SNYS001

TI32000 Family Data Manual

1985

Microprocessors
Coprocessors
Support Chips

TEXAS INSTRUMENTS

TI32000 Family
Data Manual

TEXAS
INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes in the devices or the device specifications identified in this publication without notice. TI advises its customers to obtain the latest version of device specifications to verify, before placing orders, that the information being relied upon by the customer is current.

TI warrants performance of its semiconductor products, including SNJ and SMJ devices, to current specifications in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems such testing necessary to support this warranty. Unless mandated by government requirements, specific testing of all parameters of each device is not necessarily performed.

In the absence of written agreement to the contrary, TI assumes no liability for TI applications assistance, customer's product design, or infringement of patents or copyrights of third parties by or arising from use of semiconductor devices described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor devices might be or are used.

Specifications contained in this data book supersede all data for these products published by TI in the US before January 1985.

The information presented in this document is based in part on material copyrighted by National Semiconductor Corporation.

ISBN 0-89512-186-7

Contents

Contents (Continued)

Contents (Concluded)

List of Illustrations

List of Illustrations (Continued)

List of Illustrations (Continued)

List of Illustrations (Continued)

List of Illustrations (Concluded)

List of Tables

1 Introduction

Texas Instruments, the company that ignited the microchip revolution by inventing the silicon transistor, the integrated circuit, and the microprocessor, now drives the 32-bit minicomputer to the micro level with the TI32000™ family of Microprocessor Chip Sets.

The TI32000 family consists of the following components:

COMPONENT	BUS			DESCRIPTION†
	INTERNAL	DATA	ADDRESS	
TI32032T	32	32	24	Microprocessor (CPU)
TI32032D‡	32	32	32	Microprocessor (CPU)
TI32032DC‡	32	32	32	Microprocessor (CPU), CMOS
TI32016T	32	16	24	Microprocessor (CPU)
TI32008T‡	32	8	24	Microprocessor (CPU)
TI32081W	64	16	–	Floating Point Unit (FPU)
TI32081D‡	64	32	–	Floating Point Unit (FPU)
TI32082W	32	16	24	Memory Management Unit (MMU)
TI32082WA‡	32	16	24	Memory Management Unit, No Breakpoints
TI32082DC‡	32	32	32	Memory Management Unit (MMU), CMOS
TI32202W	16	8/16	–	Interrupt Control Unit (ICU)
TI32202B‡	8	8	–	Interrupt Control Unit (ICU)
TI32201	Not Applicable			Timing Control Unit (TCU), Bipolar

The TI32000 family is the solution for users wishing to standardize on a software transportable 32-bit, 16-bit, and 8-bit microprocessor family. Its elegant, symmetrical architecture makes it suitable for applications including powerful PCs, multiuser business computers, engineering workstations, super-mini computers and high-speed digital communications equipment.

™TI32000 is a trademark of Texas Instruments.
†Except where noted, all devices are implemented in NMOS technology.
‡Product is currently undergoing development.

1.1 Typical Applications of the TI32000 Family

COMPUTERS
- Personal Computers
- Multiuser Business Systems
- Graphics Display Terminals
- Large Plotters
- Copiers
- Transaction Systems

INDUSTRIAL
- CAD/CAE Systems
- Automatic Test Equipment
- Instrumentation
- Process Control
- Robotics
- Numerical Processing

COMMUNICATIONS
- PABX
- Central Office Switching
- Digital Transmission
- Networks

CONSUMER PRODUCTS
- Home Computers
- Automotive Controllers

MILITARY
- Weapons Systems
- Aircraft Controllers
- Land Vehicles

1.2 Mainframe Performance at a Microprocessor Price

The TI32000 family provides total hardware support for Demand-Paged Virtual Memory, high-speed floating-point operations, and High-Level Language (HLL) constructs. It has greatly extended the capabilities of even the latest generation of super minicomputers in its handling of HLL modules. Its highly symmetrical instruction set, comparable to the popular VAX™ architecture, makes it particularly well suited to powerful operating systems such as UNIX™. Furthermore when it comes to software productivity, any code written for the 32-bit TI32032 CPU will run just as well on the 16-bit TI32016 or 8-bit TI32008 CPU, and vice versa. Consider this absolute upward-to-downward and downward-to-upward object code compatibility in contrast to the upward-only compatibility of all other microprocessor families. This means programs written for your top-of-the-line 32-bit machines will also run on 8- and 16-bit systems, thus reducing repetitive software development, maintenance, and overhead costs. Further, each new product can rely on existing software and be much quicker to market.

1.3 Coprocessors Increase System Performance

Included in the TI32000 family of chip sets is the TI32082 Memory Management Unit (MMU) Coprocessor. The MMU implements Demand-Paged Virtual Memory management in systems where inexpensive secondary storage (e.g., a Winchester disk) is used to supplement physical memory (RAM) in support of large programs and data structures. The MMU incorporates two-level page indexing, as found in IBM mainframes, to avoid the many problems found in single-level indexing.

Another coprocessor, the TI32081 Floating Point Unit (FPU), accelerates floating-point calculations and appears as a software-transparent extension of the CPU. A user can also design his own "Application-Specific Coprocessor Unit" which communicates with the CPU in the same manner as the dedicated coprocessors. As coprocessors

VAX is a trademark of Digital Equipment Corporation.
UNIX is a trademark of AT&T Bell Laboratories.

appear as transparent extensions of the CPU to TI32000 programers, the decision to include or omit them in your end-products (for cost/performance reasons) will not affect software compatibility across your range of products.

1.4 System Support Chips

Support chips in the TI32000 family include the TI32202 Interrupt Control Units (ICU) and the TI32201 Timing Control Unit (TCU). A single TI32202 ICU provides 16 prioritized, vectored hardware interrupts; and when cascaded with other ICUs, up to 256 prioritized external interrupts are possible. (The TI32202B, a simplified version of the TI32202W, provides only 8 external interrupts, 64 cascaded.) The TI32201 TCU provides bus cycle timing, read-write control signals, various modes of bus cycle extension, and CPU timing.

1.5 TI32000: The 32-Bit Solution

By matching a TI32000 CPU with the appropriate combination of coprocessors and support chips, a system designer can accurately match cost and performance to his end product. Clearly, the TI32000 family of microprocessor chip-sets provides a total system solution for your design, be it 8, 16, or 32 bits.

1.6 Key Features of the TI32000 Family

Some of the features that set the TI32000 family apart as the best choice for 32-bit designs are as follows:

Family of Microprocessor Chip Sets

The TI32000 is more than just a single chip set, it is a family of chip sets. By mixing and matching TI32000 CPUs with compatible coprocessors and support chips, a system designer has an unprecedented degree of flexibility in matching cost and performance to the end product.

Cleanest 32-Bit Super Mini Computer Architecture

The TI32000 was designed around a 32-bit architecture from the beginning. It has a fully symmetrical instruction set so that all addressing modes and all data types can be operated on by all instructions. This makes it easy to learn the architecture; easy to program in assembly language; and easy to write code-efficient, high-level language compilers.

8, 16, 32, and 64-Bit Compatible Architecture

The TI32000 has an absolute upward and downward object code compatible architecture. This allows upgrading a product line while still preserving your entire software data base, thus reducing development costs and the risks involved in introducing new products. TI's commitment does not stop there. Even future 64-bit family members will be designed to maintain compatibility across the entire product range.

Demand-Paged Virtual Memory Management

The TI32000 provides hardware support for Demand-Paged Virtual Memory Management. This allows use of low-cost disk storage to increase the apparent size of main memory, and is an efficient method for managing very large address spaces. It is also the same popular memory management method used by DEC and IBM in their minicomputers and mainframes.

Application-Specific Coprocessors

The TI32000 architecture allows users to design their own application-specific coprocessors to interface with the existing chip set. These coprocessors can be used to increase your overall system performance by accelerating customized CPU instructions that you would otherwise implement in software. At the same time, software compatibility is maintained, i.e., it is always possible to substitute lower-cost software modules in place of the coprocessor.

Floating-Point Coprocessor

The TI32081 Floating-Point Coprocessors provide high-speed arithmetic computation with high precision and accuracy at low cost. They support the entire TI32000 family of CPUs and comply with the proposed IEEE standard for floating-point arithmetic, Task P754.

Operating System Support

TI32000 features such as hardware support for Demand-Paged Virtual memory management, user software protection and modular programming make it much easier to implement powerful, reliable and efficient operating systems. These features along with its symmetrical architecture and powerful instruction set make the TI32000 the most efficient and highest performance UNIX engine.

High-Level Language Support

The TI32000 has special features that support high-level languages, thus improving software productivity and reducing development costs. For example, there are special instructions that help the compiler deal with structured data types such as Arrays, Strings, Records, and Stacks. Also, modular programming is supported by special hardware registers, software instructions, an external addressing mode, and architecturally supported link tables.

2 Architecture Overview

2.1 Computer Architecture

2.1.1 Introduction

The architecture of a computer describes what that computer looks like to people who write software for it. More precisely, the architecture is the complete and detailed specification of the interface between the computer and software. The architecture specifies those elementary instructions that are decoded and executed directly by the machine. But it is important to keep in mind that architecture describes only *what* the computer does, not *how* it does it. Two machines are said to have the same architecture if all the software written for one can execute on the other, even if the actual hardware construction of the two machines is entirely different. For example, the members of the IBM System 360-370 family all have basically the same architecture, but the technology used to implement that architecture ranges from discrete transistors to Very Large Scale Integration.

Occasionally, the term architecture is used in a more general sense as the boundary between different levels of the whole system. (For example, terms such as "operating system architecture" are occasionally employed.) In this document we will use "architecture" exclusively for the boundary between the actual machine hardware and the software.

A computer architect is someone who designs computer architectures. The terms architecture and architect obviously have been adapted from their ordinary use in the building construction industry. The words are apt because in many ways the job of a computer architect is similar to that of an ordinary architect. Both are more concerned with the overall design of a structure and its appearance to users than with the exact details of the construction, which is the province of the structural engineer or general contractor in the building industry and the hardware designer in the computer industry.

The relationship between computer architect and computer implementor is analogous to the relationship between an architect and a general contractor. The architect designs the overall appearance of the building, balancing a number of conflicting goals (e.g., the desirable view provided by many large windows and the equally desirable goal of energy efficiency), always keeping in mind what is possible with current construction technology (the availability and cost of materials). The general contractor is responsible for translating the architect's vision into a building. If the contractor discovers that some detail of the building's architecture will be too difficult or too expensive to build, or that it will lead to an unsafe structure, the architect will have to make changes.

Similarly, the computer architect designs the external appearance (to software) of the computer, balancing a number of conflicting goals (e.g., complete protection vs simplicity of use), always keeping in mind the current state of semiconductor technology. The computer implementor translates this design into silicon. If the implementor finds that some feature of the computer architecture is too difficult or too expensive to implement, or if another feature causes the computer to run significantly slower, the computer architect may have to make changes.

The role of the architect in both industries is to make an intelligent compromise among a number of desirable goals and to balance this against the limitations of current technology to get a cost-effective design. Architectural mistakes usually result when one goal is single-mindedly pursued to the exclusion of other goals, or when a desired goal is simply not technologically feasible.

A certain amount of controversy currently surrounds a number of issues associated with computer architecture. As defined, computer architecture is the boundary between the hardware and software. The controversy is fundamentally over where that boundary should be drawn and what trade-offs should be made between various features for reasons of performance. Discussion has centered around three main topics:

- What is the best way to support high-level languages?
- How should memory be organized?
- What protection features should be provided by the hardware?

In the remaining sections of this chapter we will examine these three topics, introduce some of the points at issue, and present the TI32000™ approach to each topic.

2.1.2 High-Level Language Support

All evidence suggests that programming in a high-level language (e.g., Pascal) is more productive than programming in assembly language. Some researchers have found that high-level language programmers produce the same number of debugged lines of code per day as assembly language programmers. Since a line of code in a high-level language usually performs a more complex operation than a line of code in assembly language, the high-level programmer is more productive.

Studies have shown that both the time to debug a program and the difficulty in understanding and maintaining it are proportional to the number of instructions, with little dependency on the complexity of each instruction. Since several instructions might be required for each high-level language statement, the savings in programming time and cost over an equivalent assembly language program are obvious.

Before the advent of the TI32000, however, these advantages had been partially offset by the inherent inefficiency of high-level languages as opposed to assembly language programs. Depending on the compiler, the computer, and the application, a compiled program might be anywhere from 0% to 300% longer and slower than the best assembly language program. The basic reason for the inherent inefficiency

of high-level languages (HLLs) when they are targeted to contemporary architectures is that these architectures were not designed to support compilers.

2.1.2.1 Deficiencies of Current Architectures

The shortcomings of current computer architectures are largely attributable to what Glenford Myers has called the *semantic gap*,[1] a measure of the difference between the concepts in high-level languages and the concepts in the computer architecture. The objects and operations reflected in these architectures are seldom closely related to the objects and operations provided in the programming languages. This semantic gap contributes to software unreliability, performance problems, excessive program size, compiler complexity, and distortions of the language.

Here are some of the heavily used concepts in high-level languages, along with a few comments on the architectural support for these concepts provided by most computer architectures.

Arrays. The array is one of the most frequently used data structures in most HLLs. An array is a set of entries, each with the same data type (i.e., arrays of integers, arrays of characters, etc.). Most languages provide for multidimensional arrays, performing operations on entire arrays and checking to see that array subscripts do not exceed the boundary of the array. However, most computer architectures provide very limited architectural features to support any of these constructs.

Records. A record consists of a number of components (usually called fields) that may be of different data types. Thus a record might consist of characters, integers, and real numbers. Until now, there was nothing in the architecture of most microprocessors to support records.

Strings. Most languages contain the concepts of fixed and variable sized strings and of string processing operations such as concatenation and searching for a specified substring within a string. Many microprocessor architectures provide no string processing instructions at all.

Procedures. The basic program unit in modern HLLs is the procedure. A procedure call entails saving the state of the calling procedure, dynamically allocating and initializing local storage for the called procedure, passing arguments, and executing the called procedure. Most microprocessor architectures provide no support for any of these operations.

Modules. Modern HLLs (Pascal, Ada) implement the concept of a software module containing several procedures and associated data. Each module may be developed independently of all other modules and combined for final execution. This modularization reduces software development cost and time, increases design flexibility, and simplifies system design. To date, most processors have not supported the modular software concept.

One source of current problems is that contemporary architectures are *asymmetric*, and therefore do not permit the concepts in HLLs to be efficiently modeled in

1. Glenford J. Myers, *Advances in Computer Architecture*, Wiley, 1978.

machine language. Symmetry is the degree.to which all addressing modes exist for all operands and all required operators that exist for every data type.

Section 2.2 discusses symmetry in detail and also defines the key terms, such as "addressing mode" and "data type".

2.1.2.2 The TI32000 Approach

The deficiencies in contemporary microprocessor architectures have been addressed by the designers of the TI32000. They have made a major effort to bridge the semantic gap with this new architecture. The TI32000 architecture, in fact, is designed specifically to support high-level language compilers; it enables even relatively unsophisticated compilers to produce efficient code. Special addressing modes are provided to access such HLL constructions as arrays and records, and new operators that are specifically tailored for high-level languages.

Addressing Modes. The TI32000 architecture supports four standard *addressing modes* (i.e., mechanisms for accessing operands) common to most processors: register, immediate, absolute, and register relative. In addition, the TI32000 introduces four HLL-oriented addressing modes: Top-of-Stack mode is very useful for evaluating arithmetic expressions in high-level languages; Scaled Indexing mode can be used to access elements in byte, word, double word, or quad word arrays; Memory Relative mode can be used for manipulating fields in a record; and External mode can be used to access data in separately compiled modules. (See section 2.2 for a discussion of addressing modes.)

New Operators. In addition to the conventional CPU instructions, such as data movement, arithmetic logic, and shifts, the architecture includes advanced instructions which are very useful in an HLL environment. These advanced instructions are: the CHECK instruction which determines whether an array index is within bounds; the INDEX instruction which implements the recursive indexing step for multidimensional arrays; the STRING instruction which manipulates data strings; and the ENTER and EXIT instructions which minimize the overhead in procedure calls by managing the resources (registers, stack frame) allocated at the beginning of a procedure and reclaimed at the end. (See section 2.2 for more on these instructions.)

2.1.2.3 Controversial Topics

The addressing modes and new operators provided by the TI32000 clearly represent an advance over contemporary architectures. Yet two of the issues faced by the TI32000 designers remain controversial.

- Should three operand instructions be provided?
- Should instructions be primarily register oriented, memory-to-memory, or top-of-stack?

Three-Operand Instructions. It is occasionally claimed that an architecture must provide general three operand instructions if it truly is to support an HLL. (A three-operand instruction is, as the name implies, an instruction which contains two source operands as well as a destination. For example, an instruction to directly implement the FORTRAN statement,

$$A = B + C$$

would be a three-operand instruction with operands A, B, C and the operator +. The reasoning behind this claim is basically that if three-operand statements are common in high-level languages, then the presence of three-operand instructions in the architecture will result in greater code density. The VAX-11™, for example, permits three-operand instructions for most arithmetic operations.

However, a study by D.E. Knuth[2] of Stanford University in 1971 showed that in 250,000 lines of FORTRAN code, 80% of all statements were of the form

A *op* B or A = B

It follows that three-operand HLL statements are extremely rare and the need for such constructs in the architecture is unproven. Moreover, since provision for three-operand instructions imposes a certain burden of its own (whether in code density or execution speed), the utility of this instruction category must certainly be questioned. The designers of the TI32000 felt that the need for three operand instructions was not great enough to justify that overhead. In fact, the TI32000 provides greater code density than the VAX-11.

Registers. It is also occasionally claimed (for example by Glenford Myers in his book *Advances in Computer Architecture*[3]) that registers are alien to the concepts in HLLs and should be done away with in the interests of bridging the semantic gap. The designers of the TI32000 disagree. The high-level language concept that relates most strongly to registers is the idea of the set of variables that are local to a procedure. The modular programming methodology described above encourages the use of a number of small procedures instead of large monolithic programs. Each of these procedures usually makes use of only a few variables of its own, but these variables are used over and over again in that procedure. For instance, a procedure that manipulates an array must constantly refer to the array index.

The chief advantage of registers is that they allow a working set of variables to be kept close at hand where they can be accessed quickly.

This working set of variables is stored in the register set. Studies by William Wulf, or *et al.*[4] have indicated that five registers are sufficient for almost all applications. TI32000 CPU uses 8 (i.e., 2^3) address-data registers and several specialized registers for particular pointers. The TI32000 architecture *allows* memory-to-memory operations, but it does not *require* them.

Registers allow the compiler writer to optimize the execution of HLL statements, whereas a purely memory-to-memory machine must constantly carry the overhead of referencing all variables in main memory. A pure stack-oriented machine (i.e., an architecture where all variables are assumed to be on the top two locations of the

VAX-11 is a trademark of Digital Equipment Corporation.

2. D. E. Knuth, "An Empirical Study of FORTRAN Programs," *Software Practice and Experience*, 1, 2 (April-June, 1971) 105-133.
3. Meyers, *op. cit.*, p. 23
4. W. A. Wulf, *et al., The Design of an Optimizing Compiler*, North Holland, 1975

stack) is essentially equivalent to a machine with two registers. Many studies have shown that pure stack machines do not give any significant advantage over a general register machine.[5]

2.1.3 Memory Organization

There are three aspects to memory organization: (1) the overall memory architecture, which is basically how the logical memory looks to the computer program; (2) logical-to-physical address translation (mapping), which maps the logical structure of memory onto hardware; and (3) virtual memory mechanisms. The TI32000 has a *linear* memory architecture; it supports *page-based* mapping; and it provides a number of mechanisms which support a virtual memory system.

2.1.3.1 Linear vs Segmented Memory Architecture

The main memory of a computer is organized as a set of consecutively numbered storage cells. In most computers these memory cells contain eight bits (a *byte*). The location number associated with one of these physical storage cells is called a *physical address*, and the set of all physical addresses is called *physical address space*.

