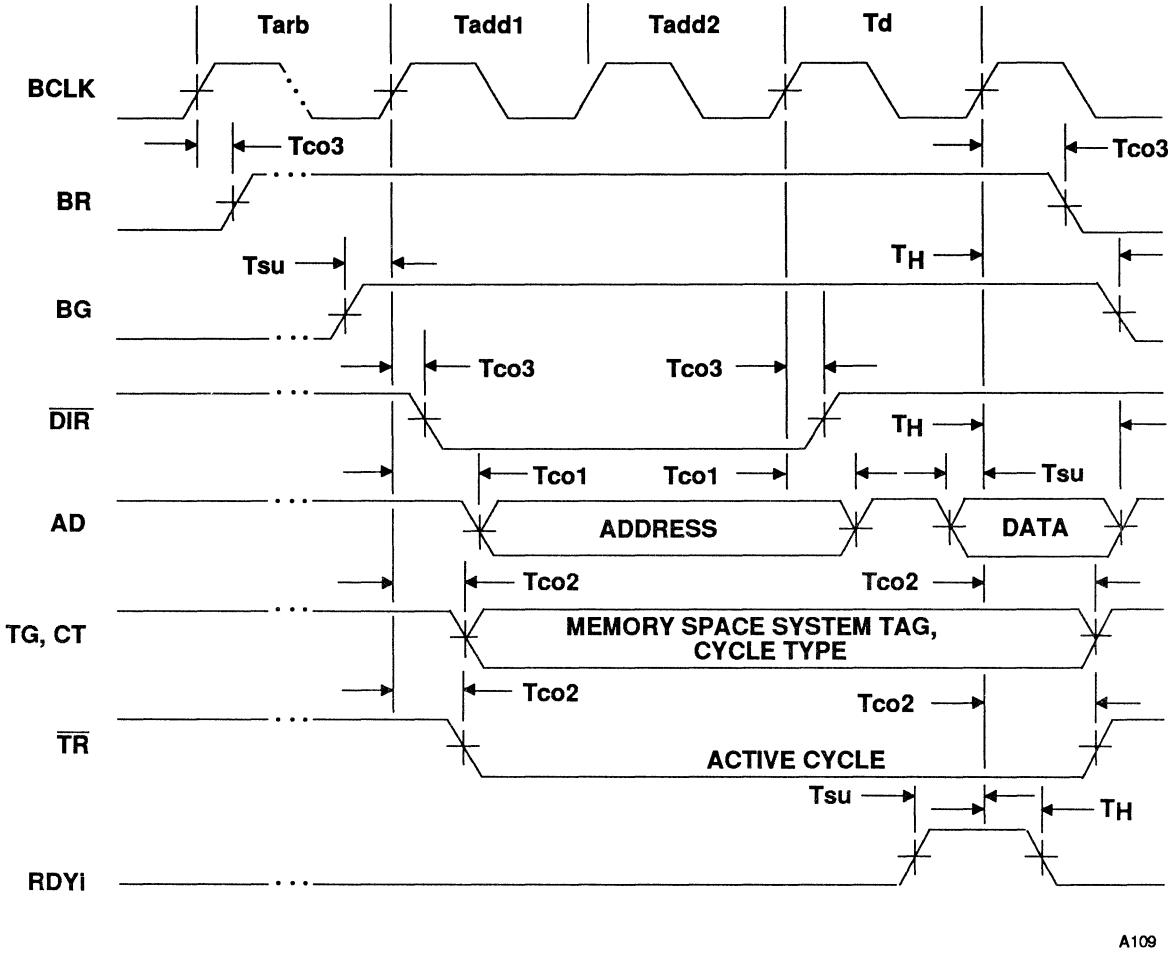
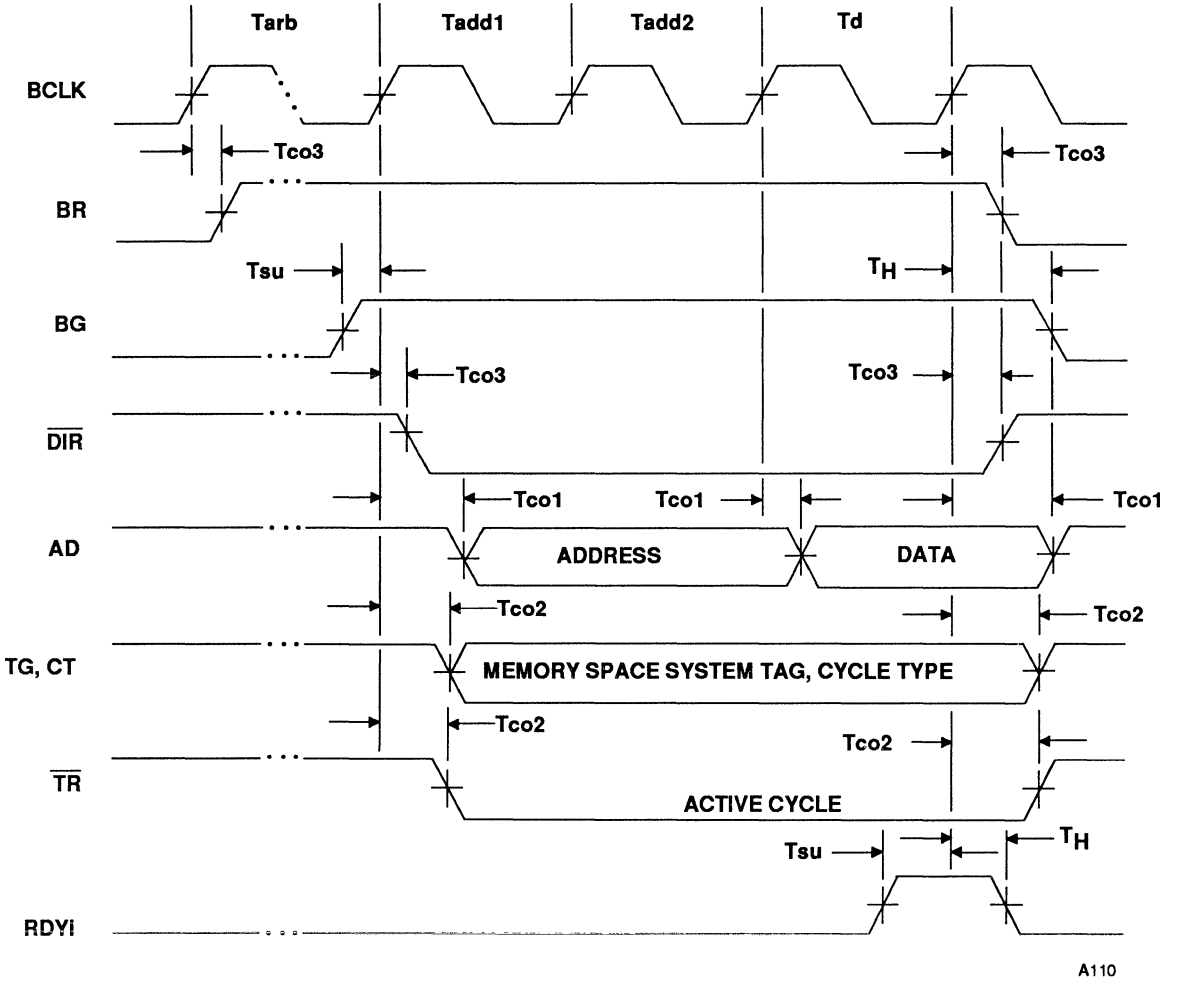


Figure 53 Read Timing Diagram



NOTE:
Timing measurements are referenced to and from a signal midpoint voltage of 1.5 volts unless otherwise stated.