The physical address space is thus determined by the actual hardware in the computer's memory system.

On the other hand, a program running on a computer can generate a set of addresses that is limited only by the number of bits in its address registers. This set of addresses is not necessarily related to the actual amount of physical memory in the system. For example, consider a computer with a 16-bit address field in instructions and 4,096 (4K) bytes of memory. A program on this computer can address 65,536 (64K) locations, for the simple reason that 2^{16} (65,536) 16-bit numbers exist. The set of these numbers is called *logical address space*; it is the set of *logically possible addresses* (even if they are not realized physically); it is the set of all addresses that can be generated by a program. The organization of the logical address space defines the *memory architecture*. The two main types of memory architecture are *linear* and *segmented*.

In a linear address space, addresses start at location zero and proceed in a linear fashion (i.e., with no holes or breaks) to the upper limit imposed by the total number of bits in a logical address. In TI32000 systems there can be up to 32 bits in a logical address, resulting in over 4 billion (2^{32}) bytes.

The alternative to a linear memory architecture is a *segmented* memory architecture. A segmented address space is basically a collection of small linear address spaces. A rigid distinction is made between the *segment* (the particular address space in which a datum is located) and the *displacement* of the datum within the segment (the distance in bytes from the start of the segment to the location in question). A segmented address is consequently a two-component value. The first component (the segment selector) picks out a particular segment while the second component specifies the displacement within the segment. (See Figure 1 for a comparison of linear and segmented memory.)

5. Meyers, *op. cit.*, p. 49

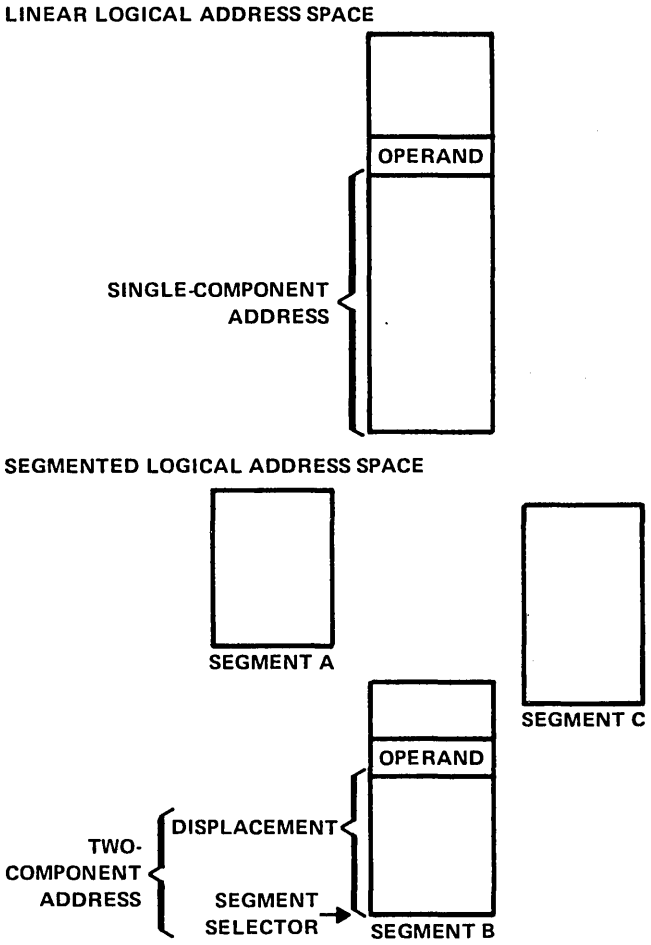


Figure 1. Linear vs Segmented Address Space

The advantages of segmented memory center around protection issues. The claim is made that a segmented memory better suits the organization of modern, modular programs and structured data than does a linear memory. Consequently, mechanisms for preventing access to segments, or preventing segments from being read or written into can be used to protect meaningful program units. In other words, since the logical address space of a segmented architecture reflects the logical structure of the program, protection mechanisms provided for segments naturally accrue to meaningful program units.

This is in fact true. However, except for a few processors (e.g., the MULTICS processor), few segmented machines have consistently carried out this program. For example, most current segmented architectures impose a limit of 64K bytes on the length of a segment. But in order for segmentation to realize its protection

advantages, segments should be allowed to have arbitrary size. A 2M byte segment, after all, will be needed to hold a 2M byte array, if the program organization is to reflect the program structure. Also, in modern bit-mapped graphics systems (a typical application for 16-bit microcomputers), 2M byte arrays are common. Moreover, since programs can consist of hundreds or even thousands of modules, it is important for the architecture to support large numbers of segments if segmentation is to be used properly.

Large data bases are a typical application that will require either segments of arbitrary size or a great many segments.

Unfortunately, most segmented architectures allow only small segments (i.e., less than 64K bytes) and usually support only a limited number of them (typically, fewer than 128). The size limitation is an artifact of earlier days when the entire (linear) address space was only 64K bytes long. The designers of segmented machines expanded the address space of their earlier processors, while attempting to preserve some measure of software compatibility by making the old 64K-byte linear address space one of the new 64K bytes segments. The 8086 and its relationship to the 8080 is the most painful illustration of this phenomenon.

In such segmented architectures, all data structures larger than the maximum segment size must be broken down to fit into several segments, since an address pointer cannot be incremented from the top of one segment to the bottom of another segment. By contrast, a linear address space can accommodate data structures of any size up to the maximum size of memory.

The TI32000 provides the protection advantages of segmentation without the segment size disadvantages, by permitting segments to be constructed out of an arbitrary number of fixed-size memory units. These memory units are called *pages*, and they form the basis for the TI32000 mapping, virtual memory, and memory protection mechanisms.

The TI32000 permits a form of segmentation, that is, it lets the operating system keep track of collections of pages with the same protection attributes, but it does not *require* segmentation by building it into the architecture. Moreover, the segmentation permitted by the TI32000 is more general than that built into standard segmented architectures (for example, segments can have arbitrary size).

2.1.3.2 Page-Based Mapping and Alternatives

Mapping is based on the distinction between logical address space and physical address space. Basically, mapping is the process of translating a logical address into an arbitrary physical address. Without mapping, logical addresses are simply equated with physical addresses; by exploiting mapping, a logical address can be assigned to an arbitrary physical address. Mapping thus provides a kind of generalized relocation mechanism.

Unmapped memory is adequate for simple, single-user, single-task systems, which is why most microcomputer applications until now have been unmapped. However, the large memory and increased power of 16-bit microcomputers have led to their being employed in multiuser, multitasking applications. And in these cases mapping

is highly desirable. Because, without mapping, the different programs in a multiprogramming system or the different tasks in a multitasking system must operate within the same logical address space. Consequently, each program or task must be careful not to access any address outside its assigned partition, and in general everyone must be familiar with the detailed organization of memory in order to make full use of it.

By contrast, mapping allows each program or task to be assigned its own logical address space, with the mapping mechanism responsible for translating these independent logical address spaces into the same physical address space. Since the programs and tasks have separate logical address spaces, there is no chance of interference.

Since it is too cumbersome to control the translation of each logical address individually, mapping is ordinarily done in blocks of addresses. The simplest and historically the earliest mapping systems mapped the entire logical address space of a program as one unit. (See Figure 2 for a diagram of such a system.)

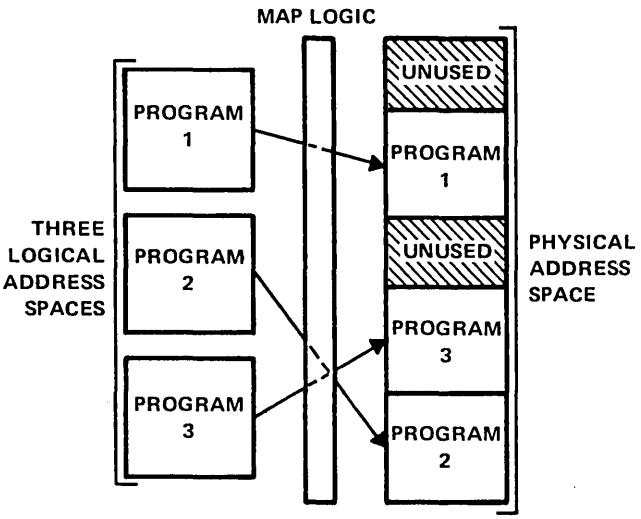


Figure 2. Mapping the Entire Address Space

More recent systems are based on mapping smaller chunks of memory, rather than the entire logical address space of a program. Basically, there are two kinds of address translation schemes, differing only in the structure of the mapping blocks: One based on variable sized segments; and the other based on fixed-size units called "pages." The TI32000 employs a page-based mapping system.

In TI32000 systems with 24 address bits, the logical address space is broken up into 32,768 pages, each with a fixed size of 512 bytes. The physical address space is broken up into the same number of pieces, each the same size as a page. These pieces of physical memory into which the pages are mapped are called page frames. Figure 3 shows a part of the TI32000 mapping scheme.

A page-based mapping system is usually more efficient than a segment-based mapping system because of the memory fragmentation problem associated with segment-based systems. This problem occurs often in segmented multiprogram systems when the available memory space becomes fragmented into many small pieces and not enough contiguous physical memory is available to contain one large segment. By contrast, since all pages are the same size, if any physical page frame is available, it can hold any page.

The mapping operation is performed by the TI32082 Memory Management Unit (MMU) and is explained thoroughly in section 2.3.2. This translation process is performed automatically, making use of a table in memory that contains the physical addresses of each page frame.

Each program or task can have its own set of translation tables, and changing the selected group of tables is simply a matter of updating an MMU register that points to the starting address of the top-level page table. Therefore, each program or task can have its own map from logical memory to physical memory, and each program or task can have its own logical address space.

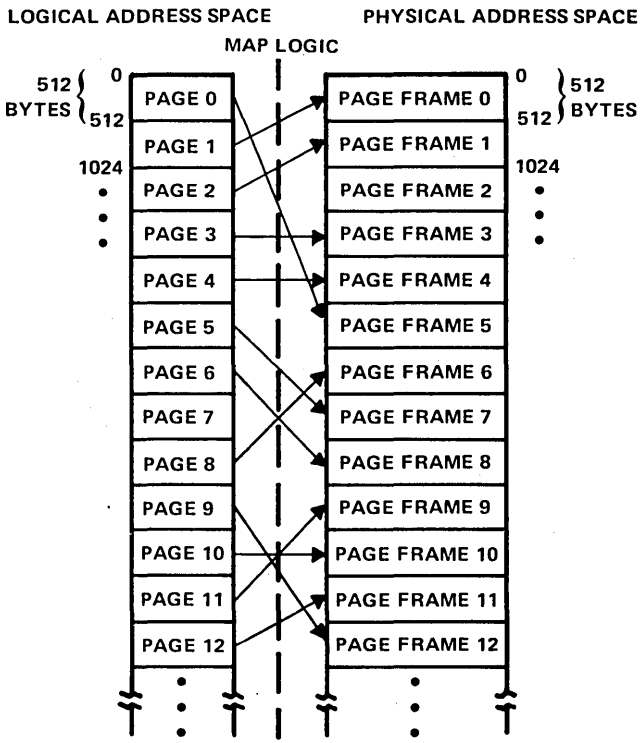


Figure 3. Page-Based Mapping

Entries in the translation tables contain protection bits along with physical addresses. These protection bits are used to provide each page with a set of protection attributes (e.g., read only). The Operating System can treat a collection of pages with the same attributes as a segment. Thus, page-based mapping provides a mechanism for implementing segmentation.

2.1.3.3 Virtual Memory

In many computer systems, the logical address space is far larger than the actual memory hardware. *Virtual memory* is a mechanism for circumventing the limits on physical memory size. Under a virtual memory system, it appears to users as if the entire logical address space is available for storage. But, in fact, at any given moment only a few pages of the logical address space are mapped into physical space. The other pages are not present in main memory at all; instead, the information in these pages is stored on a secondary storage device, such as a disk, whose cost-per-bit is more economical.

In a virtual memory system, whenever the computer generates a memory address, the hardware checks whether that address lies in a page that is actually in memory. If it does, the address is translated to the appropriate physical address, and the memory reference takes place normally. If the indicated page is not in memory, an operation called a *page swap* is performed, and the operating system software loads the missing page from disk. If this operation is performed swiftly, the user will have the illusion of a gigantic physical memory. For efficiency, when the referenced location has to be brought from the peripheral to the main memory, other locations likely to be referenced next may be brought in. Information not currently in use is removed from the main memory and returned to peripheral storage, thus making room for the new material.

Of course the beauty of virtual memory is that the user or programmer does not have to be aware of the process. He uses one consistent set of addresses called *logical addresses*. The memory management hardware keeps track of where the information resides at any given time and translates the logical address into a real location in physical memory. When the CPU finds the requested logical address to be unavailable in main memory, it notifies the operating system which initiates a swap.

When the data to be replaced has not been modified during the time it was resident in main memory, there is no need to write it back to the peripheral device since an up-to-date copy already exists there. Under such a circumstance, the old data is overwritten with the new data.

Virtual memory was first implemented on the Atlas computer at Manchester University, using special hardware. All computers with virtual memory since the Atlas have also required special hardware functions to implement virtual memory. Current microprocessors do not have adequate mechanisms to support virtual memory systems. For example, in both the Z8000 and the 68000 no provision is made for restarting an instruction that causes a page fault. In TI32000 virtual memory systems, this special hardware is provided by the TI32082 MMU (with some support from the CPU chip).

2.1.4 Protection

The last major area of debate about computer architecture concerns the whole topic of protection: memory protection, program protection, and user protection. The basic issue is what should be the granularity of the protection mechanisms that are provided. The basic difficulty is that the finer the granularity, the more the overhead associated with protection.

Some systems implement a hierarchy of protection levels from most privileged to least privileged. These levels are often called *rings*. Each ring has its own access control information for a page. Generally, a more privileged ring has access to all the information in a less privileged ring. However, the number of rings is severely limited, usually to four, and tasks often do not have a strictly hierarchical relationship; therefore, ring systems are seldom flexible enough for modern operating systems.

Instead, a capability-based protection system is often proposed as an alternative which allows non-hierarchical relationships between an arbitrary number of tasks. In a capability-based operating system, each task has a table of operations it is allowed to perform that may affect other tasks in the system. This table is protected from direct modification by the task. Thus, the only way a task can perform an operation which could affect another task is if it has the appropriate capability in its capability table. A task may give a specific capability to another task. By restricting the distribution and type of capabilities it gives out, a task may tightly control access to the services it provides.

One problem with most capability-based systems is that the concept is carried to such lengths that it interferes with efficient accessing and processing of information within a task. Since the cost of protection is always high in these capability systems, performance suffers.

The designers of the TI32000 felt that a capability-based protection scheme could be implemented at some level in the system, but that the appropriate level to do this was in the kernel of the operating system, not in the architecture itself. The basic reason for leaving capabilities out of the architecture is twofold: (1) the extra burden should not be imposed on all programmers who use this architecture or on every memory reference; (2) the implementation of a capability-based system is such a new and complex task that locking such a system into silicon before it is thoroughly proven can be very risky. The designers of the TI32000 preferred to work out the bugs in their operating system before they froze it permanently in silicon.

The protection features actually implemented in the TI32000 architecture can be divided into three groups:

1. Operating System/User mode. A distinction is made between two operating modes of the CPU: *Operating System mode* in which all the power of the instruction set is available and *User mode* in which only a restricted subset of the instructions are available. Operating System mode is intended for operating systems and other trusted programs. User mode is intended for those programs that are not trusted.

2. Separate address spaces for each task. Each task running on the TI32000 has its own collection of pages constituting its address space. Access to another task's address space is impossible.
3. Protection bits in the page and pointer table entries. Associated with each page are bits that define whether that page can be read but not written into, read and written into, or neither read nor written into. (See section 2.3.4.)

2.2 High-Level Language Support on the TI32000

2.2.1 Introduction

In the previous section it was shown that with conventional architectures the gain in programming efficiency produced by writing in high-level languages instead of assembly language is usually undermined by the larger amount of memory required to store the code. This phenomenon is a result of the large number of instructions that must be generated by the compiler to map HLL concepts onto the more restricted repertoire of machine instructions. Performance is also diminished because of the large number of memory transactions generated by the instructions. In addition, when the differences between the abstractions called for by a problem and the capabilities directly implemented in the computer's hardware is very great, the code generation portion of a compiler must be extremely complex.

A primary design objective for the TI32000 is for the structure and behavior of the processor's architecture to correspond in a reasonable way with the objects and operations of high-level languages. The goal was to develop a symmetrical architecture particularly suited to being the target for compilers. The architecture of the TI32000 meets that goal; it enables symmetric use of address-data registers, memory locations, addressing modes, data types, and instructions.

Compilers can easily generate high-performance (very dense and efficient) code for the TI32000. The TI32000 is particularly well suited to the Pascal high-level language. Because of the TI32000's Address-Data registers, the program also executes faster. In addition, the architecture avoids special-case instructions and addressing modes that compilers have difficulty using.

In this section we will examine in detail the means by which the concepts of HLLs are supported by the TI32000 architecture; namely, by a symmetrical architecture, a sophisticated instruction set, and expanded addressing capabilities.

2.2.2 Data Types Supported

The objects and concepts of a high-level language include constants, variables, expressions, and functions. Each of these has a particular *data type* which determines the range of values that the constant, variable, expression, or function may assume in the program.

A data type is said to be *supported* by a computer if the computer's instruction set contains operators that directly manipulate the data type, or has operators and addressing modes that facilitate its manipulation. Data types directly manipulated by the hardware are called *primitive data types.* Those data types supported by the hardware, but not manipulated directly, consist of ordered collections of primitive types and are called *structured data types.*

The TI32000 supports the following data types:

- Primitive Data Types (see Figure 4)
  - Integers (signed and unsigned)
  - Floating-Point
  - Booleans
  - Binary Coded Decimal (BCD) digits
  - Bit Fields
- Structured Data Types
  - Arrays
  - Records
  - Strings
  - Stacks

2.2.2.1 Integer Data Types

The integer data type is used to represent *integers*, i.e., whole numbers without fractional parts. Integers may be signed (negative as well as positive) or unsigned (positive only). Integer data types on the TI32000 are available in three sizes: 8-bit (byte), 16-bit (word) and 32-bit (double word). Signed integers are represented as binary two's complement numbers and have values in the range -2^7 to 2^7-1, -2^{15} to $2^{15}-1$ or -2^{31} to $2^{31}-1$; unsigned integers have values in the range 0 to 2^8-1, 0 to $2^{16}-1$, 0 to $2^{32}-1$. When integers are stored in memory, the least significant byte is stored at the lowest address; the most significant byte at the highest address.

2.2.2.2 Floating-Point Data Types

The *floating-point* data type is used to represent real numbers, i.e., numbers with fractional parts. Floating-point numbers are represented by an encoded version of the familiar scientific notation:

$$n = s \times f \times 10^e$$

where s is the *sign* of the number, f is called the *fraction*, or *mantissa*, and e is a positive or negative integer called the *exponent*. (Figure 4 shows how these values are represented by fields within the number.) Floating-point numbers are available in two sizes: 32-bit (single-precision) and 64-bit (double-precision). Double-precision offers both a larger range (larger exponent) and more precision (larger mantissa). The TI32000 floating-point data type is compatible with the proposed IEEE floating-point standard (Task P754).

Manipulation of the floating-point data type is actually handled by the TI32081 Floating-Point Processor (FPU) (see section 2.4.4). If an FPU exists in the system, the user can treat floating-point numbers (both single- and double-precision) as any other TI32000 data types and may use any of the TI32000 addressing modes to reference them. Also, conversion is provided from every integer and floating format to every other integer and floating format. If an FPU is not present, these functions must be simulated in software.

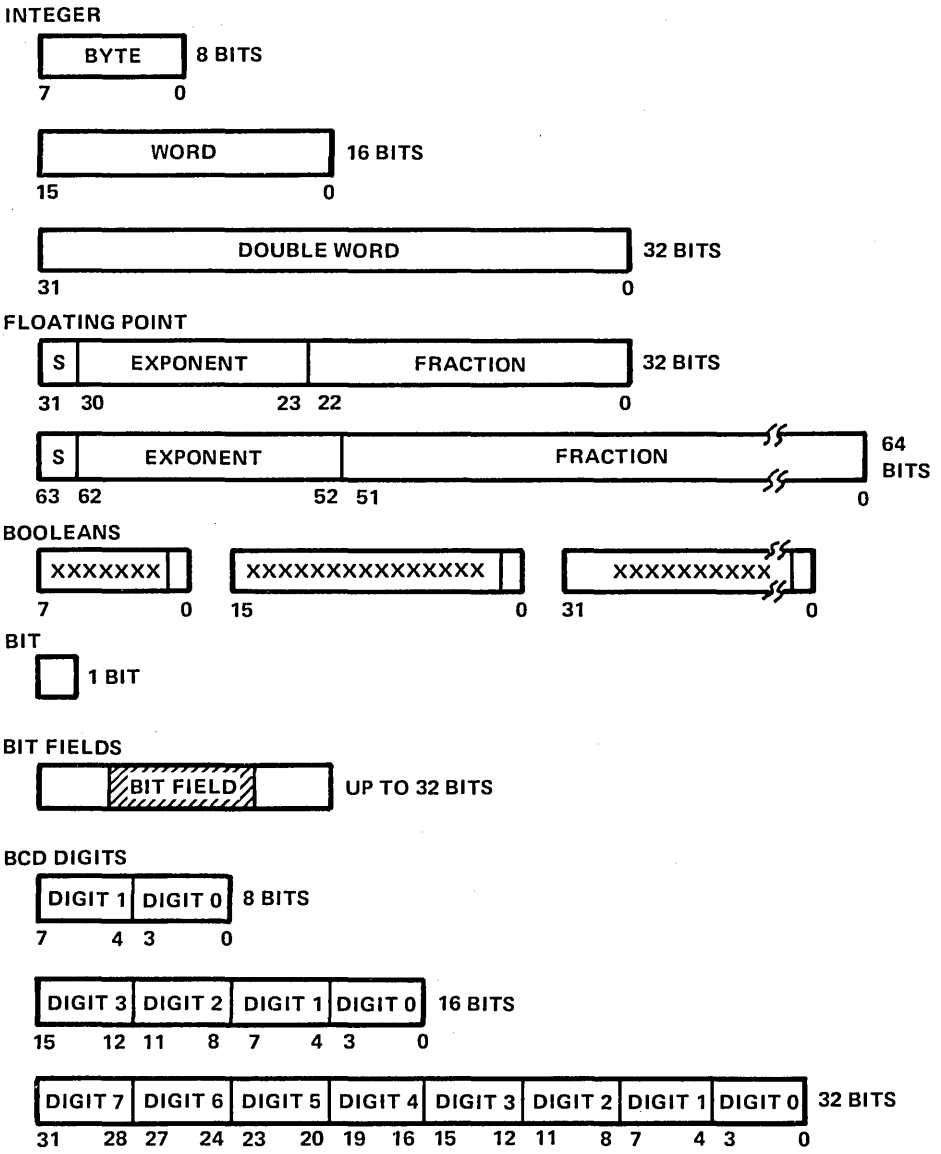


Figure 4. Primitive Data Types

2.2.2.3 Other Primitive Data Types

The *Boolean* (or logical) data type is a single bit whose value, 1 or 0, represents the two logic values *true* and *false*. A Boolean data type has many uses in a program, for example, to save the results of comparisons, to mark special cases, and in general to distinguish between two possible outcomes or conditions. Booleans are represented on the TI32000 by integers (byte, word, or double word). True is integer 1; false is integer 0.

The *bit field* data type is different from other primitive data types in that the basic addressable unit is measured in bits instead of bytes. On the TI32000, bit fields may be 1 to 32 bits long, and located arbitrarily with respect to the beginning of a byte. They are useful when a data structure includes elements of nonstandard lengths, since they allow programs to manipulate fields smaller than a byte.

With the *binary-coded decimal* (BCD) data type, unsigned decimal integers can be stored in the computer, using 4 bits for each decimal digit. The BCD data type is represented on the TI32000 by three formats, consisting of 2, 4, or 8 digits. Two BCD digits may be packed into a byte, four to a word, or eight to a double word. Thus one byte may represent the values from 0 to 99, as opposed to 0 to 225 for a normal unsigned 8-bit number. Similarly, a word can represent values in the range 0 to 9,999, or a double word can represent values in the range 0 to 99,999,999.

Although BCD requires more bits to represent a large decimal number, it does have certain advantages over binary. For many business applications, the amount of actual computing to be done between source input and output is small, so that converting data from binary to decimal formats can represent a significant portion of the total processing overhead. BCD arithmetic eliminates this conversion overhead since the computations are actually performed in decimal. Also of importance to business applications is the loss of accuracy which can result from conversions from decimal to binary and back again, a loss which is avoided by using decimal arithmetic.

Arrays. An *array* is a structured data type consisting of a number of components, all of the same data type, such that each data element can be individually identified by an integer index. Arrays represent a basic storage mode for all high-level languages.

In Pascal programs, for example, each element of an array is referenced by the array name and an index value giving the component's position in the array. Arrays range from simple one-dimensional vector arrays to more complex multidimensional arrays. The elements of an array may be integers, floating-point numbers, Booleans, characters, or more complex objects built up from these types.

The TI32000 provides special operators that facilitate calculation of the array index and determination if the index is outside the limits of the array. In addition, certain TI32000 addressing modes facilitate quick access to array elements. (See section 2.2.5.2.)

Records. A *record*, like an array, is a structured data type with several components. However, unlike arrays, the components of a record may each be of a different data type. In high-level languages, such as Pascal, a component of a record is selected by using both the name of the record variable and the name of the component. Usually, records are grouped into large arrays, called *files* in COBOL, *structures* in PL/1, and *record structures* in Pascal.

The TI32000 addressing modes facilitate quick access to record elements. (See section 2.2.5.2.)

Strings. A *string* is an array of integers, all of the same length. The integers may be bytes, words, or double words. Strings are common data structures in high-level languages. For example, strings of ASCII characters (i.e., bytes) are commonly used to contain alphanumeric text.

On the TI32000, a string is represented by a sequence of integers stored in contiguous memory. Special instructions exist that facilitate comparison of strings, movement of strings, and searching strings for particular integer values. (See section 2.2.3.4.)

Stacks. A *stack* is a one-dimensional data structure in which values are entered and removed one item at a time at one end, called the *top-of-stack*. It consists of a block of memory and a variable called the *stack pointer*.

Stacks are important data structures in both systems and applications programming. They are used to store return address and status information during subroutine calls and interrupt servicing. Also, algorithms for expression evaluation in compilers and interpreters depend on stacks to store intermediate results. Block-structured HLLs such as Pascal keep local data and other information on a stack. Parameters of a procedure in a block-structured HLL are usually passed on a stack, and assembly language programs sometimes use this convention as well.

The TI32000 supports both a User Stack and an Interrupt Stack. Depending on the mode of operation, one of two stack pointers (SP0 or SP1) contains the memory address of the top item on the stack. Instructions exist which allow for explicit manipulation of the stack pointer, and the current stack can be used in almost all TI32000 instructions to hold an operand.

For example, an item may be pushed onto the stack by subtracting the length of the item from the stack pointer (since stacks, by convention, grow downward in memory) then moving the item to the address now pointed to by the stack pointer. An item may be popped off the stack by moving the item pointed to by the stack pointer to the destination then adding the length of the item to the stack pointer. Both of these operations are performed by selecting the Top-of-Stack Addressing mode.

Instructions also exist which push or pop the contents of one or more registers. For example, the Jump to Subroutine instruction causes the Program Counter's contents to be pushed on the stack, and the Enter instruction causes the contents of the Frame Pointer and specified address-data registers to be pushed on the stack. (See section 2.2.3.6.)

2.2.3 Instruction Set

One of the most important considerations in evaluating a computer architecture is the relationship between the machine's primitive data types and the instructions that manipulate those data types. For example, if a processor has byte, word, and double word integers, it should have an Add instruction that operates on each of these data types in a uniform and consistent manner. In the TI32000 architecture, a complete and comprehensive set of instructions is available for every hardware recognized primitive data type. In addition, special instructions are available that facilitate manipulation of structured data types.

The instruction set includes over 100 basic instruction types, chosen on the basis of a study of the use and frequency of specific instructions in various applications; special case instructions, which compilers cannot use, have been avoided. The instruction set is further expanded through the use of special coprocessors, acting as extensions to the CPU.

This instruction set is symmetrical; that is, instructions can be used with any general addressing mode, any operand length (byte, word, and double word), and can make use of any address-data register.

The TI32000 instructions are genuine two operand instructions, although many instructions use more (up to five) operands. This, combined with the consistent and symmetric architecture, reduces the code size considerably.

2.2.3.1 Integer Instructions

A large set of arithmetic instructions are provided for integer manipulation: addition and subtraction, multiplication and division (with various remainder, rounding, modulus and result-length options), two's complement, and absolute value. Other instructions include:

- Move instructions that allow either zero or sign extension (a useful feature when the size of the destination exceeds the size of the source).
- Shift instructions allowing logical and arithmetic shifts, as well as rotation left or right, both by any amount.
- Boolean instructions (AND, OR, Exclusive OR, Complement, and Bit Clear) allowing each bit in a data word to be manipulated independently.
- Two BCD arithmetic instructions, Add and Subtract, handling up to eight digits at a time.
- Extended Multiply and Divide instructions which return a result which is twice the size of the operands which they read.

2.2.3.2 Floating-Point (FPU) Instructions

The TI32000 supports 32-bit and 64-bit precision floating-point calculations, as well as 8-, 16-, and 32-bit fixed-point calculations. In addition to the floating Add, Subtract, Multiply, Divide, and Compare instructions, there is a Move instruction that doubles as a conversion instruction for converting from integer to floating-point format. Instructions are also provided to Round off a floating-point number toward zero, and to convert a floating-point number to the largest integer less than or equal to itself (the Floor of that number). For positive floating-point numbers these last two operations have the same effect; they differ, however, for negative numbers. For example, -3.17 truncates to -3, but its Floor is -4.

These instructions are implemented by the FPU and display the same symmetry, addressing modes, and flexibility as the rest of the instruction set. The architecture of the TI32000 makes available to the FPU all the TI32000 addressing modes, and any instructions can be register-to-register, memory-to-register, or memory-to-memory.

2.2.3.3. Boolean, Bit, and Bit Field Instructions

Boolean instructions treat a data word as an array of bits and allow each bit to be handled independently. Boolean operators include AND, OR, Exclusive OR, Complement, and Bit Clear.

The TI32000 family provides a special *Boolean* Not instruction for implementing high-level languages which require that TRUE = 1 and FALSE = 0. To simplify the handling of Boolean expressions in compilers, a Set-on-Condition instruction stores a '1' into its only operand if a condition code check is satisfied; if not, it stores a '0'.

Bit instructions allow convenient handling of individual bits or arbitrarily large bit arrays. In addition to the ability to set, clear, complement or test any bit in memory or in a register, the TI32000 family provides semaphore primitives (test and set, test and clear) for multiprocessing and multitasking coordination. Also provided is a Convert to Bit-Field Pointer instruction which converts a byte address and a bit offset into a bit address. This allows a field address to be converted to an integer and thus passed to a procedure or function, which is very useful in HLLs. A Find First Set instruction searches a sequence of bits, either in memory or in a register, and returns the bit number of the first '1' bit it sees.

Two *Bit Field* instructions can access bit fields up to 32 bits in length anywhere in memory, independent of byte alignments. The Extract instruction reads a bit field, expands the result to the length specified in the opcode, and then stores the expanded result into another operand. An Insert instruction reads an operand of the length specified in the opcode and stores the low-order part into a bit field.

2.2.3.4 Block, String, and Array Instructions

For the many iterative operations which are required in high-level languages, the Block Move and Block Compare instructions facilitate efficient generation of compiler code. They are written the same way as the standard memory-to-memory move and comparison instructions, except for the addition of a third displacement operand which specifies how many elements (bytes, words or double words) are to be moved or compared.

Strings of bytes, words, or double words are easily manipulated with the Move String, Compare String, and Skip instructions. To avoid destructive overwriting, move and compare operations can proceed from low addresses to high addresses, or vice versa. These operations can proceed unconditionally or be terminated when a comparison condition is met (when either a specific value is encountered or when a value is no longer encountered). Also, a string of instructions may be interrupted or aborted, and then restarted where it left off. These string instructions are comparable in their power to those available on large minicomputer and mainframe computers.

For array handling, two instructions are provided, Check and Index. The Check instruction determines whether an array index is within bounds. It allows the user to specify both an upper and a lower bound. It also subtracts the lower bound from the value being checked and stores the difference in a register, where it can be used in an Index instruction or in an index addressing mode.

The array Index instruction performs one step of a multidimensional array-address calculation. The opcode specifies the length of the second and third operands; the first operand is an address-data register. The Index instruction performs a multiplication and an addition, leaving the result in a register. The result is then used in another Index instruction for the next dimension, or it is used in an index addressing mode.

2.2.3.5 Jumps, Branches, and Calls

A number of different Jumps and Branches are implemented: simple Jump, Jump to Subroutine, simple Branch, Conditional Branch, and Multiway Branch (a branch is a PC-relative Jump). Since the displacement in these instructions can be as large as the PC, there is no limit to their range. In addition, several different returns are supported: return from subroutine, return from trap, and return from interrupt. The latter two are discussed in more detail in the section covering interrupts and traps, (section 2.4.3).

2.2.3.6 Register Manipulation Instructions

Any address-data register can be accessed via the general addressing modes. Thus any TI32000 instruction that uses a general addressing mode to access one of its operands can manipulate these registers. In addition, several instructions are provided explicitly for register manipulation.

The Save and Restore instructions manipulate the address-data registers. The instruction format for these operations includes an immediate field of 8 bits, each bit specifying which of the eight address-data registers are to be stored or fetched from the stack.

Instructions manipulating the dedicated registers allow these registers to be loaded and stored; bits in the program status register may be set and cleared, and the stack pointer may be adjusted.

2.2.3.7 Instruction Format

The TI32000 has a variable-length instruction format in which instructions are represented as a series of bytes. Figure 5 shows the general format of a TI32000 instruction.

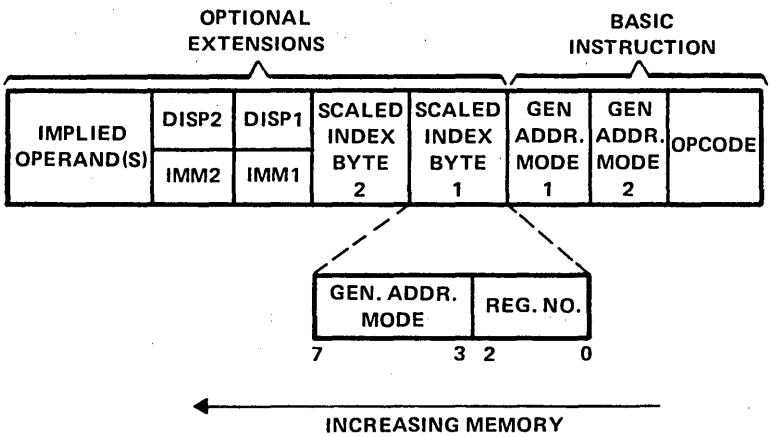


Figure 5. General Instruction Format

The Basic Instruction is one to three bytes long and contains the Opcode and up to two 5-bit General Addressing mode (gen) fields. Following the Basic Instruction field is a set of optional extensions, which may appear depending on the instruction and the addressing modes selected.

The Opcode specifies the operation to be performed, for example, ADD, MOV, etc., and the number of operands to be used in the instruction. The specification of an operand length (B, W, D, F, or L) is written appended to the opcode. For example, ADDW specifies the addition of two word-long operands, while MOVF specifies a move to a single-precision floating-point operand. The length specification in integer instructions is encoded in the basic instruction as B=00, W=01, or D=11; the length specification in floating-point instructions is encoded in the basic opcode as F=1 or L=0.

The General Addressing mode fields specify the addressing mode to be used to access the instruction's operands.

Index Bytes appear in the instruction format when either or both *gen* fields specify Scaled Index mode. In this case, the *gen* field specifies only the Scale Factor (1, 2, 4 or 8), and the Index Byte specifies which address-data register to use as the index and which addressing mode calculation to perform before indexing.

Following Index Bytes come any displacements (addressing constants) or immediate values associated with the selected addressing modes. Each Displacement/Immediate (Disp/Imm) field may contain one or two displacements, or one immediate value. The size of a *Disp* field is encoded within the top bits of that field, with the remaining bits interpreted as a signed (two's complement) value (see Figure 6). The size of an immediate value is determined from the *Opcode* field.

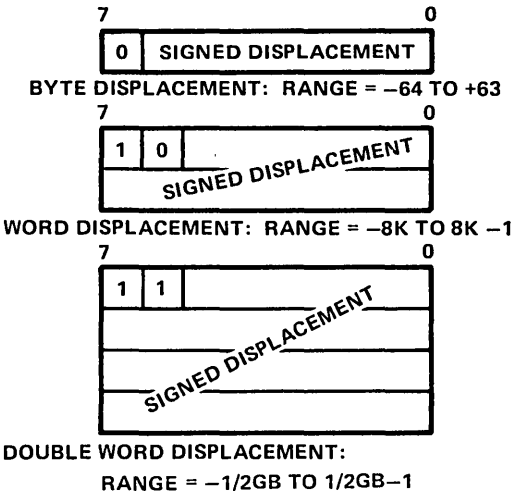


Figure 6. Displacement Encodings

2.2.3.8 Special Encodings

Two other special encodings, *reg* and *quick*, allow the very compact encoding of frequently used instructions. For example, there are quick forms of add, move and compare instructions which encode a small integer operand (range from −8 to +7) in place of a second general addressing mode. Some instructions require additional, 'implied' immediates and/or displacements, apart from those associated with addressing modes. Any such extensions appear at the end of the instruction, in the order that they appear within the list of operands in the instruction definition.

2.2.4 Register Set

The TI32000 architecture supports 33 registers, grouped into two register sets: 16 address-data registers and 17 dedicated registers (see Figure 7). Eight of the address-data registers are located on the CPU and eight are located on the FPU. The 17 dedicated registers include nine on the CPU, one on the FPU, and seven on the MMU. Besides storing operands and the results from arithmetic operations, these registers may also be used for the temporary storage of program instructions and control information concerning which instruction is to be executed next.

2.2.4.1 CPU Address-Data Registers

Internal to the CPU are eight 32-bit address-data registers R0 through R7, which provide local, high speed storage for the processor. They can be used to store bytes, words, double words, and quadruple words.

All address-data registers are available to all instructions. Thus, the compiler has freedom in its use of the registers and needn't do much housekeeping. The architecture also enables address-data registers to be used as accumulators, data registers, and address pointers. This represents a great improvement over machines that permit only a few registers to serve as address pointers, creating a bottleneck in address calculations, a very important function in high-level language programming.

2.2.4.2 CPU Dedicated Registers

The nine dedicated registers on the CPU chip are used for storing address and status information. The MOD register and the Processor Status Register are both 16 bits; the other registers are effectively 24 bits in length, although an additional eight bits (which in the current implementation are always set to zero) have been provided to allow for future expansion.

PC: The *Program Counter* register is a pointer to the first byte of the currently executing instruction. After the instruction is completed, the program counter is incremented to point to the next instruction. Since this register is 24 bits wide, all 16M bytes of memory can be directly addressed without the need for segmented addresses.

SP0, SP1: The *SP0* register points to the lowest address of the last item stored on the Interrupt Stack. This stack is normally used only by the operating system, primarily for temporary data storage and for holding return information for operating system subroutines and interrupt and trap service routines. The *SP1* register points to the lowest address of the last item stored on the User Stack. This stack can be used by normal user programs to hold temporary data and subroutine return information.