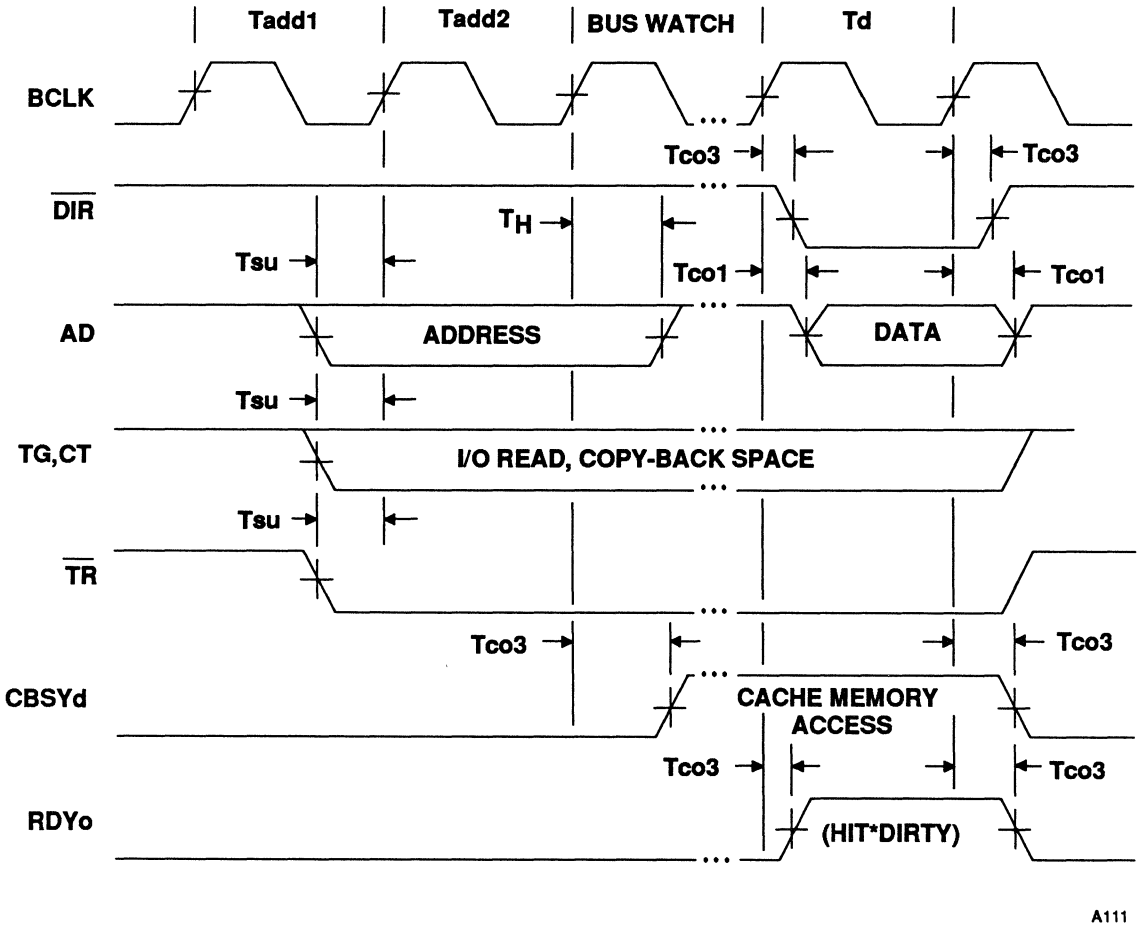
Figure 54 Write Timing Diagram



NOTE:

Timing measurements are referenced to and from a signal midpoint voltage of 1.5 volts unless otherwise stated.

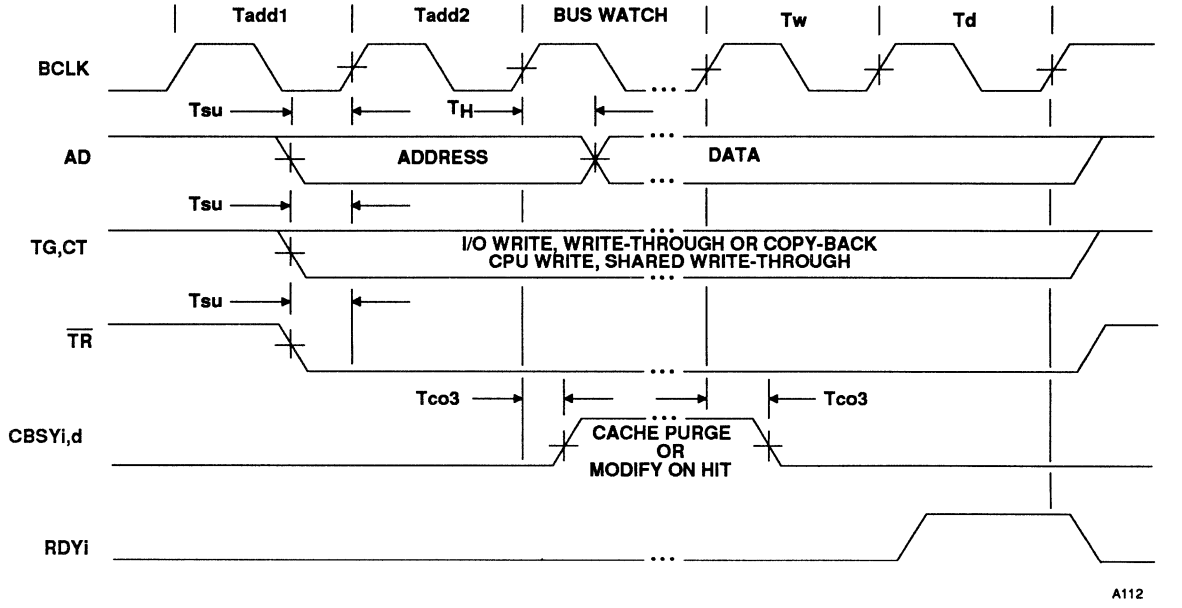
Figure 55 Watch I/O Reads



NOTES:

1. I/O READ: Device reads from main memory (e.g., main memory to disk).
2. Timing measurements are referenced to and from a midpoint signal voltage of 1.5 volts unless otherwise stated.