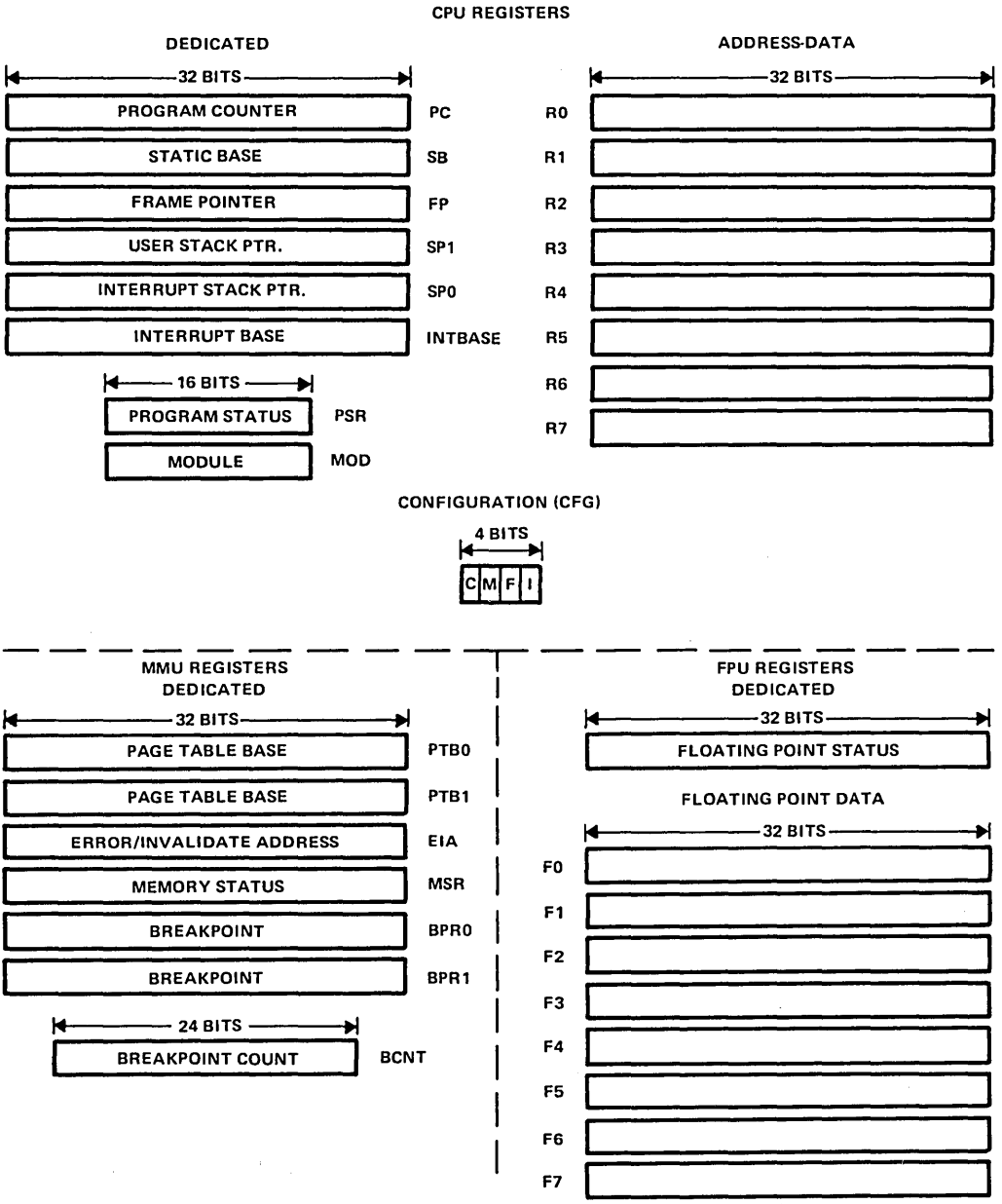


Figure 7. Register Set

FP: The *Frame Pointer* register is used by a procedure to access parameters and local variables on the stack. It is set up when a procedure is entered and points to the stack frame of the currently executing procedure, which contains the parameters for the currently executing subroutine and also the volatile (as opposed to static) local variables. The procedure parameters are addressed with positive offsets from the frame pointer; the local variables of the procedure are addressed with negative offsets from the frame pointer.

SB: The *Static Base* register points to the global variables of a software module. All references to a module's data are relative to this register. (See section 2.4.2.)

INTBASE: The *Interrupt Base* register holds the address of the dispatch table for interrupts and traps. (See sections 2.4.3.2 and 2.4.3.3.)

MOD: The *Module* register holds the address of the Module Descriptor of the currently executing software module. (See section 2.4.2.2.)

PSR: The *Processor Status register* holds the CPU status and control flags for the TI32000. The PSR is 16 bits long and is divided into two eight-bit halves. The low-order eight bits are accessible to all programs, but the high-order bits are accessible only to programs executing in Operating System mode. Among the bits in the PSR are the Carry bit, the Trace bit, (which causes a trap to be executed after every instruction), the Mode bit (which is set when the processor is in User Mode), the Interrupt Enable bit (which if set will cause interrupts to be accepted), and several other bits which can be used by comparison instructions.

CFG: The I bit indicates the presence of external interrupt vectoring circuitry (specifically, the TI32202 Interrupt Control Unit). If the CFG I bit is set, interrupts requested through the $\overline{\text{INT}}$ pin are 'vectored'; if it is clear, these interrupts are 'non-vectored'. The F, M, and C bits indicate the presence of the FPU, MMU, and Application-Specific Coprocessors. If these bits are not set, the corresponding instructions are trapped as being undefined.

2.2.4.3 FPU Registers

The *Floating Point Unit* registers are located on the Floating Point Unit coprocessors and consist of eight 32-bit address-data registers and a dedicated 32-bit Floating-Point Status Register. The eight floating-point registers can each store a single-precision operand or half of a double-precision operand. When 64-bit double-precision operands are to be operated upon, the specified register (n) and the next register (n + 1) are concatenated for the operation. Register n + 1 contains the high-order bits.

The *Floating-Point Status* register (FSR) holds mode control information, error bits, and trap enables. Like the other registers, the FSR is 32 bits wide. (See section 2.4.4.2.)

2.2.4.4 MMU Registers

The optional memory management architecture uses the following 32-bit dedicated registers to control address translation:

The *Page Table Base* registers (PT0 and PTB1) are controlled by the operating system and point to the starting location of the address translation tables in physical memory. All Operating System mode addresses are translated with the PTB0 register. User mode addresses are translated using this register if the Dual Space (DS) bit in the Memory Status Register (MSR) is one; if this bit is zero, the PTB1 register is used.

EIA: The *Error/Invalidate Address* register is used to invalidate addresses in the translation buffer. The translation buffer is a transparent cache of the most recently used pointer table entries. When an entry in a table is modified in memory, the copy of it in the translation buffer is deleted by writing the address of the affected virtual page into the EIA register. When a PTB register is modified, all cache entries made using that register are deleted. The EIA is also used to store the address which caused a memory management exception to occur.

MSR: The *Memory Status* register holds fields which control and examine the memory management status, and is accessible only in the Operating System mode. (See sections 2.4.4.2 and 2.4.4.3.)

Other registers in the MMU provide high-level software debug facilities during program execution.

2.2.5 Addressing Modes

Information encoded in an instruction includes a specification of the operation to be performed, the type of operands to be manipulated, and the location of these operands. An operand can be located in a register, in the instruction itself (as an immediate operand), or in memory. Instructions specify the location of their operands by nine *addressing modes*. Two addressing modes are used to access operands in registers and in instructions — Register mode and Immediate mode. The other modes are used to access operands in memory. The address of the operand is calculated in accordance with the desired addressing mode. The calculation is done by taking the sum of up to three components:

- a displacement element in an instruction
- a pointer (i.e., an address) in a register or in memory
- an index value in a register

The nine addressing modes may also be divided into standard modes for microprocessor architectures and those modes which are particularly suited to the operations and data structures of high-level languages.

2.2.5.1 Standard Modes

The following standard addressing modes are supported by the TI32000 architecture (see Figure 8 for a diagram of each one):

- Register
- Immediate
- Absolute
- Register relative

REGISTER: In the Register addressing mode, the operand is in one of the eight address-data registers. In certain Coprocessor instructions, an auxiliary set of eight registers may be referenced instead.

IMMEDIATE: The immediate mode operand is in the instruction. The length of the immediate mode operand is specified by the operand length or by the basic instruction length.

ABSOLUTE: With absolute mode, the operand address is the value of a displacement in the instruction.

REGISTER RELATIVE: The register relative mode computes an effective address (the operand address) by adding a displacement given in the instruction to a pointer in an address-data register.

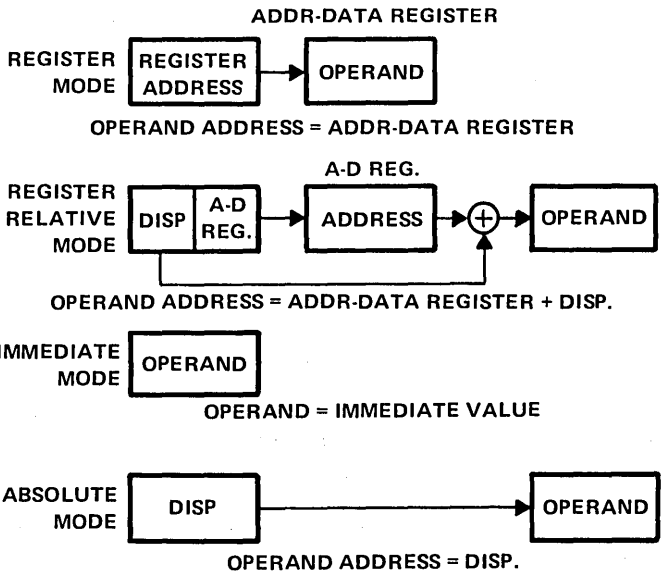


Figure 8. Standard Addressing Modes

2.2.5.2 High-Level Language Modes

In addition to these standard addressing mode types, the TI32000 employs several addressing mode types which, in combination with the already powerful instruction set, make the TI32000 a superb vehicle for high-level languages. They are listed below and diagrammed in Figure 9:

- Memory Space
- Memory Relative
- External
- Top-of-Stack
- Scaled Index

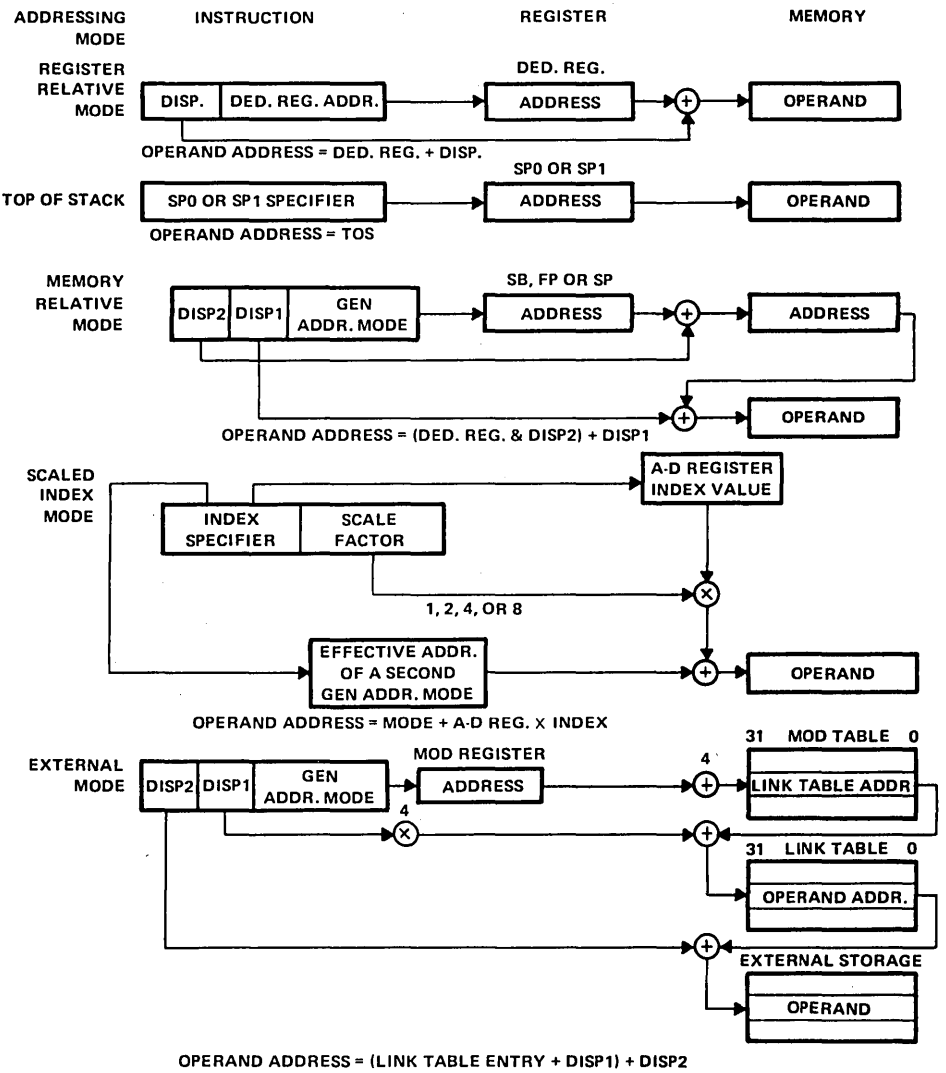


Figure 9. High-Level Language Addressing Modes

MEMORY SPACE: This addressing mode is identical to Register Relative, discussed above, except that the register used is one of the dedicated registers — PC, SP, SB or FP. These registers point to data areas generally needed by high-level languages.

MEMORY RELATIVE: The Memory Relative mode allows pointers located in memory to be used directly, without having to be loaded into registers (as is required in other microprocessors). Memory relative mode is useful for handling address pointers and manipulating fields in a record. When this addressing mode is used, the instruction specifies two displacements. The first displacement is added to a

specified dedicated register, and a double word is fetched from this address. The operand address is the sum of this value and the second displacement. In accessing records, the second displacement specifies the location of a field in the record pointed to by the double word. The exact size of the contents of this field is programmable.

EXTERNAL: The External Addressing mode is unique to the TI32000, and supports the software module concept, which allows the modules to be relocated without linkage editing. This mode is used to access operands that are external to the currently executing module. Associated with each module is a Link Table, containing the absolute addresses of external variables. The external addressing mode specifies two displacements: the ordinal number of the external variable (i e., the Link Table entry to be used) and an offset to a subfield of the referenced variable (e.g., a subfield of a Pascal record). (See section 2.4.2.)

TOP-OF-STACK: In this addressing mode, also unique to the TI32000, the currently selected Stack Pointer (SP0 or SP1) specifies the location of the operand. Depending on the instruction, the SP will be incremented or decremented, allowing normal push and pop facilities. This addressing mode allows manipulation or accessing of an operand on the stack by all instructions. For instance, the Top-of-Stack (TOS) value can be added to the contents of a memory location, a register, or to itself, and the result saved on the stack. On most other microprocessors, in which top-of-stack addressing is limited to a very small number of instructions, these manipulations would require several instructions to achieve the same results. The great advantage of this addressing mode is that it allows quick reference using a minimum number of bits to intermediate values in arithmetic computations.

SCALED INDEX: This addressing mode computes the operand address from one of the address-data registers and a second addressing mode. The register value is multiplied by one, two, four or eight (index byte, index word, index double, or index quad). The effective address of the second addressing mode is then added to the multiplied register value to form the final operand address. The Scaled Index mode is used for addressing into arrays, when the elements of the array are bytes, words, double words, floating-point numbers or long floating-point numbers.

2.3 Memory Organization

2.3.1 Introduction

Microprocessors were first developed when the design of complex, special-purpose chips became so expensive that it was more cost effective to use a general-purpose programmable device instead of a special-purpose chip. The programs for these early microprocessors were very small, typically requiring 2K to 8K bytes of memory and rarely exceeding 16K bytes. (This was just as well, since memory was very expensive.)

Now, almost exactly a decade since the microprocessor was invented, the memory requirements for typical applications approach those of minicomputers or even mainframes. Consequently, the memory organization issues discussed in section 2.1 have arisen.

In this section, we will cover the memory organization and memory management mechanisms of the TI32000. The key topics to be discussed are page-based mapping, virtual memory, memory protection, and virtual machines. The address translation, virtual memory, and memory protection mechanisms of the TI32000 architecture are contained in the TI32082 Memory Management Unit (MMU). The MMU also contains the logic for debugging as well as on-chip cache. Special instructions are provided in the TI32000 instruction set to control the MMU.

2.3.2 Mapping Mechanisms in the TI32000

Present TI32000 systems have a logical address space of 16 million bytes divided into 32,768 pages, each with a fixed size of 512 bytes. The potential physical address space is the same size and is also divided into similarly sized page frames. As described earlier, address translation (mapping) is the process of translating a logical address to a physical address. In the TI32000 architecture, address translation is done in units of a page. Thus two addresses next to each other in the same logical page will be next to each other in the same physical page frame, although two pages which are contiguous in logical memory may not be contiguous in physical memory.

For purposes of implementing the address translation, the 24 bits of a logical address may be thought of as consisting of two fields: the page selector field, which is the upper 15 bits, and the offset field, which is the lower 9 bits. Only the page selector bits are actually translated in the mapping process. The 9 bits of the offset specify a location within a page and are passed through the mapping process unaltered. The mapping process is performed automatically by the MMU.

Basically the mapping operation consists of treating the page selector field as an index into a table of physical addresses. Entries in this table hold the upper 15 bits of the physical address of a page frame. When a logical address is sent to the MMU, its lower 9 bits are appended to the 15-bit physical address in the table and the resulting 24-bit physical address is actually used to fetch data. (See Figure 10 for a diagram of this operation. This figure shows an abstract view of the TI32000 mapping operation; in reality, a two-level mapping is employed.)

2.3.2.1 Page Tables, Pointer Tables, and Entries

The address translation mechanism is carried out by tables in memory. The MMU contains a special register (PTB1) that points to the beginning of the *page table*. This table has 256 entries, each of which is 4 bytes wide, thus its total size is 1,024 bytes. Each entry in the page table points to a *pointer table*. Pointer tables contain 128 entries of 4 bytes, thus the pointer tables are each contained in a page. Each entry in a pointer table points to a physical page. (See Figure 11 for a diagram of this pointer tree.)

Each program or task can have its own page table. Changing the page table is simply a matter of changing an MMU register that points to the starting address of the current page table. Therefore, each program or task can have its own map from logical memory to physical memory and each program or task can have its own logical address space.

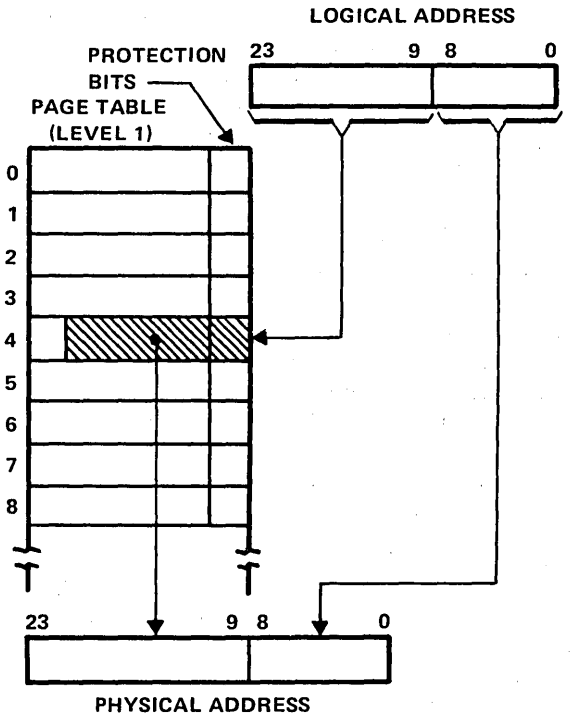


Figure 10. Mapping

Each entry in the page table or in one of the pointer tables has the same basic format (see Figure 12).