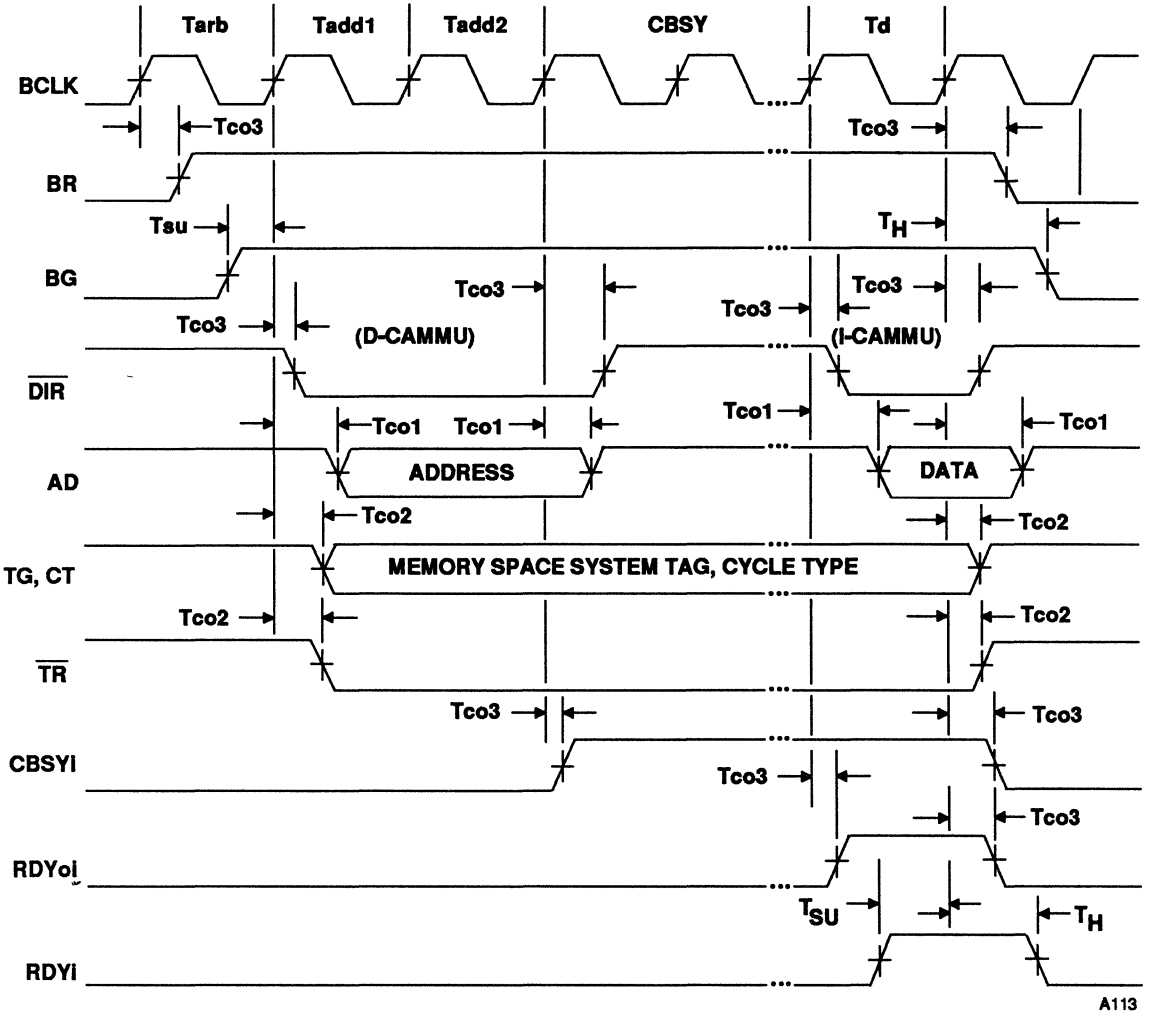
Figure 56 Watch CPU and I/O Writes



NOTES:

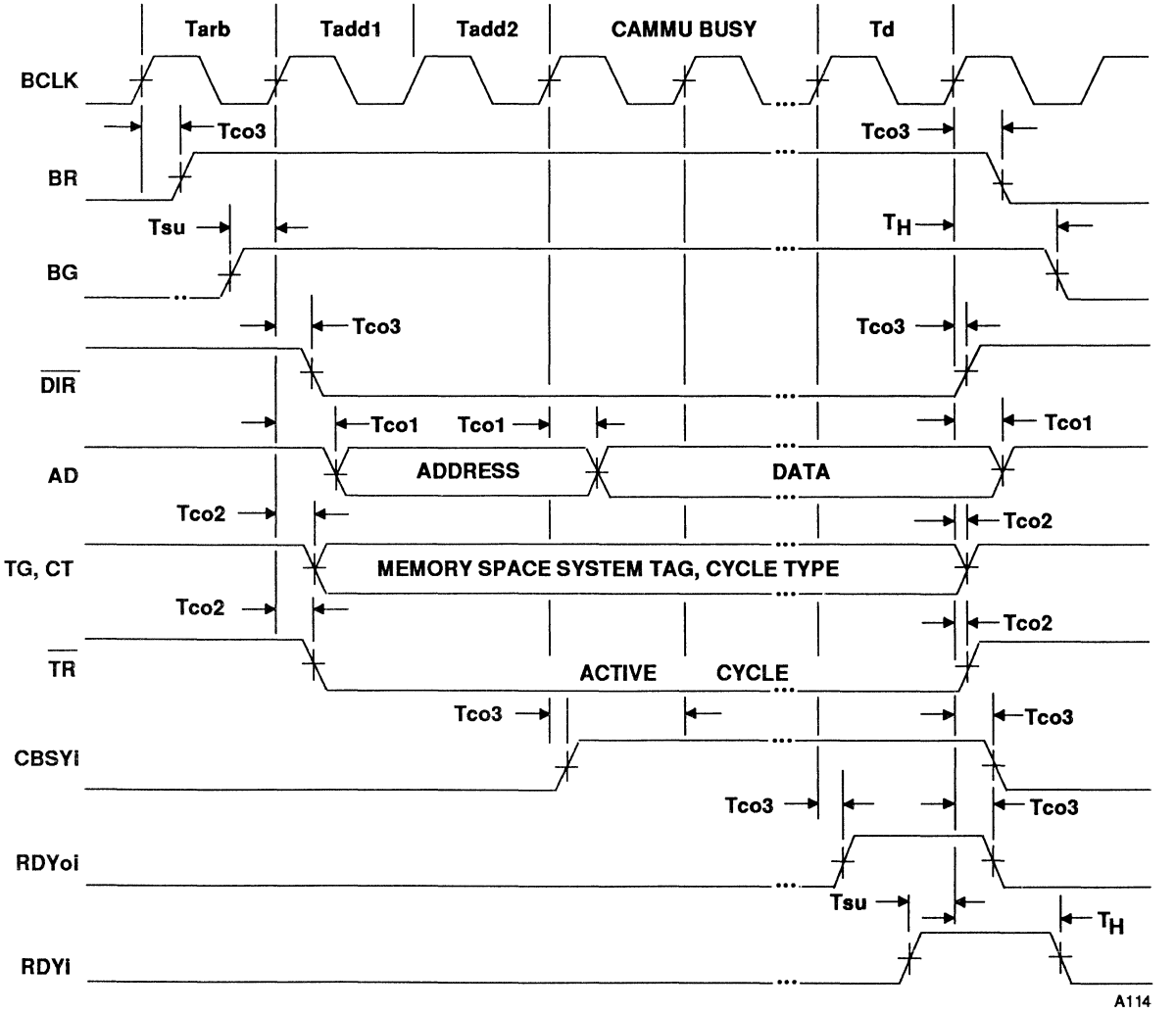
1. I/O WRITE: I/O device writes into main memory (e.g., disk to main memory).
2. WRITE TO SHARED PAGE One of the CAMMUs writes into the shared area of main memory.
3. Timing measurements are referenced to and from a midpoint signal voltage of 1.5 volts unless otherwise stated.
4. RDYi is asserted by the memory interface.

Figure 57 D-CAMMU Read from Companion I-CAMMU



NOTES:
1. Timing measurements are referenced to and from a midpoint signal voltage of 1.5 volts unless otherwise stated.
2. I-CAMMU internal registers can be accessed through the companion D-CAMMU (D-CAMMU of same CLIPPER module).

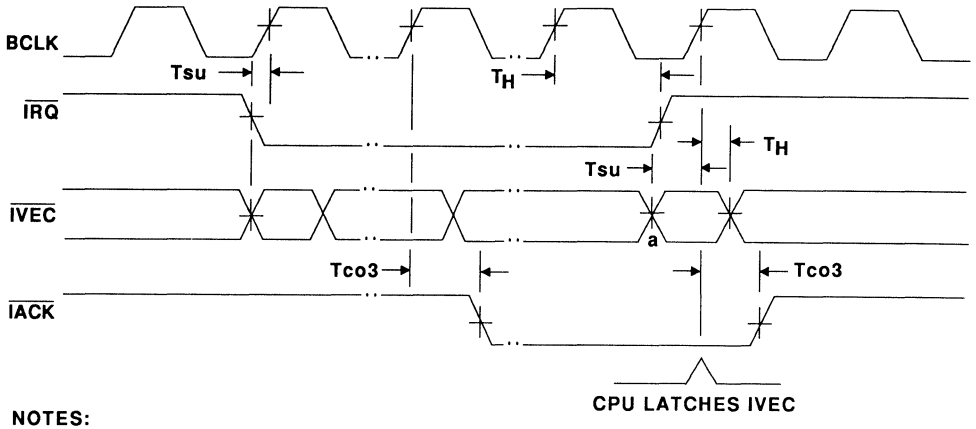
Figure 58 D-CAMMU Write into Companion I-CAMMU



NOTES:

1. Timing measurements are referenced to and from a midpoint signal voltage of 1.5 volts unless otherwise stated.
2. I-CAMMU internal registers can be accessed through the companion D-CAMMU (D-CAMMU of same CLIPPER module).

Figure 59 Maskable Interrupt Request/Acknowledge Timing

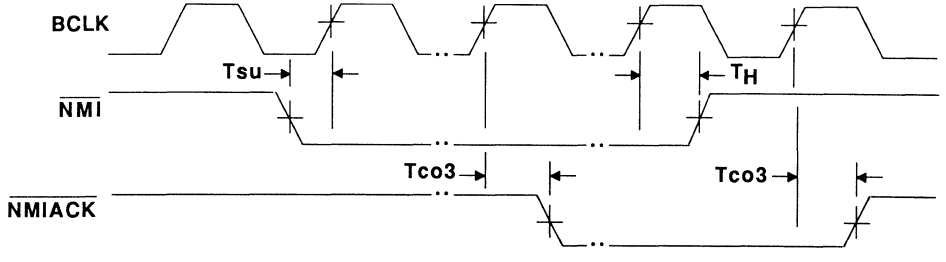


NOTES:

1. After IACK returns high the IRQ line must be high for one clock before another interrupt request returns IRQ low.
2. The IVEC lines can change only to a higher priority when IRQ is low. The higher priority value must be on the IVEC lines by "a".
3. CPU latches IVEC on the rising edge of BCLK following release of IRQ. The CPU releases IACK during the BCLK period following release of IRQ.
4. Timing measurements are referenced to and from midpoint voltages of 1.5 volts unless otherwise stated.

A115

Figure 60 Non-Maskable Interrupt Request/Acknowledge Timing



A116

NOTES:

1. After NMIACK returns high the NMI line must be high for one clock before another nonmaskable interrupt request returns NMI low.
2. Timing measurements are referenced to and from midpoint signal voltages of 1.5 volts unless otherwise stated.