The high-order 23 bits contain the starting physical address of the specified page frame.

Bits 0 through 4 contain status bits:

- V The Valid Bit indicates whether the entry specifies a page that is present in memory. (See section 2.3.3.1, Page Faults and the Valid Bit.)
- R The Referenced Bit indicates whether the page has been accessed. This bit is automatically set when the corresponding page has been accessed for reading or writing. (See section 2.3.3.3, Support for Page Swapping Algorithms.)
- M The Modified Bit indicates whether the page has been written into. This bit is automatically set when any attempt is made to write to the corresponding page.
- PL The Protection Level field indicates the level of protection provided for the page. (See section 2.3.4, Protection.)

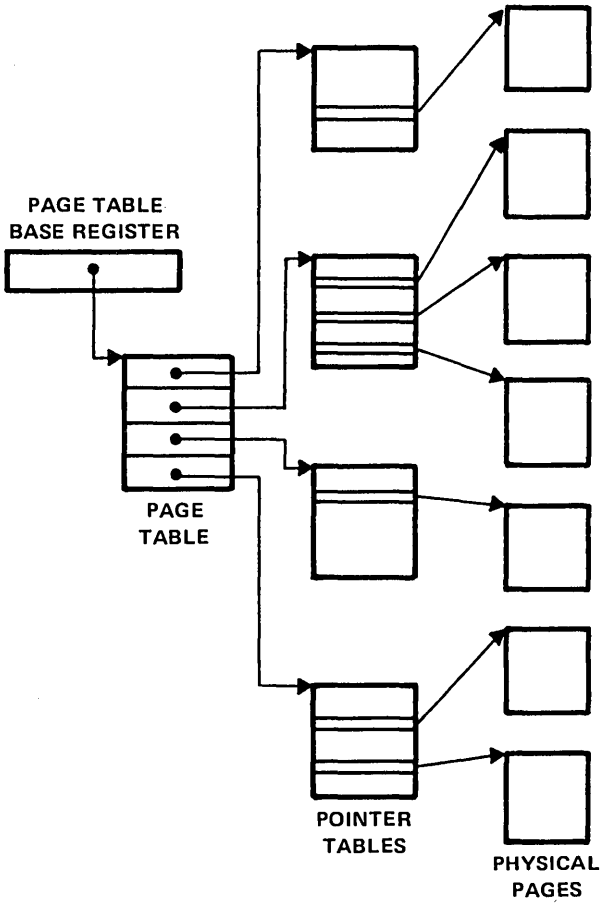


Figure 11. Hierarchy of Tables

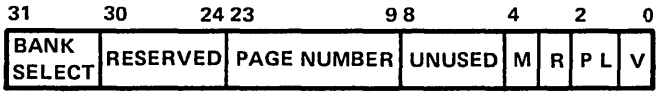


Figure 12. Page or Pointer Table Entry

2.3.2.2 The Complete Mapping Process

The mapping operation shown schematically in Figure 10, above, is actually accomplished by the following process:

The page selector component of the logical address (the high-order 15 bits), shown in Figure 10, actually consists of two subfields: the high-order 8 bits, which select an entry in the page table, and the low-order 7 bits, which select an entry in the

appropriate pointer table. (The offset component of a logical address specifies the displacement from the base of a page to the specified item.) Figure 13 shows a more complete version of the mapping process outlined in Figure 10.

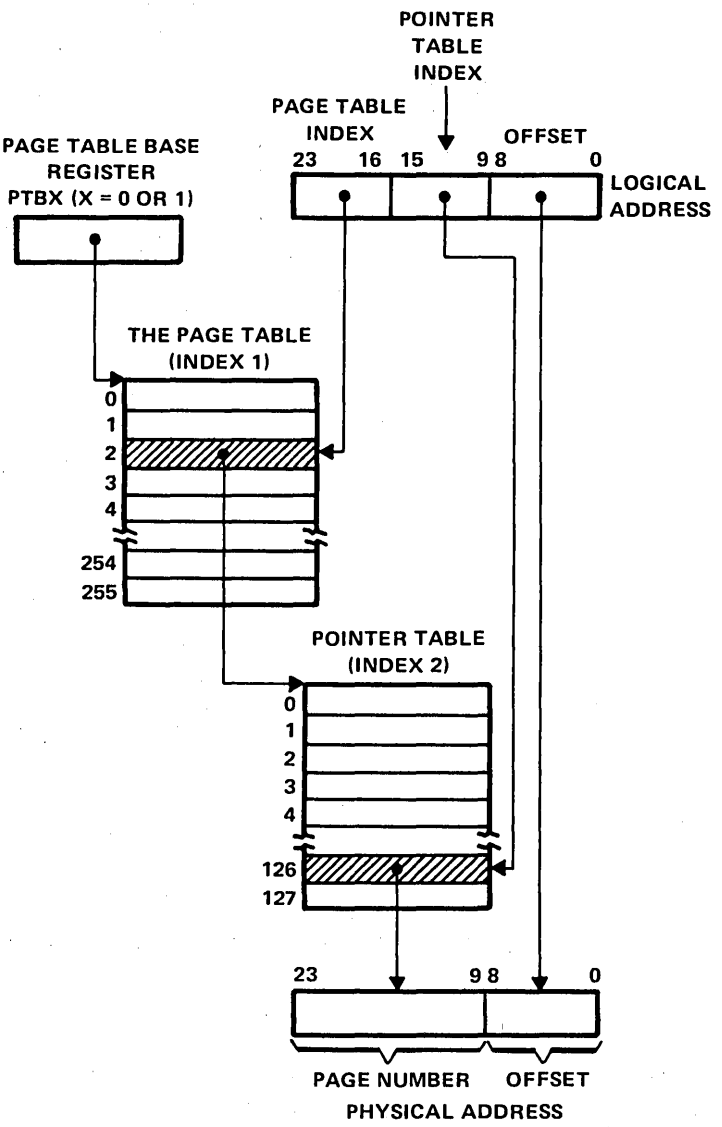


Figure 13. Table Driven Mapping

To speed up the mapping process, the MMU provides an associative cache on the chip itself. The cache contains the 32 most recently accessed logical addresses along with their translated physical addresses. Each entry consists of the high-order 15 bits of a logical address and the high-order 15 bits of the translated physical address (see Figure 14).

When a logical address is passed from the CPU to the MMU, the MMU first attempts to match that logical address with an entry in the cache. If the entry is present, the physical address portion of the entry is used immediately. If the entry is not present, the MMU must fetch the page table and pointer table entries from memory before address translation can be performed.

If the entry is present, address translation requires only one clock cycle. If the entry is not present, address translation will take slightly longer. This associative table is transparent to the user and calculations indicate that it dramatically speeds up address translation since the hit ratio (the percentage of time the cache contains the entry) is about 98%.

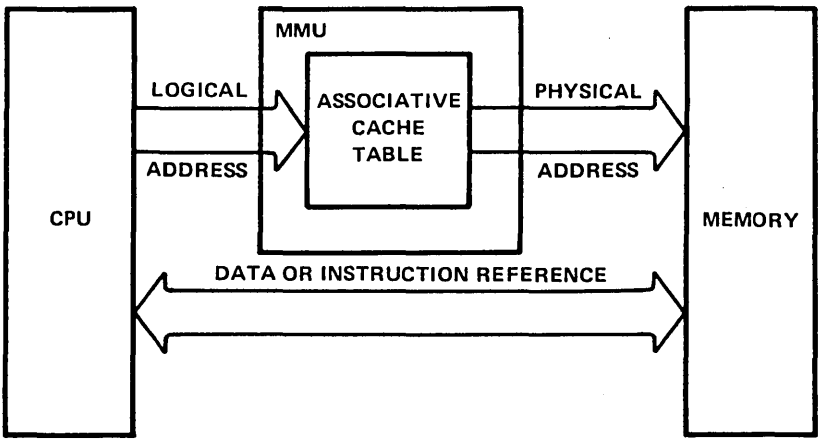


Figure 14. Associative Cache

2.3.3 Virtual Memory Mechanisms of the TI32000

Programs share many traits in common with human beings. For example, they obey Parkinson's Law. Just as work expands to fill the time available, so programs tend to expand over their lifetime to fill the physical memory available to them. Once the memory limits have been reached, further expansion of the program is difficult and error prone, usually requiring hard to manage overlays. The ideal solution to this problem is to give the program a virtually infinite (limitless) memory. A program in an infinite memory can be enlarged without bumping into any barriers. Unfortunately, memory costs usually preclude enormous physical memories. However, *virtual memory* gives the programmer the illusion of a gigantic memory at minimal cost.

With virtual memory, the user regards the combination of main and peripheral storage as a single large storage. The user can write large programs without worrying about the physical memory limitations of the system. To accomplish this, the operating system places some of the user programs and data in peripheral storage and brings them into main memory only as they are needed.

The TI32000 makes virtual memory operating systems easy to implement by means of its page-based mapping mechanism. Programs and data are swapped between main memory and secondary storage units of a page, as was described in section 2.1. In addition, the architecture provides several other mechanisms which support virtual memory.

Three bits in the page entry are important for virtual memory systems. These bits were discussed briefly in section 2.3.2.1. In the following three subparagraphs we will cover in much greater detail the use of these three bits in virtual memory systems. Also covered will be the instruction abortion/reexecution facility, the other TI32000 feature specifically designed to support virtual memory.

2.3.3.1 Page Faults and the Valid Bit

The *Valid Bit* in a page or pointer table entry indicates whether the corresponding page is present in main memory or not. Whenever an address is generated by the CPU and passed to the MMU for translation into a physical address, the MMU checks the valid bit of the table entry specified by the incoming logical address. If the valid bit is 1, the page is assumed to be present in main memory and address translation proceeds in the normal fashion.

However, if the valid bit is 0, then the page is assumed not to be in main memory and a *page fault* occurs. A page fault is a hardware generated trap that is used to tell the operating system to bring the missing page in from secondary storage. The page fault occurs in the MMU, which generates an Abort signal to the CPU. The Abort signal causes the CPU to immediately halt execution of the current instruction.

2.3.3.2 Instruction Aborting and Reexecution

When a page fault occurs, for whatever reason, the MMU sends the Abort signal to the CPU. At this point the CPU will stop executing the instruction and return any register that was altered by the instruction to its condition before the instruction started. The operating system will then be called to initiate a page swap. Once the appropriate page is in memory, the CPU and MMU also must insure that the aborted instruction can be reexecuted.

One of the problems in implementing virtual memory systems is that an instruction may generate a page fault at any time during the course of its execution. If the instruction itself occupies several bytes, it may overlap a page boundary and the act of fetching an instruction may itself cause a page fault. Or the process of fetching the source or destination operand may cause a page fault.

In order to permit the instruction to be restarted, the Abort signal usually causes the CPU to be returned to its state before the aborted instruction happened. The program counter is automatically saved as are the processor status register, the

stack pointer, and several other registers. When the operating system has completed the page swap, it executes a RETURN FROM TRAP instruction and execution resumes with the aborted instruction, with all registers being restored to their old values.

String handling instructions require special treatment during an abort. Obviously it is not desirable to have a long string instruction repeated from the beginning if an abort occurs somewhere in the string. The TI32000 provides for the aborted instruction to be reexecuted from the point where the problem occurred.

2.3.3.3 Support for Page Swapping Algorithms

To facilitate virtual memory implementation, two other bits in the page and pointer table entries are used: the Referenced Bit (R) and the Modified Bit (M).

It has been tacitly assumed that there is a vacant page frame in which to put the newly loaded page. In general, such will not be the case and it will be necessary to remove some page (i.e., copy it back into the secondary memory) in order to make room for the new page. Thus, an algorithm that decides which page to remove is needed.

Choosing a page to remove at random is certainly not a good idea. If the page containing the instruction is the one chosen, another page fault will occur as soon as an attempt is made to fetch the next instruction. Most operating systems try to predict which of the pages in memory is the least useful, in the sense that its absence would have the smallest adverse effect on the running program. One way of doing so is to make a prediction when the next reference to each page will occur and remove the page whose next reference lies farthest in the future. In other words, to try to select the page that will not be needed for a long time.

One popular algorithm evicts the page least recently used because that page has a high probability of not being in the working set. This algorithm is called the Least Recently Used algorithm. The Referenced bit can be used to implement a version of this algorithm.

The Referenced bit is set by the hardware when the page is referenced (read or written) by an instruction. By periodically checking and clearing this bit in all page and pointer table entries, the operating system can gain insight into the frequency with which pages are being used.[6] This information can be used to select pages to be swapped out, for example, on a least recently used basis.

If a page about to be evicted has not been modified since it was read in (a likely occurrence if the program contains program rather than data), then it is not necessary to write it back into secondary memory, as an accurate copy already exists there. If, however, it has been modified since it was read in, the copy in secondary storage is no longer accurate and the page must be rewritten. The Modified bit is set by the hardware whenever a page is written to during the time it is resident in main memory.

6. Peter J. Denning, 'Working Sets Past and Present,' IEEE *Transactions on Software Engineering*, SE-6, No. 1, 1980.

When the time comes to swap this page, the operating system can check this bit to see if there is a need for updating the copy on disc. If the bit is 1 (i.e., the page has been modified), then the page must be swapped out to secondary storage. However, if this bit is 0, then the page has not been modified since it was last read in and it can simply be discarded.

2.3.4 Memory Protection Mechanisms on the TI32000 Family

The page mechanism can also provide the basis for memory protection within a logical address space. Each page can have attributes associated with it that indicate how the page can be accessed. These attributes can allow reads only, reads and writes, or they can prevent any access at all. Entries in the page and pointer tables contain protection bits (the PL field) along with physical addresses (see section 2.3.2.1). These protection bits define the attribute of that page (e.g., read only).

The interpretation of the protection bits depends on the operating mode of the CPU. A given setting of the PL field will be interpreted differently when the CPU is in Operating System mode than when the CPU is in User mode. The bits have the following interpretation.

PL	Operating System Mode	User Mode
00	read only	no access
01	read and write	no access
10	read and write	read only
11	read and write	read and write

The operating system can treat a collection of pages with the same attributes as a segment. For example, a constants segment might be a set of pages containing data with the read-only attribute set, so users could not modify the data. Thus, page-based mapping provides a mechanism for implementing segmentation.

Intertask protection is accomplished by giving each task its own set of page tables. Thus each task has its own address space, providing maximum flexibility and virtual memory for each task. By changing the single register that points to the page table, one can switch to the new task's address space.

2.3.5 Virtual Machines

If the virtual memory hardware allows application software to execute in a different address space from the operating system, then it is possible to implement virtual machines. Software running on a virtual machine believes that it is running on a processor whose hardware provides the functions that are, in fact, provided by the operating system. In fact, the virtual memory hardware and I/O devices are simulated by the operating system with the aid of the real memory management hardware and I/O devices. Thus software which normally must be run alone (e.g., an operating system) can be run under the control of another operating system. This can be very useful for debugging a new operating system or running several incompatible operating systems on the same machine.

Figure 15 shows a simplified diagram of such a virtual machine.

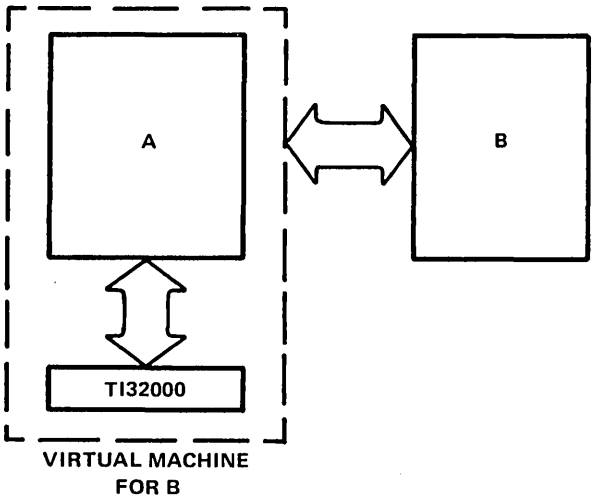


Figure 15. Virtual Machines

Operating system A and operating system B run in different address spaces. System A manipulates the actual TI32000 hardware, whereas system B manipulates an illusory machine consisting of the TI32000 hardware and virtual peripherals simulated by system A. The actual mechanisms employed to create such a virtual machine are somewhat technical and are covered in detail in the *TI32000 Programmer's Reference Manual.* Basically, system A constructs a simulated table onto the real page table. Virtual I/O devices are simulated similarly.

2.4 Other Features of the TI32000 Architecture

2.4.1 Introduction

This section will discuss additional architectural features of the TI32000 that reduce the traditional gap between the semantics of high-level programming languages and microprocessor architectures. Specifically, these are features which support good software design and programming practices. The topics covered include support for modular software design, input/output implementation, extension of the instruction set by means of coprocessors, and software debugging support.

2.4.2 Modular Software

Modular programming is one of the principle techniques for the systematic design of well-structured software. Large programs are among the most complex creations of human intellect. This complexity has been a major factor contributing to software unreliability. The concept of modularity in software design provides a means of overcoming natural human limitations for dealing with programming complexity by specifying the subdivision of large and complex programming tasks into smaller and simpler subtasks, or *modules*, each of which performs some well-defined

portion of the complete processing task. Such modules may then be independently designed, written, tested, and compiled, perhaps by different programmers working in parallel.

Programs which are written as a set of modules are more likely to be correct. They are more easily understandable and therefore more easily modified, maintained, and documented. Also, because communication between modules is permitted only through well-defined interfaces, the inner workings of a module need not be known to other modules. This protects a module's code and allows design changes to be done locally to a module without side effects on other modules or on the use of the system.

Nearly all HLLs incorporate features to support modular programming. For example, programs in Ada, the new Department of Defense high-order language, are composed of one or more program units — subprograms, packages or tasks — which can be compiled separately. In Pascal, separately compiled program modules may refer to variables, functions or procedures declared in another module by using certain extensions to the language, e.g., Import and Export directives.

The ultimate extension of the concept of modularity and the ultimate simplicity in software design and implementation is achieved when the modules are written to be used in ROM form. (For example, the VRTX real-time executive from Hunter & Ready, Inc.) Such software modules are simple hardware-like components and require a minimal amount of program design overhead.

Up to now, microprocessor architectures have provided inadequate and cumbersome architecture support for a modular programming methodology. The following section will discuss the problems associated with the implementation of modularity by a microprocessor; and the two subsequent sections will explain the TI32000's architectural solutions to these difficulties.

2.4.2.1 Overview

The major difficulty limiting the widespread use of libraries of ROM modules has been the necessity of modifying a module's addresses when it is linked with other separately compiled modules and loaded into memory for execution. Since addresses in ROM cannot be modified, it has been difficult to devise a uniform method of employing ROM modules in programs. Even when the module's code can be modified (e.g., modules on disk), this is a tedious and often lengthy enterprise.

The problems result from the fact that when several modules are combined into a single memory image, a module's final position can vary widely. Consequently, all addresses in jumps and calls or in data accesses that are dependent on knowing the module's absolute address at run time must be different according to where the module is loaded. Similarly, when a module calls another module, the address of the called module will be dependent on the relative position of the two modules. Thus, a module's code will not be identical for each position it occupies in memory and a linkage editor must be used to modify the addresses in each module according to its assigned position in memory.

2.4.2.2 Support Mechanisms

Software modules which have been compiled and assembled are known as *Object Modules* and are typically stored in relocatable object code. The function of a linkage editor is to merge the object modules into a single linear address space which may then be loaded into memory for execution. This requires binding (converting to an absolute value) all unresolved addresses. *Relocation* refers to the binding of the nonsequential addresses within the module (calls, returns, branches, and nonsequential data references); *linking* is the process of binding the addresses of subroutines or variables in other modules.