Figure 61 $\overline{\text{LOCK}}$ Timing

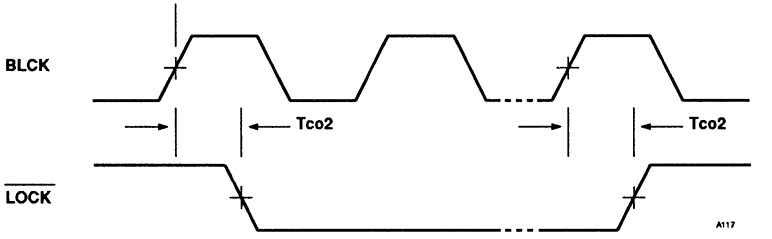


Figure 62 $\overline{\text{URF}}$ Timing

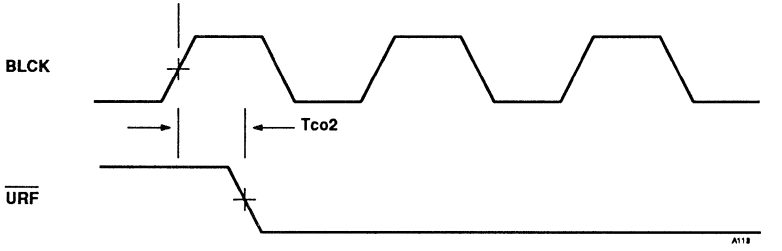


Figure 63 $\overline{\text{RESET}}$ and $\overline{\text{URDIAG}}$ Timing

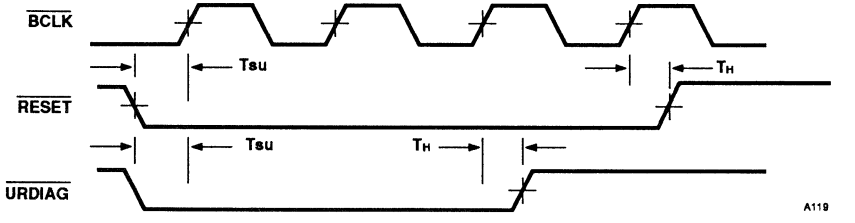


Figure 64 Module Output Test Load

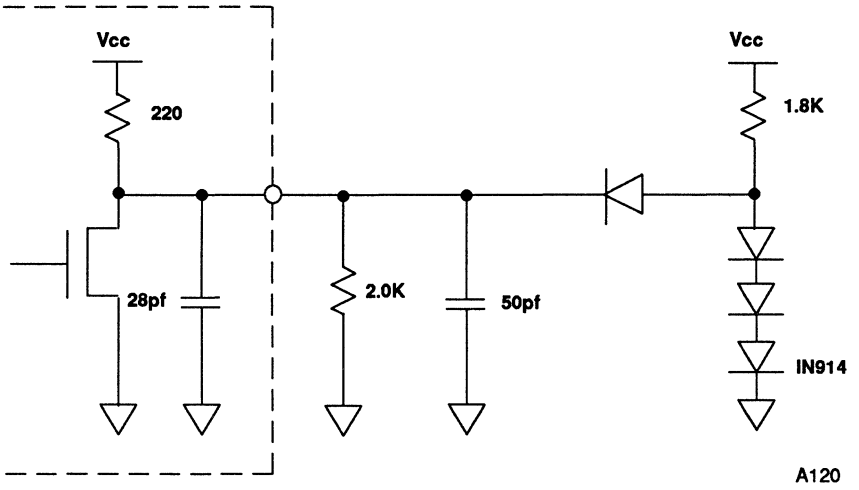


Figure 65 BLCK Output Test Load

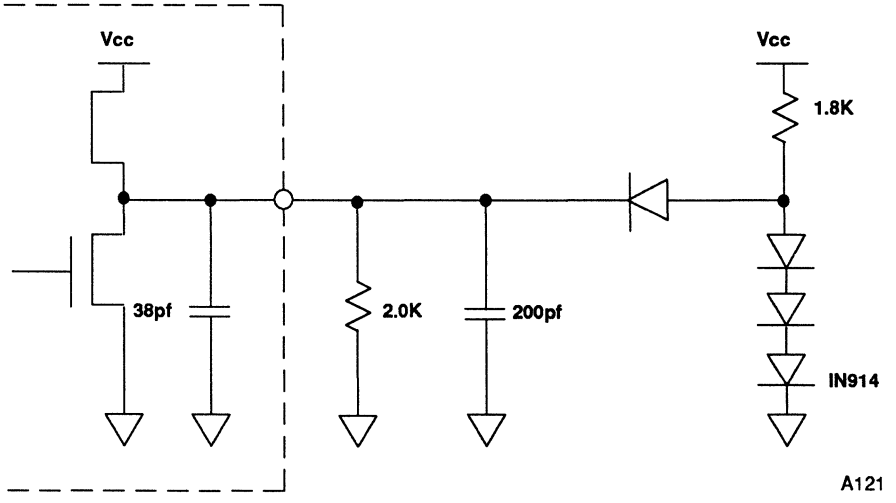


Figure 66 Maximum Output Delay vs. Capacitive Loading

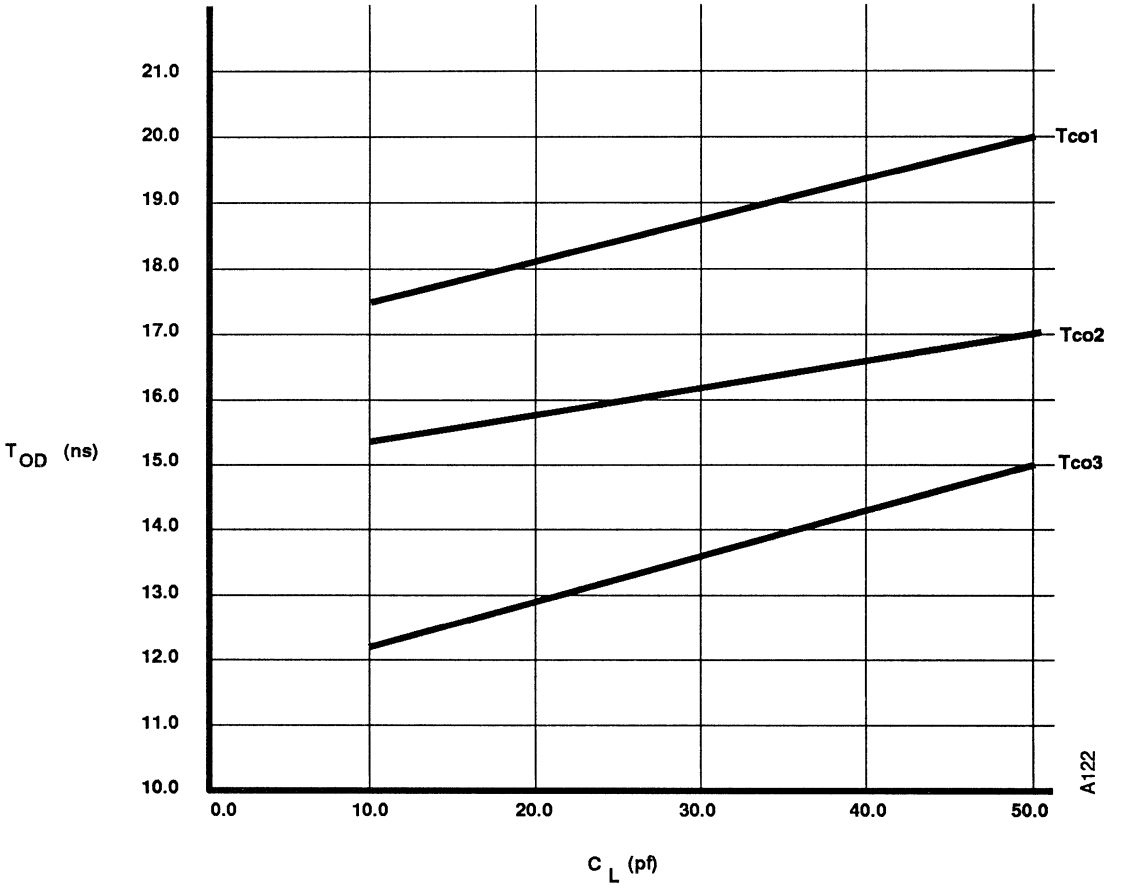
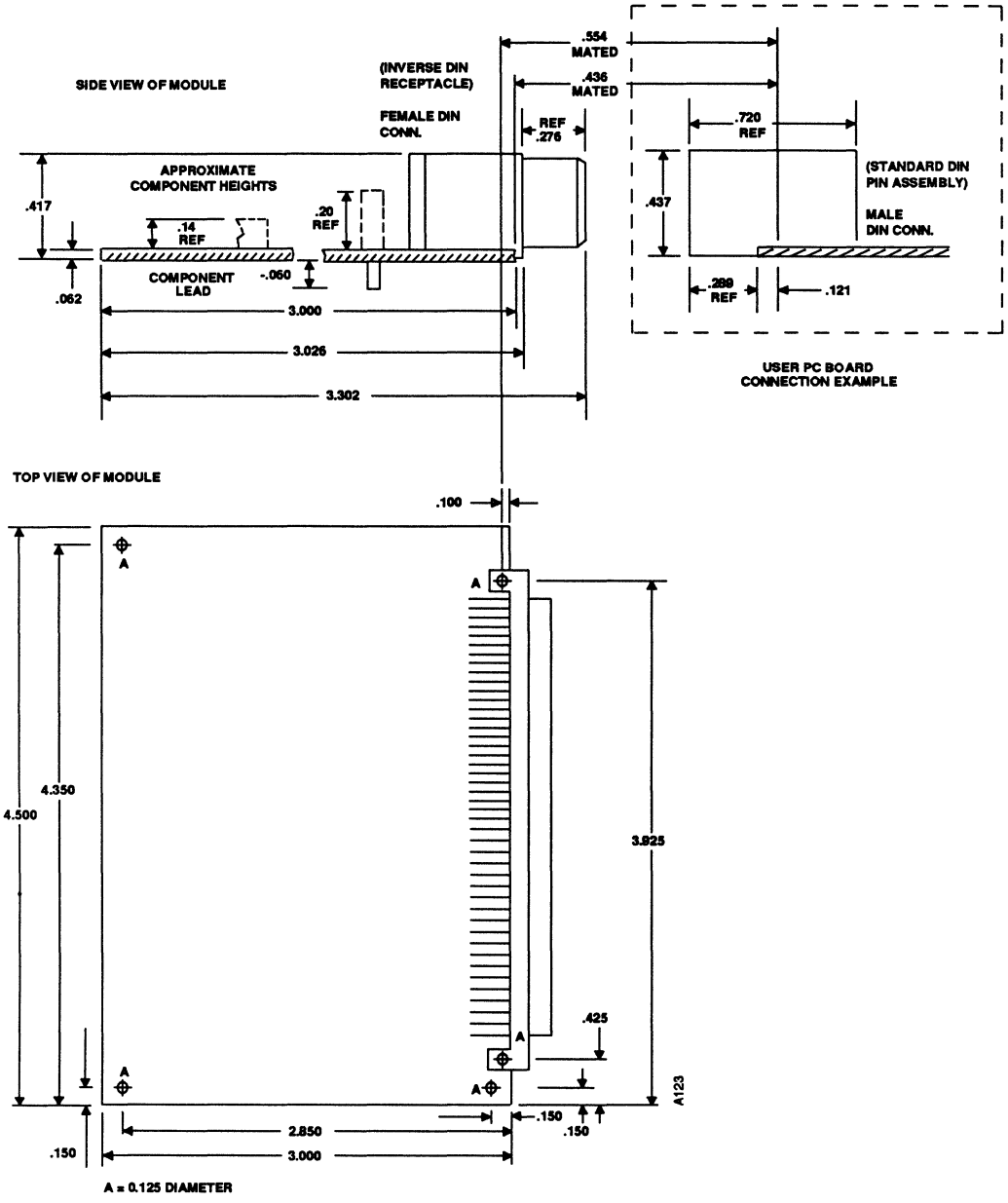
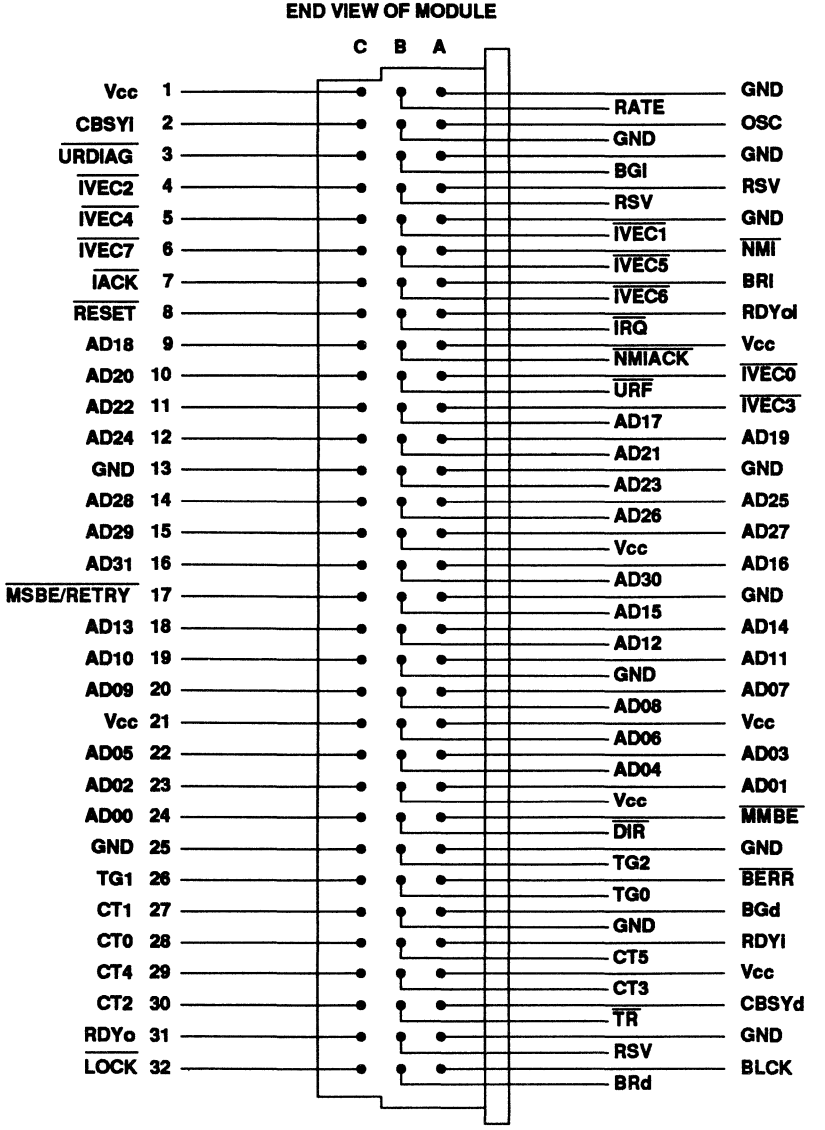


Figure 67 CLIPPER C100 Module C100C1MLX



NOTE:
Package dimensions are given in inches.

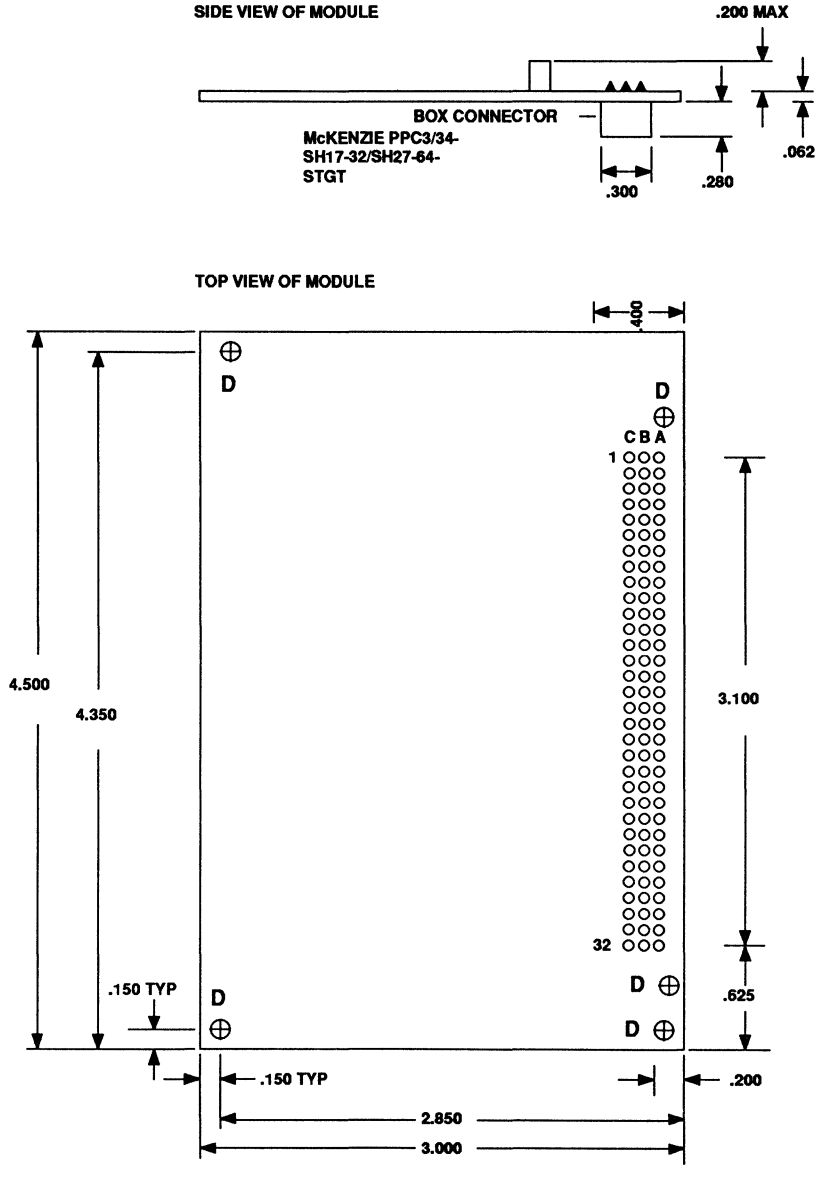
Figure 68 Pinout of CLIPPER C100 Module C100C1MLX



NOTES:
1. Numbering on the CLIPPER Module female connector may not correspond to numbering on user-supplied male connectors.
2. Pin B31 (RSV) should be tied to pin A32 (BCLK) to ensure compatibility with future enhanced versions of the CLIPPER Module.