On the TI32000, editing of nonsequential addresses (jumps) within a module is not required, since TI32000 assembly language code is Position-Independent Code (PIC). This is achieved by the use of addressing modes which form an effective memory address relative to a base register — PC, FP, SP or SB. Since the relative distance between two nonsequential addresses remains constant, the same offset relative to the base register can be used in all positions in memory. This means a program can be loaded anywhere in memory and run correctly. In addition, facilities provided by the MMU allow a program to be moved in memory after it has been linked and loaded. This is especially important in time-sharing systems where programs must be swapped in and out of main memory to allow sharing of the processor. Also, because the base register relative addressing mode allows 30-bit signed displacements, which is 6 bits more than any present logical address, no code editing is ever necessary for branching, regardless of the amount of code in a module.

Position-independent code combined with the TI32000 virtual memory mechanism allows a program to be relocated in the logical address space as well as the physical address space. Machines that use paging or a relocation register, but lack base register relative addressing, allow programs to be moved in physical memory, but do not allow them to be moved to a different logical address after linking.

For references to variables and subroutines in other modules, the TI32000 provides a sophisticated linkage facility such that no editing of a module's external addresses is required.

To begin with, all programs for the TI32000 are organized as modules. Each module consists of three components:

1. The *Program Code* component contains the code to be executed by the processor and the module's constant data (or 'literals').
2. The *Static Data* component contains the module's global variables and data, i.e., data which may be accessed by all procedures within the module. In a Pascal program, for example, this component would contain the data structures declared in the outermost block.
3. The *Link Table* contains two types of entries: External Variable Descriptors and External Procedure Descriptors. The External Variable Descriptor is the absolute address of a variable located in the static data component or program code area of another module. This value is used in the External Addressing mode, in conjunction with the current Mod Table address (see below), to compute the effective address of the external variable. The

External Procedure Descriptor is used in the Call External Procedure (CXP) instruction and will be discussed later. There is one entry in the Link Table for each external variable and procedure referenced by the module.

In a typical system, the linker program (in conjunction with the loader) specifies the locations of the three components of a module. The static data and Link Table typically reside in RAM; the code component can be either RAM or ROM. The three components can be mapped into noncontiguous locations in memory and each can be independently relocated. Since the Link Table contains the absolute addresses of external variables, the linker need not assign absolute memory addresses for these in the module itself; they may be assigned at load time.

To allow the transfer of control from one module to another, the TI32000 provides three structures: a Module Table in memory and two dedicated registers on the CPU.

1. The *Module Table* is set up in random-access memory starting at logical address 0 and contains a *Module Descriptor* for each module in the address space of the program. A Module Descriptor has four 32-bit entries corresponding to each component of a module:
   - The *Static Base* entry contains the base address of the beginning of the module's static local data area.
   - The *Link Base* points to the beginning of the module's Link Table.
   - The *Program Base* is the address of the beginning of the code and constant data for the module. Since a module may have multiple entry points, this pointer is used with an offset from the Link Table to find them. One entry is currently unused but has been allocated to allow for future expansion.
2. The *Mod Register* on the CPU contains the address of the Module Descriptor for the current module.
3. The *Static Base Register* contains a copy of the Static Base component of the Module Descriptor of the currently executing module, i.e., it points to the beginning of the current module's static data area.

See Figure 16 for a illustration of a module's environment.

On the TI32000, modules need not be linked together prior to loading. As modules are loaded, a linking loader simply updates the Module Table and fills the linkage table entries with the appropriate values. No modification of a module's code is required. Thus, modules may be stored in read-only memory and may be added to a system independently of each other, without regard to their individual addressing. Also, since the pointers in the Module Table reach any point within the address space, modules can be located anywhere in memory.

2.4.2.3 Programming with Modules

The Call External Procedure (CXP) instruction is used to execute a procedure residing in another module. Recall that the Link Table contains two types of entries for each module in the program's address space: External Variable Descriptors and External Procedure Descriptors. The latter entries consist of two 16-bit fields each. The MODULE field contains the address of the referenced procedure's Module Table entry. The OFFSET field is an unsigned number giving the position of the entry point

relative to the new module's Program Base pointer (in called module's Mod Table). This allows a called procedure to be found automatically, without requiring the calling routine to supply any addressing information.

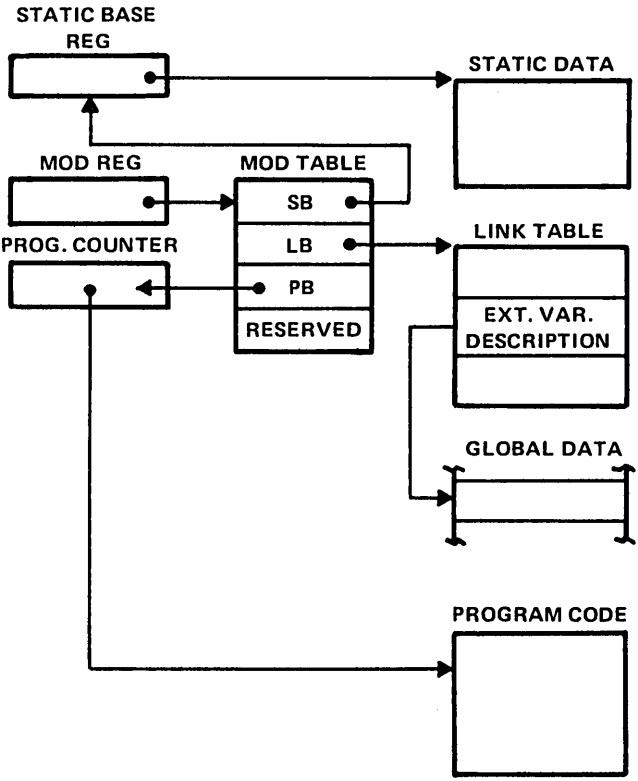


Figure 16. Module Run-Time Environment

Figure 17 depicts the execution of the CXP instruction where Module #2 calls Module #3.

This instruction automatically performs the following sequence of operations.

1. The External Procedure Descriptor for Module #3 is found by adding a displacement specified in the instruction to the Link Table address of Module #2. In the assembly language program this displacement is represented by a label name; the actual numerical value of the displacement is assigned by the assembler.
2. The current status of Module #2 is saved by pushing the contents of its PC and Mod registers onto the stack.
3. The Module field of the Link Table's External Procedure Descriptor for Module #3 is moved into the MOD register so that this register now points to the Module Table for Module #3.

4. The Static Base value in the Module Table is placed in the Static Base Register (this is done to speed up accesses to the module's static variables, which would otherwise be referenced by indexing into the Module Table).
5. The Offset field in the External Procedure Descriptor is added to the contents of the Mod Table's Program Base and this value is placed in the PC. The CPU is now in the environment of Module #3.

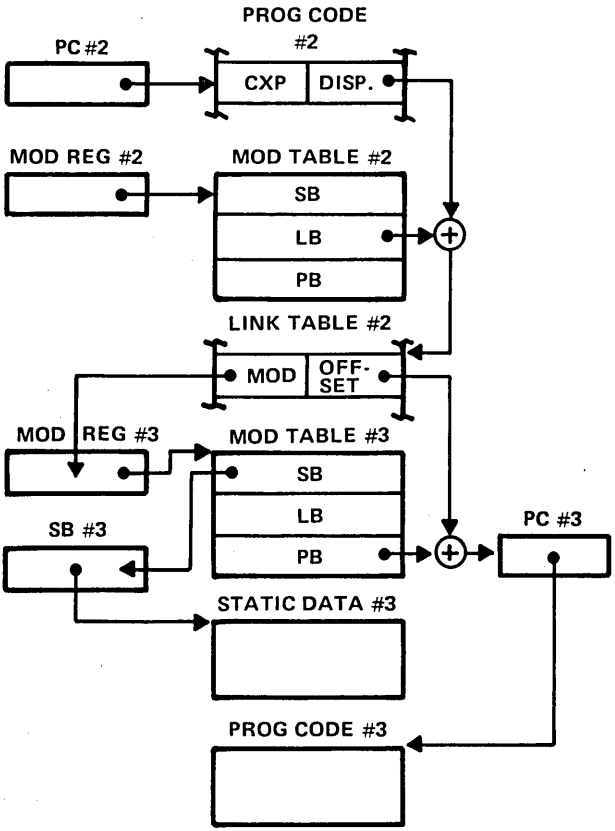


Figure 17. CXP Instruction

The Call External Procedure with Descriptor (CXPD) instruction allows an External Procedure Descriptor to be passed as a parameter to a called module. The address of a function or procedure from the calling module's Link Table is pushed onto the stack and the called module may then use this value to call the procedure.

The Enter and Exit instructions minimize the overhead in procedure calls by automatically managing the resources that must be allocated at the beginning of a procedure and reclaimed at the end.

The Enter instruction saves the Frame Pointer (FP) of the calling module on the stack and loads the Stack Pointer value into the FP register so that they now point to the same location, i.e., the saved FP value on the stack. Space on the stack is allocated for the procedure's local variables and a specified number of registers required for use by the procedure are pushed on the stack. See Figure 18 for an example of one procedure calling another.

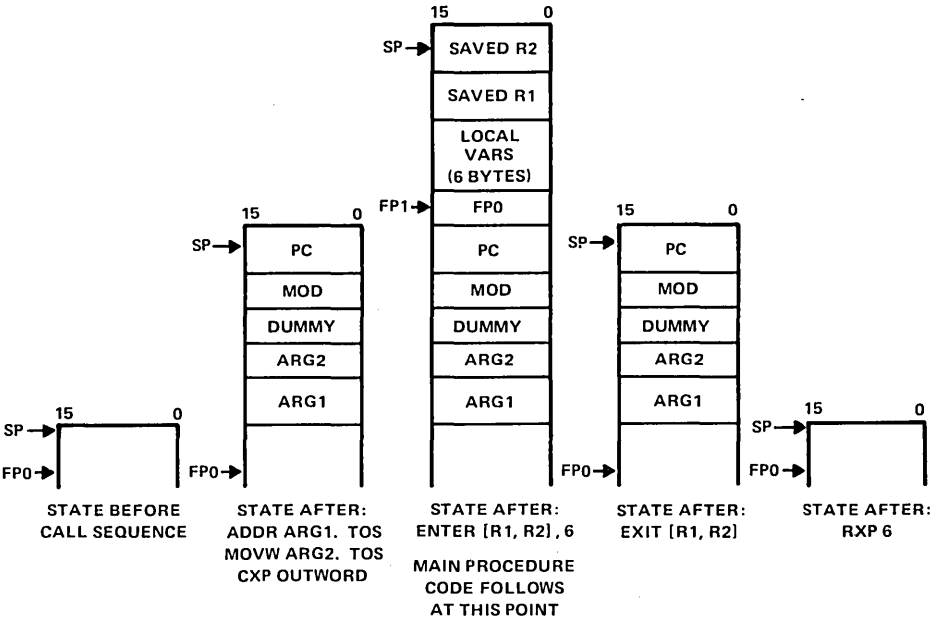


Figure 18. Stack Flow for Procedure Calls

The TI32000's use of the FP allows the procedure to allocate local variables on the stack and address them as fixed offsets from the FP. Also, once the local storage is allocated, the stack can still be used for temporary storage without affecting the addressing of the local variables. The programmer need not keep track of the changing offset between the SP and local storage, which is especially advantageous for nested procedure calls and recursive functions.

The Exit instruction automatically restores the registers saved by the Enter instruction, loads the value of the FP into the Stack Pointer thus deallocating the procedure variables, and restores the previous FP.

The Return from External Procedure (RXP) instruction restores the Static Base, the Mod register, and the PC of the calling procedure. In addition, this instruction may be used to remove the parameters which were passed to the called procedure.

Data accesses by modules are provided in the following manner.

1. Parameters and local variables on the stack may be stored and accessed with the Memory Space addressing mode or the Memory Relative addressing mode using the FP register. Parameters are addressed with positive offsets from the FP; local variables are addressed with negative offsets from the FP.
2. A module's static data is accessed by using the Memory Space addressing mode with the Static Base (SB) register. Since displacement fields relative to SB register can be 1, 2 or 4 bytes, no limit is imposed on the amount of static data a module may have. Note that on other microprocessors, which handle static data in the same way as any other external references, no protection is provided for accesses by other modules. The TI32000 provides this protection at the hardware level. The Mod Table allows each module to have its own static data area so that a procedure being executed by a module will not modify that module's data. In applications requiring two or more tasks to be executing the same code concurrently, this protection is essential to insure reentrancy.
3. For operands that are external to the currently executing module, the External addressing mode is used. This addressing mode specifies two displacements. The first is added to the Link Base entry in the Mod Table to obtain the External Variable Descriptor entry in the Link Table. The second displacement is added to the External Variable Descriptor to compute the effective address of the operand. Since both displacements may be as large as the logical address space, there is no limit to the size of the Link Table or to the size of the external variable (which might be a structure rather than a single data element).

Indexing by the contents of any one of the CPU's eight address-data registers is an option on all addressing modes which generate an effective address to memory, so that a static or external variable can also be an array. For example, to access an array that has been passed by reference, the starting address of the array may be found by using the Memory Space mode relative to the FP; this value can then be loaded into an address-data register and used with the Scaled Index mode.

2.4.3 Input/Output

The input/output structure defined by a computer's architecture provides the interface between the central processor and the outside world, as well as between the processor and its secondary storage devices, external support circuits, and coprocessors.

The first two sections will discuss one aspect of the TI32000's architectural support for I/O operations, specifically, its sophisticated and efficient exception handling mechanism.

2.4.3.1 Overview

Program exceptions are conditions which alter the normal sequence of instruction execution, causing the processor to suspend the current process and call the operating system for service. An exception resulting from the activity of a source external to the processor is known as an *interrupt*; an exception which is initiated by

some action or condition in the program itself is called a *trap*. Thus, an interrupt need not have a relationship to the executing program, while a trap is caused by the executing program and will recur each time the program is executed. The TI32000 recognizes 12 exceptions: 9 traps and 3 interrupts.

The exception handling technique employed by an interrupt-driven processor determines how fast the processor can perform input/output transfers, the speed with which transfers between tasks and processes can be achieved, and the software overhead required for both. Therefore, it determines to a large extent the efficiency of a processor's multiprogramming and multitasking (including real-time) capabilities.

Exception handling on the TI32000 makes use of the hardware structures provided for external procedure calls and, in addition, establishes a *Dispatch* Table in memory whose base address is contained in the CPU Interrupt Base register. This table contains an External Procedure Descriptor for each interrupt service procedure required. See Figure 19.

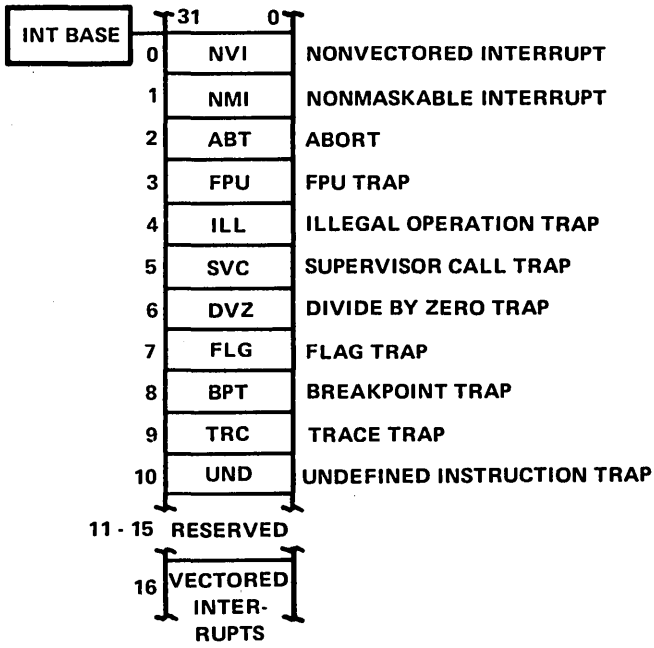


Figure 19. Dispatch Table

For purposes of addressing the Dispatch Table, each of the 12 exceptions has been assigned a number. This exception number (or Interrupt vector) is used to compute the starting address of the service procedure for the particular exception required, i.e., the exception number is multiplied by 4, added to the contents of the Interrupt Base register, and this value is used as an index into the Dispatch Table to obtain the External Procedure Descriptor of the service routine to call.

When an exception occurs, the CPU automatically preserves the complete machine state of the program immediately prior to the occurrence of the exception. Depending on the kind of exception, it will restore and/or adjust the contents of the Program Counter, the Processor Status register, and the current Stack Pointer. A copy of the PSR is made and pushed onto the Interrupt Stack. The PSR is set to reflect Operating System mode and the selection of the service routine's Interrupt Stack. The Interrupt exception number is then used to obtain the address of the External Procedure Descriptor from the Dispatch Table and an External Procedure Call is made. As with any such call, the Mod register and the Program Counter are pushed onto the Interrupt Stack. See Figure 20.

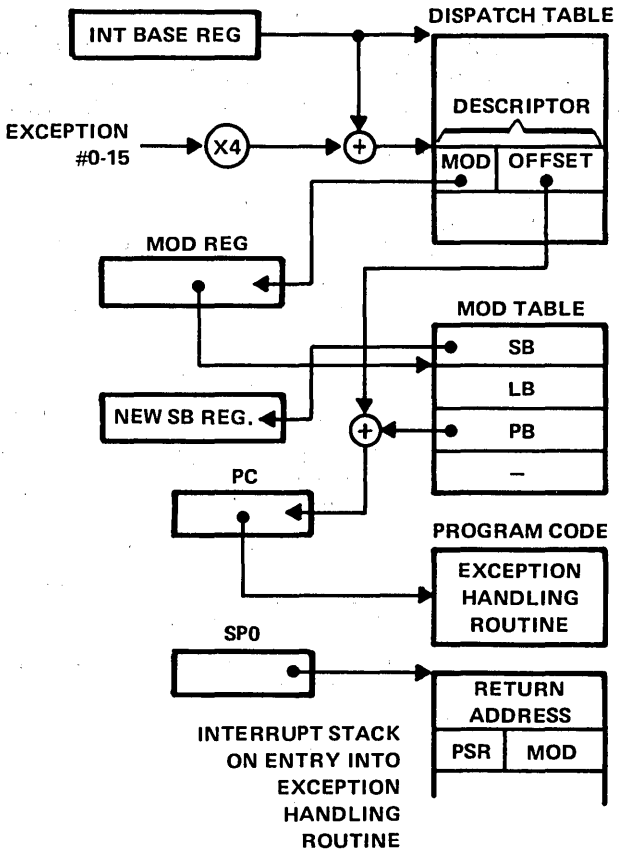


Figure 20. Nonvectored Interrupts and Traps

To return control to the interrupted program, one of two instructions is used. The Return From Trap instruction (RETT) is used for all traps and nonmaskable interrupts. It restores the PSR, the Mod register, and the PC and SB registers to their

previous contents and, since traps are often used deliberately as a call mechanism for Operating System mode procedures, it discards a specified number of parameters from the User's stack. See Figure 21.

For maskable interrupts, the Return from Interrupt (RETI) instruction is used. It is basically the same as the Return From Trap instruction except that any Interrupt Control Units (see section 2.4.3.3) are informed that interrupt service has completed. Also, since interrupts are generally asynchronous external events, this instruction does not pop any parameters.