A124

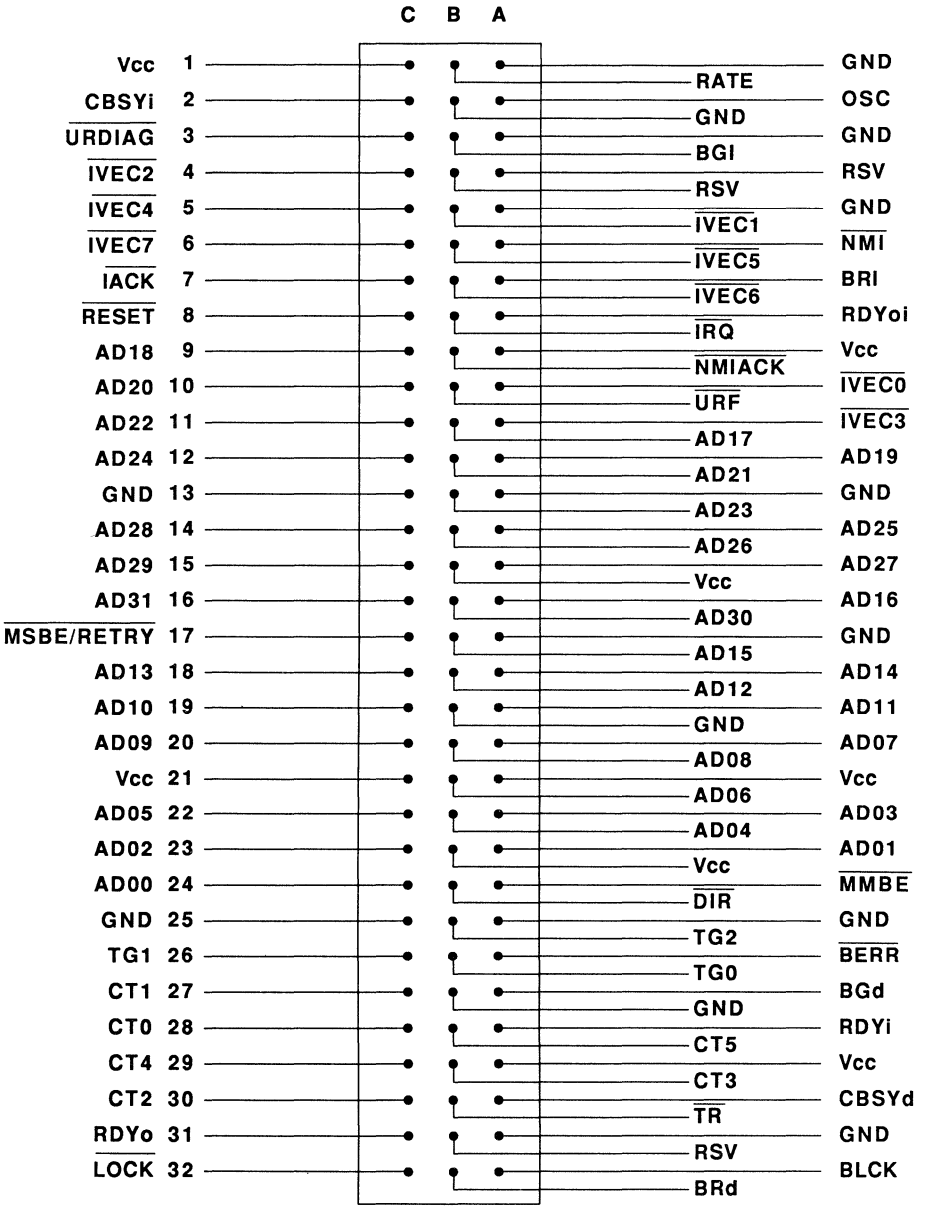
Figure 69 CLIPPER C100 Module C100C1BLX



NOTE:
Package dimensions are given in inches.

A125

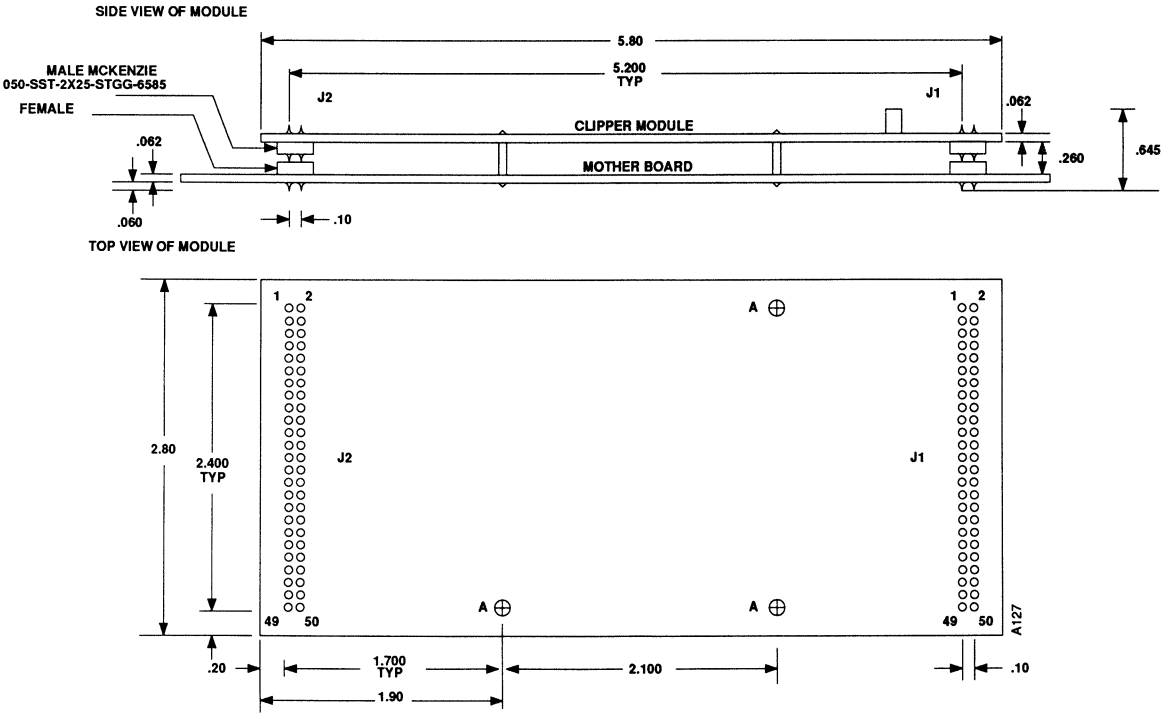
Figure 70 Pinout of CLIPPER C100 Module C100C1BLX



NOTE: Pin B31 (RSV) should be tied to pin A32 (BCLK) to ensure compatibility with future enhanced versions of the CLIPPER Module.

A126

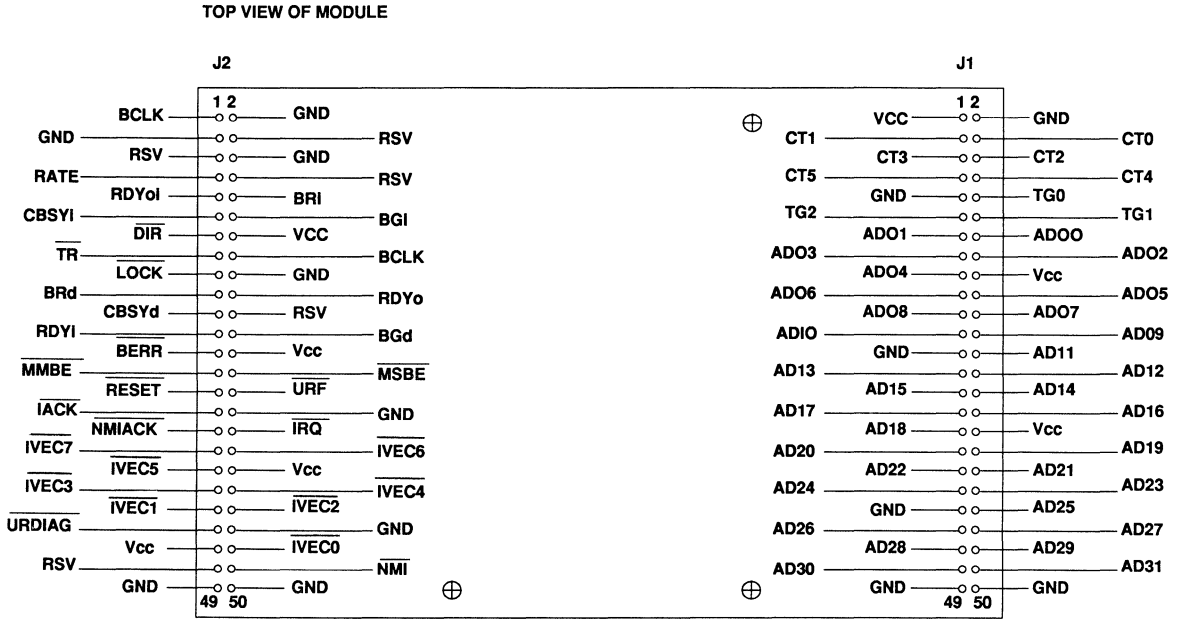
Figure 71 CLIPPER C100 Module C100C1DLX



A=.125 DIAMETER

NOTE:
Package dimensions are given in inches

Figure 72 Pinout of CLIPPER C100 Module C100C1DLX



NOTE: PIN J3-8 SHOULD BE TIED TO BCLK (PIN J2-1) TO ENSURE COMPATIBILITY WITH FUTURE ENHANCED VERSIONS OF THE CLIPPER MODULE.

A128

U.S. Offices

Region I – West Coast

Regional Office – Irvine
Irvine, CA (714) 863-9170
Los Angeles, CA (213) 479-3400
Phoenix, AZ (602) 263-0363
San Jose, CA (408) 287-2522
Seattle (Bellevue), WA (206) 455-9945

Region II – Western

Regional Office – Dallas
Dallas, TX (214) 669-9680
Englewood, CO (303) 220-9010
New Orleans (Metairie), LA (504) 837-8282
Tulsa, OK (918) 622-6891
Houston, TX (713) 978-7337
San Antonio, TX (512) 646-7711

Region III – Midwest

Regional Office – Detroit
Detroit (Farmington Hills), MI (313) 851-3520
Chicago (Schaumburg), IL (312) 885-0110
Indianapolis, IN (317) 845-7455
South Bend (Crown Point), IN (219) 662-1820
Lansing, MI (517) 485-5577
Minneapolis (Mendota Heights), MN (612) 681-1795
St. Louis (Ballwin), MO (314) 256-3200
Cleveland (Middleburg Heights), OH (216) 572-0056
Columbus, OH (614) 431-1168
Dayton, OH (513) 433-0195
Brookfield, WI (414) 355-1900

Region IV – Southeast

Regional Office – Atlanta
Atlanta, GA (404) 434-5598
Birmingham, AL (205) 822-1454
Orlando (Casselberry), FL (305) 260-5997
Tampa, FL (813) 885-8925
Lexington, KY (606) 223-4433
Cary, NC (919) 467-0021
Winston Salem, NC (919) 721-0068
Memphis, TN (901) 767-8641
Nashville, TN (615) 331-9603

Region V – Northeast

Regional Office – Washington D C
Washington, D.C. (Reston, VA) (703) 264-5600
Westborough, MA (617) 870-5970
Lyndhurst, NJ (201) 460-7421
Clinton, NY (315) 733-2199
Rochester, NY (716) 424-7230
King of Prussia, PA (215) 265-3562
Pittsburgh, PA (412) 391-6551

Federal Sales

Regional Office – Washington D C
Washington, D.C. (Reston, VA) (703) 264-5600
Huntsville, AL (205) 772-6862
Los Angeles, CA (213) 479-3400
Nashville, TN (615) 331-9603

Intergraph Corporation

One Madison Industrial Park
Huntsville, Alabama 35807-4201

Advanced Processor Division

Embarcadero Place
Building D
2400 Geng Road
Palo Alto, California 94303

Intergraph Europe

European Headquarters:

Intergraph Europe, Inc.
Hoofddorp, The Netherlands
Tel. (31) 2503-66333

Belgium: Intergraph (Benelux) BV
Brussels, Tel. (32) 2-217-5122

Denmark: Intergraph (Scandinavia) A/S
Frederiksberg, Tel. (45) 1-875888

Finland: Intergraph (Finland) Oy
Espoo, Tel. (358) 0-455-4744

France: Intergraph (France) S.A.R.L.
Rungis Cedex, Tel. (33) 1-45603000
Software Development Center (33) 1-46871562
Yvette, Tel. (33) 1-6907-7802

Italy: Intergraph Italia S.p.A.
Milanofiori, Tel. (39) 2-824-3043

The Netherlands: Intergraph (Benelux) BV
Aalsmeer, Tel. (31) 2977-21511

Norway: Intergraph (Norge) A/S
Asker, Tel. (47) 2-787980

Spain: Intergraph (España) S.A.
Barcelona, Tel. (34) 3-200-5299
Madrid, Tel. (34) 1-455-6446

Sweden: Intergraph (Scandinavia) AB
Taby, Tel. (46) 8-792-1150

Switzerland: Intergraph (Switzerland) AG
Zürich, Tel. (41) 1-3025202

United Kingdom: Intergraph (Great Britain) Ltd.
Swindon, Tel. (44) 793-619999
Derby, Tel. (44) 332-384815

West Germany: Intergraph (Deutschland) GmbH
Munich, Tel. (49) 89-461040
Düsseldorf, Tel. (49) 211-742076
Frankfurt, Tel. (49) 069-664001-0
Hamburg, Tel. (49) 40-630-4035

Other Intergraph International Operations

Intergraph Corporation

International Operations HQ036
One Madison Industrial Park
Huntsville, Alabama 35807-4201
(205) 772-2000/Ext. 2206

Australia: Intergraph Corporation Pty., Ltd.
Sydney (North Ryde), Tel. (61) 2-888-9900
Adelaide, Tel. (61) 8-239-0413
Brisbane, Tel. (61) 7-239-8905
Perth, Tel. (61) 9-368-2522
Melbourne, Tel. (61) 3-859-9421

Canada: Intergraph Systems, Ltd.
Calgary, Tel. (403) 250-6100
Edmonton, Tel. (403) 424-7431
Mississauga, Tel. (416) 625-2081
Montreal (Verdun), Tel. (514) 766-1292
Ottawa, Tel. (613) 230-8385
Vancouver (Burnaby), Tel. (604) 434-2677

Cyprus: Intergraph Middle East Ltd.
Larnaca, Tel. (357) 41-28700

Hong Kong: Intergraph Graphics Systems Hong Kong, Ltd.
Hong Kong, Tel. (852) 5-284012

Japan: Nihon Intergraph K.K.
Tokyo, Tel. (81) 03-576-1881

Korea: Intergraph Korea, Inc.
Seoul, Tel. (82) 2-784-8725

Mexico: Intergraph de Mexico, S.A. de C.V.
Mexico City, Tel. (905) 531-0862

New Zealand: Intergraph Corporation (N.Z.), Ltd.
Wellington, Tel. (64) 4-734005

Republic of Singapore:
Intergraph Systems South-East Asia (Pte), Ltd.
Singapore, Tel. (65) 733-9511

Taiwan: Intergraph Corporation Taiwan
Taipei, Tel. (886) 2-716-4458

Venezuela: Intergraph Servicios de Venezuela C.A.
Caracas, Tel. (58) 2-925552

For more information, call 1-800-826-3515 or 205-772-2700.
For sales information in countries not listed above, call 205-772-2206.

INTERGRAPH

Advanced Processor Division

Embarcadero Place
Building D
2400 Geng Road
Palo Alto, CA 94303

(415) 494-8800

DCLP1000