The TI32000 implements a five level priority system for scheduling exceptions which occur in the same instruction. They are ordered as follows:

1. Traps other than trace (highest priority)
2. Abort trap
3. Nonmaskable interrupt
4. Maskable interrupts
5. Trace trap (lowest priority)

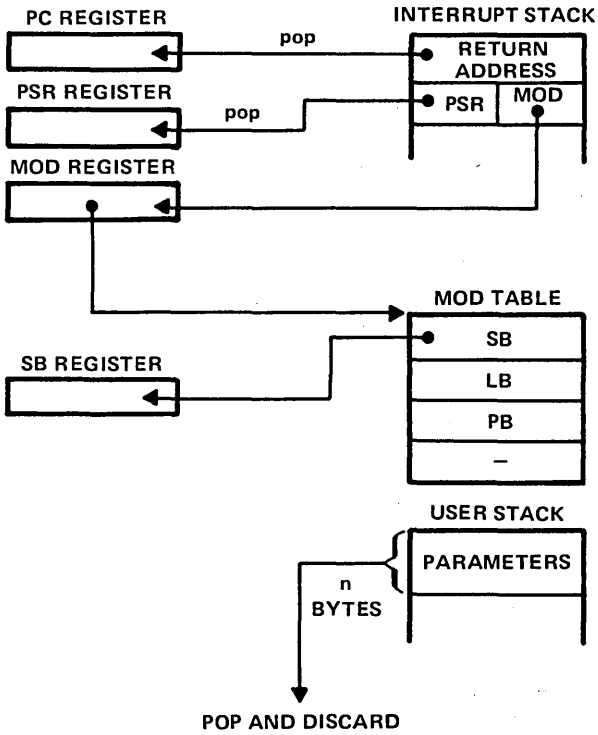


Figure 21. Return from Trap Instruction

Maskable interrupts may individually be assigned separate relative priorities (see below). Exceptions with the same priority are serviced in the order received.

This, then, is the basic plan for exception handling on the TI32000. The specifics of interrupt and traps are discussed in the following two sections of this chapter.

2.4.3.2 Interrupts

The TI32000 provides three types of interrupts: Nonmaskable, Vectored, and Nonvectored.

Nonmaskable interrupts cannot be disabled and occur when catastrophic events (such as imminent power failure) require immediate handling in order to preserve system integrity. A nonmaskable interrupt also occurs when a breakpoint condition is met. (See section 2.4.5.2)

The Nonvectored interrupt mode may be used by smaller systems in which an interrupt priority system is not required. In this case, no index into the Dispatch Table is needed and the CPU simply uses a default vector of zero.

For Vectored interrupts, prioritization of interrupt requests is provided by the TI32202 Interrupt Control Unit. The basic idea in a priority interrupt mechanism is that each device, along with its interrupt handler, is assigned a rank indicating its priority. An interrupt handler can then be interrupted only by devices with a higher priority.

Each Interrupt Control Unit can prioritize up to 16 interrupt requests, eight of which can be from external peripheral devices. The ICU provides a vector used as an index into the Dispatch Table to obtain the address of the service routine required. In a system with only one ICU, the vectors provided must be in the range of 0 through 127.

To further expand the interrupt handling capability of a system, a single TI32202, acting as the Master ICU, can be cascaded with up to 16 additional TI32202s, providing up to 256 levels of hardware or software interrupt. To support the cascaded configuration, a *Cascade Table* is established in memory, in a negative direction from the Dispatch Table. The entries in the table are the 32-bit addresses pointing to the Vector Registers in each ICU. To address the Cascade Table, the ICU provides a negative vector number. The fact that it is a negative number indicates to the CPU that the interrupt vector is from a cascaded ICU. See Figure 22 for a detailed explanation of cascaded interrupts.

The Interrupt Control Unit can function in either a fixed priority or an auto-rotate mode. In auto-rotate mode, the interrupt source, after being serviced, is rotated automatically to the lowest priority position.

All interrupts except the nonmaskable interrupt may be disabled by the program with the Bit Clear in PSR instruction; each of the ICU's 16 interrupt sources can be individually masked by setting a bit in that device's Mask Register.

Interrupt handling on the TI32000 provides a number of features which contribute to efficiency and programming flexibility. For example, on some microprocessors, all registers are automatically saved when an interrupt occurs. The TI32000 automatically saves only the Program Counter, the Program Status Register and the Mod register; the other registers are under program control. They may be saved and

restored by specifying the required ones in a single instruction, allowing for extreme flexibility in adjusting interrupt response speed. Fast context switching for interrupts is facilitated by the treatment of all memory locations as though they are internal address-data registers by virtue of memory-to-memory operations. This allows a temporary variable to be left in memory during a context switch. Also, the use of an Interrupt Stack allows context switching in a multiprogramming or multitasking environment to be done without having to disable interrupts.

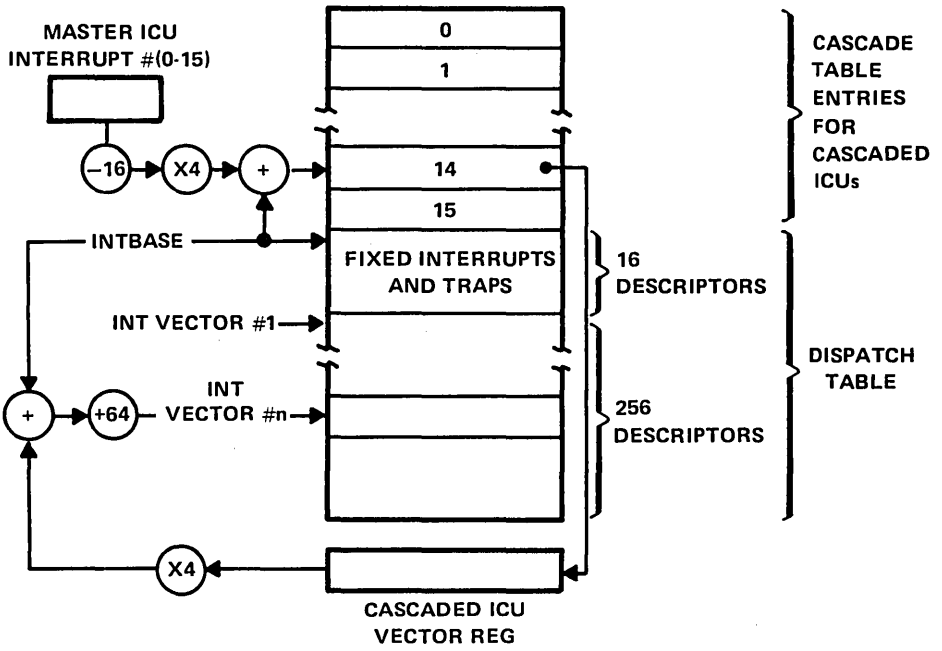


Figure 22. Cascaded Vectored Interrupts

2.4.3.3 Traps

The TI32000 recognizes nine traps. Three of the traps are implemented by explicit instructions: the Flag Trap (FLAG) allows overflow checking in any arithmetic operation and is enabled by setting the F bit in the PSR; the Supervisor Call Trap (SVC) is used to transfer to system mode software in a controlled way, typically to access facilities provided by the operating system. The Breakpoint Trap (BPT) instruction is used for program debugging, and is discussed in section 2.4.5.2.

The Abort Trap (ABT) occurs when an attempt is made to access a protected page in memory or when page swapping is required in the MMU. If the page fault occurs in a string instruction, the processor state is set to reflect the progress made by the instruction up to the time of the trap; all other instructions are reexecuted from the beginning.

The Illegal Trap (ILL) results when a privileged instruction occurs while the processor is in the User mode. Traps are also provided for undefined opcodes (UND), for attempted division by zero (DVZ), and for the occurrence of an exceptional condition in an FPU or Application Specific Coprocessor (ACU) instruction. The Trace Trap is enabled by setting the T bit in the PSR and is used for program debugging.

All traps except the Trace trap occur as an itegral part of the execution of an instruction, and are serviced before interrupts. The return address pushed by any trap except the Trace trap is the address of the first byte of the instruction during which the trap occurred; the return address of a Trace trap is the address of the next instruction to be traced.

2.4.3.4 Memory-Mapped I/O

The architecture of the TI32000 implements a *memory-mapped I/O* system, in which peripheral devices are treated as a specified section of memory. The basic motivation of a memory-mapped system is to allow the use of the full range of the microprocessor's instructions and addressing modes for I/O operations.

Each device interface is organized as a set of registers (or ports) that responds to read and write commands to locations in the normal address space of the microprocessor. For example, a memory store becomes an I/O *write* if a peripheral device is addressed; a load from memory becomes an I/O *read*. A compare with memory is a very powerful instruction that can take a group of input data and successively compare their magnitude with a value in a register. Also, data in an external device register can be tested or modified directly, without bringing it into memory or disturbing the address-data registers.

Memory-mapped I/O allows I/O operations to be performed directly in a high-level language, i.e., an I/O device may be declared as a data structure and then manipulated with the use of pointers. In an isolated I/O system, assembly language subroutines for I/O must be written and then called by the HLL. Memory-mapped I/O also insures that the I/O space is protected by the same memory management facilities that are used to protect critical areas of memory.

2.4.4 Coprocessors

A *coprocessor* is an auxiliary processing unit that operates in coordination with the TI32000 CPU, allowing architectural capabilities which, in view of the limitations in contemporary integration technology, could not otherwise be provided. Communication between the master CPU and the coprocessors takes place by means of a very fast, well defined, and self-contained protocol, which is transparent to the programmer.

The TI32000 family now includes two coprocessors: the TI32081 Floating Point Unit and the TI32082 Memory Management Unit. In addition, the TI32000 CPUs provide the capability of communicating with a user-defined, generalized *Application-Specific Coprocessor.*

2.4.4.1 Overview

A TI32000 CPU recognizes three groups of instructions as being executable by external coprocessors: 1) Memory Management Instructions, 2) Floating-Point Instructions, and 3) Application-Specific Coprocessor Instructions.

Coprocessor instructions have a three-byte Basic Instruction field, consisting of an ID byte followed by an Operation Word. The ID Byte identifies the instruction as being a coprocessor instruction, specifies which coprocessor will execute it, and determines the format of the following Operation Word of the instruction. The Operation Word specifies the size and number of operands, the addressing modes used to access them, and the type of operation to be performed.

In all coprocessor operations, the CPU fetches the instruction, performs any address calculation that may be needed, and then routes the instruction with the appropriate data to the coprocessor for execution. The actual data manipulation is handled by the coprocessor. If the necessary coprocessor chip is not in the system, the CPU generates a software trap, allowing the instruction to be emulated with software routines.

Though the coprocessor is external to the host CPU, all of the CPU's registers and facilities (such as effective address calculation, memory bus interface, etc.) can be considered an integral part of the system.

A four-bit CFG register, located in the control section of the TI32000 CPU, indicates to the CPU the presence of coprocessors in the system configuration (see Figure 7). The F, M, and C bits indicate the availability of the FPU, the MMU, and an Application-Specific Coprocessor. The I bit indicates the presence of the TI32202 Interrupt Control Unit. These four bits must be set by the user during system initialization with the Set Configuration instruction (SETCFG).

There are no restrictions on the number of coprocessors that can be used in the system, as long as only one coprocessor of each kind is on the bus. Thus, four or five coprocessors, each with a different instruction set, could work alongside the CPU on the same bus.

The coprocessor concept has two main advantages for software development. First, the coprocessors are so designed that when integration technology advances to the point where coprocessor hardware can be incorporated within the CPU chip, no software modifications will be required — the same programs will simply execute much faster. Second, the programmer has the option of building an entry-level system without coprocessors by using software emulators. Later, higher performance systems can be built by simply adding the coprocessor chips and removing the emulators.

2.4.4.2 MMU

The MMU provides dynamic address translation, virtual memory management, memory protection, and both hardware and software debugging support.

The MMU address translation and virtual memory mechanisms are described in section 2.3; section 2.4.5 covers the debugging facilities of the MMU. In addition, six

instructions are provided for manipulating the MMU's status. The Read Address Validate (RDVAL) instruction and the Write address validate (WRVAL) instruction provide read and write address translation validation for the user mode. The Load MMU Register (LMR) instruction allows the programmer to store data into any of the MMU registers. The Store MMU Register (SMR) instruction allows any register to be read.

The MOVSU and MOVUS instructions permit the operating system to transfer data to and from user space. Without these instructions, the operating system would have no way of accessing data in the user's address space. Many microprocessors that distinguish Operating System mode from User mode lack this instruction and the design of operating systems for these machines is adversely affected.

2.4.4.3 FPU

The FPU extends the TI32000 instruction set with very high-speed floating-point operations for both single- and double-precision operands, as well as 8, 16 and 32-bit fixed point calculations.

The FPU contains eight 32-bit data registers and a 32-bit Floating-Point Status Register (FSR) which contains mode control information, the floating-point error bits, and trap enables. The data registers contain 32-bit single-precision operands; for 64-bit double-precision operands, two registers are concatenated.

Unlike other microprocessors which support floating-point operations, the architecture of the TI32000 makes available to the FPU all the TI32000 addressing modes. For example, the Scaled Index mode permits an array of floating-point data elements to be addressed by its logical index, rather than its physical address. Also, any instructions can be register-to-register, register-to-memory, or memory-to-memory.

The FPU executes 18 instructions which supplement the integral arithmetic instructions and provide conversion from one precision type to another. Three separate processors in the chip manipulate the mantissa, sign, and exponent, respectively, under the control of microcode stored on the chip. (See section 2.2.3.2)

Traps are provided for overflow, underflow, divide by zero, reserved operand, invalid operations, illegal instructions, and inexact results. All traps can be individually enabled or disabled by the programmer.

2.4.4.4 Application-Specific Coprocessors

The user-defined Application-Specific Coprocessor (ACU) instruction set can be used to control any generic external chip. This chip is assumed to need some opcodes for arithmetic-like calculations, some opcodes for data moves, and some opcodes for examining and modifying status registers. The instruction set defines the instruction formats, the operand classes, and the communication protocol. Left to the user are the interpretations of the Op Code fields, the programming model of the ACU, and the actual types of data transferred. The protocol specifies only the size of an operand, not its data type.

2.4.5 Debugging Facilities

Debugging is one of the most difficult stages in program development. Though structured design techniques and modular programming have helped to reduce program debugging time, 20% of software development effort remains committed to this enterprise. Clearly, any debugging assistance provided by the hardware is of particular value. The support provided by the TI32000 is unique for microprocessors.

2.4.5.1 Overview

Hardware support is provided for an operation crucial to program debugging: *breakpointing*. The implementation of this operation uses a set of registers on the MMU and the Breakpoint Trap instruction.

2.4.5.2 Breakpoint Trap and MMU Breakpoint Registers

Setting breakpoints is a technique for halting a program's execution at a particular instruction or data access for purposes of examining the program's state and thereby determining the cause of improper program behavior.

On the TI32000, breakpoints may be set either when a specified address is accessed or after a specified number of such accesses have been made. Also, more than one breakpoint address may be simultaneously selected, allowing a halt to be implemented after either fork of a conditional branch. These facilities are provided by the Breakpoint Trap instruction (BPT) and three dedicated registers located on the MMU.

The Breakpoint Trap instruction is a one byte instruction which replaces the first byte of the opcode of the instruction that is to be breakpointed. To allow breakpoints to be set in PROM, as well as RAM, two Breakpoint registers, BPR0 and BPR1, are provided. These registers hold the double word addresses of two selected breakpoints which are compared with the contents of the address bus for every memory cycle. When a breakpoint address appears in the program and other conditions specified by the contents of the register are met, a nonmaskable Interrupt occurs.

Because these registers are located in the MMU, they may be selected to look at either the logical addresses from the CPU or the physical addresses from the MMU. In addition, the Breakpoint registers may be designated to operate when the indicated address is either written to or read from or when there is an instruction fetch.

A third register on the MMU, the Breakpoint Count register, specifies the number of matches of the BPR0 register breakpoint condition to pass over before a breakpoint occurs. This is useful for selecting a particular iteration in a loop instruction. See Figure 23 for a schematic representation of the operation of the three Breakpoint registers. In this example, the program contains a loop which will be executed 100 times. For purposes of debugging, the breakpoint is set to occur on the last time through the loop. This is done by setting BPR0 to the address of the particular instruction and by setting the BC register to 99, this being one less than the number of times the loop will be executed in the program.

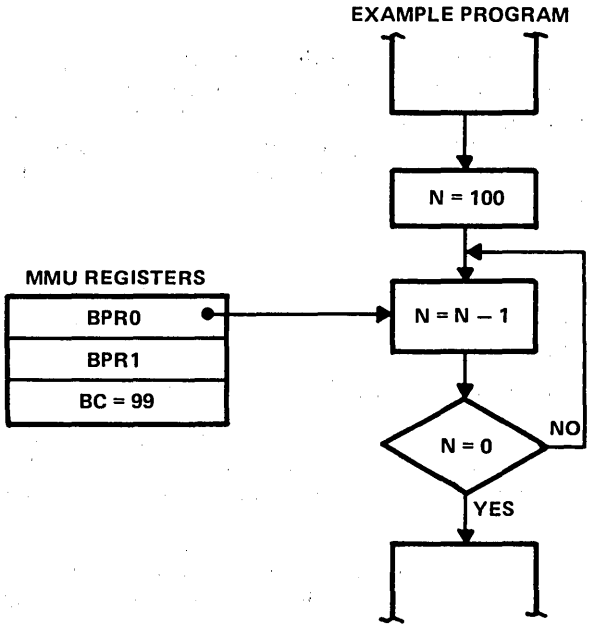


Figure 23. Breakpointing

In most other microprocessors, breakpointing is provided by a trap or breakpoint instruction which single steps the CPU. This can result in a myriad of problems for a virtual memory system. First and foremost is the fact that all addresses emanating from the CPU are logical addresses. It is often necessary when debugging Operating System mode software to be able to set breakpoints at absolute addresses; i.e., at addresses in physical memory. This is not possible with CPU-based debugging techniques, since the CPU has no concept of the distinction between the two types of addresses. Also, the setting of breakpoints with special instructions that overlay existing code can cause much additional overhead for the memory manager. For these and other reasons, the designers of the TI32000 have chosen to implement debug support on the MMU.

3.1 TI32032T Microprocessor

3.1.1 Programming Model

The TI32000 microprocessor family architecture includes 16 registers on the TI32032T Central Processing Unit (CPU) (Figure 1).

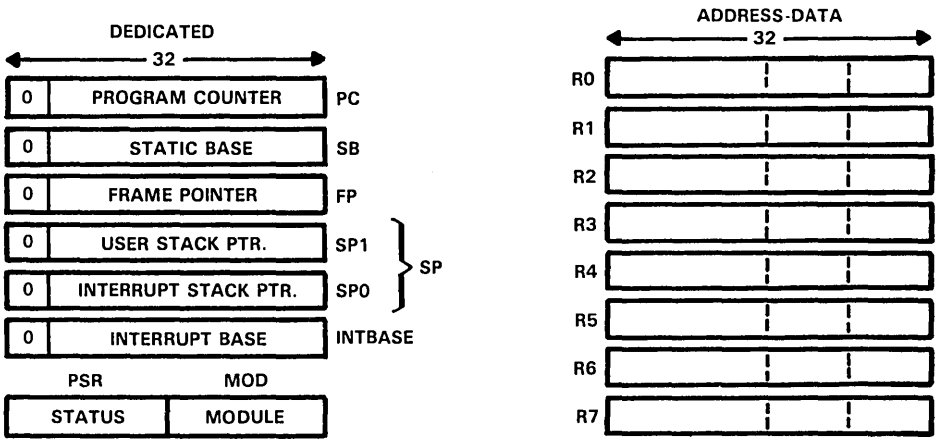


Figure 1. Address-Data and Dedicated Registers

3.1.1.1 Address-Data Registers

The TI32032T contains eight registers (R0 through R7) for meeting high-speed general-storage requirements, such as for holding temporary variables and addresses. These registers are free for any use by the programmer. Each is 32 bits in length. If an Address-Data Register is specified for an operand that is 8 or 16 bits long, only the low part (8 or 16 bit section) of the register is used and the high part is not referenced or modified.

3.1.1.2 Dedicated Registers

The eight dedicated registers of the TI32032T are assigned specific functions.

PC: THE PROGRAM COUNTER Register is a pointer to the first byte of the instruction currently being executed. The PC Register is used to reference memory in the program section. In the TI32032T CPU, the upper eight (most significant) bits of this register are always zero.

SP0, SP1: The function of the STACK POINTER Registers is as follows: (1) The SP0 Register points to the lowest address of the last item stored in the Interrupt Stack. This stack is normally used only by the operating system. It is primarily used for storing

temporary data, and holding return information for operating system subroutines and interrupt and trap service routines. (2) The SP1 Register points to the lowest address of the last item stored on the User Stack. This stack is used by normal user programs to hold temporary data and subroutine return information.

In this document, reference is made to the SP Register. The terms "SP Register" or "SP" refers to either SP0 or SP1, depending on the setting of the S bit in the Processor Status Register (PSR). If the S bit in the PSR is 0, then SP refers to SP0. If the S bit in the PSR is 1, then SP refers to SP1. In the TI32032T CPU, the upper eight (most significant) bits of these registers are always zero.

Stacks in the TI32000 microprocessor family grow downward in memory. A Push operation predecrements the Stack Pointer by the operand length. A Pop operation postincrements the Stack Pointer by the operand length.

FP: The FRAME POINTER Register is used by a procedure to access parameters and local variables on the stack. The FP Register is set up on procedure entry with the ENTER instruction and stored on procedure termination with the EXIT instruction.

The FP Register holds the address in memory occupied by the old contents of the Frame Pointer. In the TI32032T CPU, the upper eight (most significant) bits of this register are always zero.

SB: The STATIC BASE Register points to the global variables of the software module. This register is used to support relocatable global variables for software modules. The SB Register holds the lowest address in memory occupied by the global variables of a module. In the TI32032T CPU, the upper eight (most significant) bits of this register are always zero.

INTBASE: The INTERRUPT BASE Register holds the address of the dispatch table for interrupts and traps (section 3.1.3.8). The INTBASE register holds the lowest address in memory occupied by the dispatch table. (In the TI32032T CPU, the upper eight (most significant) bits of this register are always zero.

MOD: The MODULE REGISTER holds the address of the module descriptor of the currently executing software module. The MOD register is 16 bits long, therefore the module table must be contained within the first 64k bytes of memory.

PSR: The PROCESSOR STATUS Register holds the status codes for the TI32032T microprocessor. The PSR, as shown in Figure 2, is 16 bits long, divided into two 8-bit halves. The low-order 8 bits are accessible to all programs, but the high-order 8 bits are accessible only to programs executing in Operating System Mode.

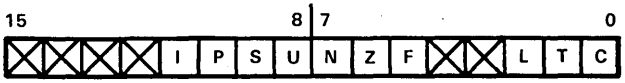


Figure 2. Processor Status Register

C: The C bit indicates that a carry or borrow occurred after an addition or subtraction instruction. It can be used with the ADDC and SUBC instructions to perform multiple-precision integer arithmetic calculations. It may have a setting of 0 (no carry or borrow) or 1 (carry or borrow).

T: The T bit causes program tracing. If this bit is a 1, a Trace Trap (TRC) is executed after every instruction (section 3.1.3.8.5).

L: The L bit is altered by comparison instructions. In a comparison instruction, the L bit is set to "1" if the second operand is less than the first operand, and both operands are interpreted as unsigned integers. Otherwise, it is set to "0". In floating-point comparisons,this bit is always cleared.

F: The F bit is a general condition flag, which is altered by many instructions (e.g., integer arithmetic instructions use it to indicate overflow).

Z: The Z bit is altered by comparison instructions. In a comparison instruction, the Z bit is set to "1" if the second operand is equal to the first operand; otherwise it is set to "0".

N: The N bit is altered by comparison instructions. In a comparison instruction, the N bit is set to "1" if both operands are interpreted as signed integers and the second operand is less than the first operand. Otherwise, it is set to "0".

U: If the U bit is "1", no privileged instructions may be executed. If the U bit is "0", then all instructions may be executed. When U = 0, the TI32032T is said to be in Operating System Mode; when U = 1, the TI32032T is said to be in User Mode. A User Mode program is restricted from executing certain instructions and accessing certain registers which could interfere with the operating system. For example, a User Mode program is prevented from changing the setting of the flag used to indicate its own privilege mode. An Operating System Mode program is assumed to be a trusted part of operating system, hence it has no such restrictions.

S: The S bit specifies whether the SP0 Register or SP1 Register is used as the Stack Pointer. The S bit is automatically cleared on interrupts and traps. It may have a setting of 0 (use SP0 Register) or 1 (use SP1 Register).

P: The P bit prevents a TRC trap from occurring more than once for an instruction (section 3.1.3.8.5). It may have a setting of 0 (no trace pending) or 1 (trace pending).

I: When the I bit is "1", all interrupts will be accepted (section 3.1.3.8). If the I-bit is "0", only the NMI interrupt is accepted. Trap enables are not affected by this bit.

3.1.1.3 Configuration Register (CFG)

Within the Control section of the TI32032T CPU is a four-bit CFG Register that declares the presence of certain external devices. It is referenced by only one instruction, SETCFG, which is intended to be executed only as part of system initialization after reset. The format of the CFG Register is shown in Figure 3.

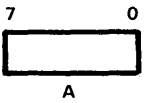
Figure 3. CFG Register

The CFG I bit declares the presence of external interrupt vectoring circuitry, specifically, the TI32202 Interrupt Control Unit. If the CFG I bit is "1", interrupts requested through the $\overline{INT}$ pin are "vectored". If it is clear, these interrupts are "nonvectored" (section 3.1.3.8).

The F, M, and C bits declare the presence of the Floating Point Unit (FPU), Memory Management Unit (MMU), and Application-Specific Coprocessors. If these bits are 0s, the corresponding instructions are trapped as being undefined.

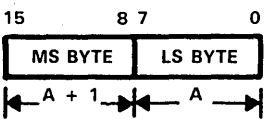
3.1.1.4 Memory Organization

The main memory of the TI32032T is a uniform linear address space. Memory locations are numbered sequentially starting at 0 and ending at $2^{24} - 1$. The number specifying a memory location is called an address. The contents of each memory location is a byte consisting of 8 bits. Unless otherwise noted, diagrams in this document show data stored in memory with the lowest address on the right and the highest address on the left. In addition, when data is shown vertically, the lowest address is at the top of a diagram and the highest address is at the bottom of the diagram. When bits are numbered in a diagram, the least significant bit is given the number 0, and is shown at the right of the diagram. Bits are numbered in increasing significance and toward the left.



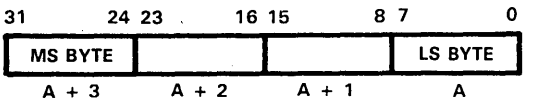
Byte at Address A.

Two contiguous bytes are called a word. Except where noted (section 3.1.2.1), the least significant byte of a word is stored at the lower address, and the most significant byte of the word is stored at the next higher address. In memory, the address of a word is the address of its least significant byte, and a word may start at any address.



Word at Address A.

Two contiguous words are called a double word. Except where noted (section 3.1.2.1), the least significant word of a double word is stored at the lowest address and the most significant word of the double word is stored at the address two greater. In memory, the address of a double word is the address of its least significant byte, and a double word may start at any address.



Double word at Address A.

Although memory is addressed as bytes, it is actually organized as double words. Note that access time to a word or a double word depends upon its address, e.g., double words that are aligned to start at addresses that are multiples of four will be accessed more quickly than those not so aligned. This also applies to words that cross a double-word boundary.

3.1.1.5 Dedicated Tables

Two of the TI32032T dedicated registers (MOD and INTBASE) serve as pointers to dedicated tables in memory (section 3.1.3.8).

The INTBASE Register points to the Interrupt Dispatch and Cascade tables.

The MOD register contains a pointer into the Module Table, whose entries are called Module Descriptors. A Module Descriptor contains four pointers, three of which are used by the TI32032T. At any time, the MOD register contains the address of the Module Descriptor for the currently running module. It is automatically updated by the Call External Procedure instructions (CXP and CXPD).

The format of a Module Descriptor is shown in Figure 4. The Static Base entry contains the address of static data assigned to the running module.It is loaded into the CPU Static Base register by the CXP and CXPD instructions. The Program Base entry contains the address of the first byte of instruction code in the module. Since a module may have multiple entry points, the Program Base pointer serves only as a reference to find them.

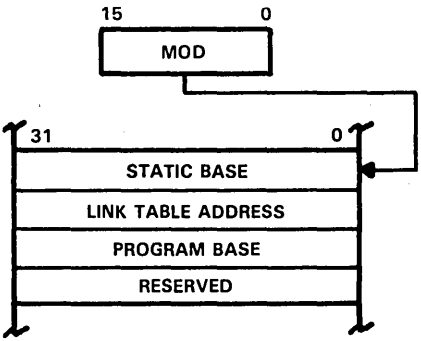


Figure 4. Module Descriptor Format

The Link Table Address points to the Link Table for the currently running module. The Link Table provides the information needed for:

1. Sharing variables between modules. Such variables are accessed through the Link Table via the External addressing mode.
2. Transferring control from one module to another. This is done via the Call External Procedure (CXP) instruction.

The format of a Link Table is shown in Figure 5. A Link Table Entry for an external variable contains the 32-bit address of that variable. An entry for an external procedure contains two 16-bit fields: Module and Offset. The Module field contains the new MOD register contents for the module being entered. The Offset field is an unsigned number giving the position of the entry point relative to the new module's Program Base pointer.

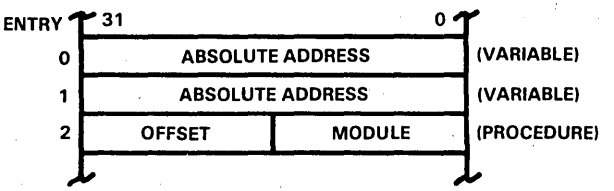


Figure 5. A Sample Link Table

For further details of the functions of these tables, refer to the *TI32000 Programmer's Reference Manual*.

3.1.2 Instruction Set

3.1.2.1 General Instruction Format

Figure 6 shows the general format of a TI32000 instruction. The Basic Instruction is one to three bytes long and contains the Opcode and up to two 5-bit General Addressing Mode (gen) fields. Following the Basic Instruction field is a set of optional

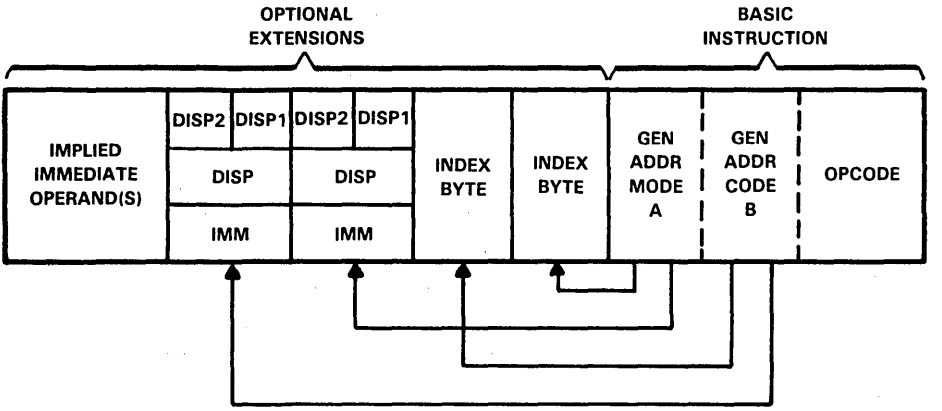


Figure 6. General Instruction Format

extensions, which may appear depending on the instruction and the addressing modes selected. Index Bytes appear when either or both *gen* fields specify Scaled Index. In this case, the *gen* field specifies only the Scale Factor (1, 2, 4, or 8) and the Index Byte specifies which Address-Data Register to use as the index and which addressing mode calculation to perform before indexing. See Figure 7.

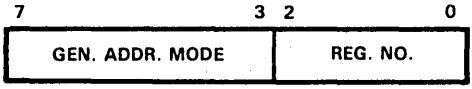


Figure 7. Index Byte Format

Following Index Bytes come any displacements (addressing constants) or immediate values associated with the selected addressing modes. Each Displacement/Immediate (*Disp*/*IMM*) field may contain one or two displacements, or one immediate value. The size of a Displacement field is encoded within the top bits of that field, as shown in Figure 8, with the remaining bits interpreted as a signed (two's complement) value. The size of an immediate value is determined from the Opcode field. Both *Disp* and *IMM* fields are stored most significant byte first. Note that this is different from the memory representation of data (section 3.1.1.4).

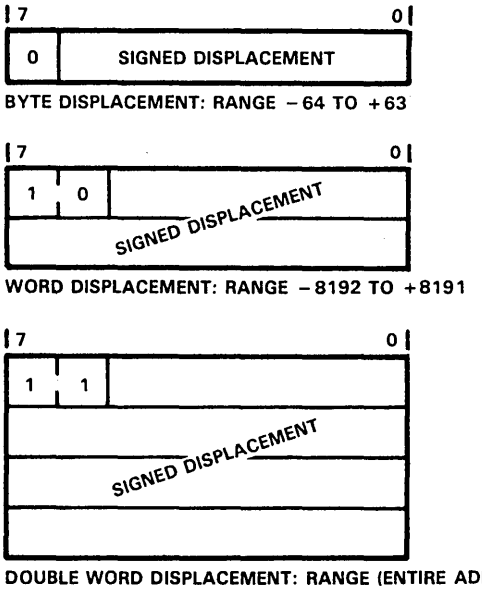


Figure 8. Displacement Encodings

Some instructions require additional, "implied" immediates and/or displacements, apart from those associated with addressing modes. Any such extensions appear at the end of the instruction, in the order that they appear within the list of operands in the instruction definition (section 3.1.2.3).

3.1.2.2 Addressing Modes

The TI32032T CPU generally accesses an operand by calculating its Effective Address based on information available when the operand is to be accessed. The method to be used in performing this calculation is specified by the programmer as an "addressing mode".

Addressing modes in the TI32032T are designed to optimally support high-level language accesses to variables. In nearly all cases, a variable access requires only one addressing mode within the instruction that acts upon that variable. Extraneous data movement is therefore minimized.

TI32032T Addressing Modes fall into nine basic types:

1. Register — The operand is available in one of the eight Address-Data Registers. In certain Coprocessor instructions, an auxiliary set of 8 registers may be referenced instead.
2. Register Relative — An Address-Data Register contains an address to a diplacement value from the instruction, yielding the Effective Address of the operand in memory.
3. Memory Space — Identical to Register Relative above, except that the register used is one of the dedicated registers PC, SP, SB, or FP. These registers point to data areas generally needed by high-level languages.
4. Memory Relative — A pointer variable is found within the memory space pointed to by the SP, SB, or FP register. A displacement is added to that pointer to generate the Effective Address of the operand.
5. Immediate — The operand is encoded within the instruction. This addressing mode is not allowed if the operand is to be written.
6. Absolute — The address of the operand is specified by a displacement field in the instruction.
7. External — A pointer value is read from a specified entry of the current Link Table. To this pointer value is added a displacement, yielding the Effective Address of the operand.
8. Top-of-Stack — The currently selected Stack Pointer (SP0 or SP1) specifies the location of the operand. The operand is pushed or popped, depending on whether it is written or read.
9. Scaled Index — Although encoded as an addressing mode, Scaled Indexing is an option on any addressing mode except Immediate or another Scaled Index. It has the effect of calculating an Effective Address, then multiplying any Address-Data Register by 1, 2, 4, or 8 and adding it to the total, yielding the final Effective Address of the operand.

Table 1 is a brief summary of the addressing modes. For a complete description of their actions, see the *TI32000 Programmer's Reference Manual.*

3.1.2.3 Instruction Set Summary

Table 2 presents a brief description of the TI32032T instruction set.The Format Column refers to the Instruction Format Tables (See Appendix).The Instruction Column gives the instruction as coded in assembly language, and the Description column provides a short description of the function provided by that instruction. Further details of the exact operations performed by each instruction may be found in the *TI32000 Programmer's Reference Manual.*

Notations:

i = integer length suffix: B = Byte
W = Word
D = Double Word

f = Floating Point length suffix: F = Standard Floating
L = Long Floating

gen = General operand: Any addressing mode can be specified.

short = A 4-bit value encoded within the Basic Instruction (see Appendix for encoding).

imm = Implied immediate operand. An 8-bit value appended after any addressing extensions.

disp = displacement (addressing constant): 8, 16, or 32 bits. All three lengths equal.

reg = Any address-data register: R0-R7.

areg = Any Dedicated address register: SP, SB, FP, MOD, INTBASE, PSR,US (bottom 8 PSR bits).

mreg = Any Memory Management Status/Control Register.

creg = An Application-Specific Coprocessor Register (Implementation Dependent).

cond = Any condition code, encoded as the 4-bit field within the Basic Instruction (see Appendix for encodings).

3.1.3 Functional Description

3.1.3.1 Power and Grounding

The TI32032T requires a single 5-V power supply, applied on pin 18 (V_{CC}). See DC specifications in the TI32032T data sheet.

Grounding connections are made on three pins. Logic Ground (GNDL, pin 44) is the common pin for on-chip logic, and Buffer Grounds (GNDB1, pin 43 and GNDB2, pin 11) are the common pins for the output drivers. For optimal noise immunity, it is recommended that GNDB1 and GNDB2 be connected together through a single conductor, and GNDL be directly connected to the center of this conductor. All other ground connections should be made to the common line as shown in Figure 9.

In addition to V_{CC} and GND, the TI32032T CPU uses an internally generated negative voltage. It is necessary to filter this voltage externally by attaching a pair of capacitors (Figure 9) from the BBG pin to ground. Recommended values for these are:

C1: 1 μF, Tantalum.

C2: 1000 pF, low inductance. This should be either a disc or monolithic ceramic capacitor.

Table 1. TI32000 Addressing Modes

ENCODING	MODE	ASSEMBLER SYNTAX	EFFECTIVE ADDRESS
Register			
00000	Register 0	R0 or F0	None: Operand is in the specified register
00001	Register 1	R1 or F1	
00010	Register 2	R2 or F2	
00011	Register 3	R3 or F3	
00100	Register 4	R4 or F4	
00101	Register 5	R5 or F5	
00110	Register 6	R6 or F6	
00111	Register 7	R7 or F7	
Register Relative			
01000	Register 0 relative	disp(R0)	Disp + Register.
01001	Register 1 relative	disp(R1)	
01010	Register 2 relative	disp(R2)	
01011	Register 3 relative	disp(R3)	
01100	Register 4 relative	disp(R4)	
01101	Register 5 relative	disp(R5)	
01110	Register 6 relative	disp(R6)	
01111	Register 7 relative	disp(R7)	
Memory Relative			
10000	Frame memory relative	disp2(disp1(FP))	Disp 2 + Pointer; Pointer found at address Disp1 + Register. "SP" is either SP0 or SP1, as selected in PSR.
10001	Stack memory relative	disp2(disp1(SP))	
10010	Static memory relative	disp2(disp1(SB))	
Immediate			
10100	Immediate	value	None: Operand is input from instruction queue.
Absolute			
10101	Absolute	@disp	Disp.
External			
10110	External	EXT (disp1) + disp2	Disp2 + Pointer; Pointer is found at Link Table Entry number Disp1.
Top of Stack			
10111	Top of Stack	TOS	Top of current stack, using either User or Interrupt Stack Pointer, as selected in PSR. Automatic Push/Pop included.

Table 1. TI32000 Addressing Modes (Continued)

ENCODING	MODE	ASSEMBLER SYNTAX	EFFECTIVE ADDRESS
Memory Space			
11000	Frame memory	disp(FP)	Disp + Register, "SP" is either SP0 or SP1, as selected in PSR.
11001	Stack memory	disp(SP)	
11010	Static memory	disp(SB)	
11011	Program memory	* + disp	
Scaled Index			
11100	Index, bytes	mode[Rn:B]	EA (mode) + Rn.
11101	Index, words	mode[Rn:W]	EA (mode) + 2 × Rn.
11110	Index, double words	mode[Rn:D]	EA (mode) + 4 × Rn.
11111	Index, quad words	mode[Rn:Q]	EA (mode) + 8 × Rn.
			"Mode" and "n" are contained within the Index Byte.
			EA(mode) denotes the effective address generated using mode.

Table 2. TI32000 Instuction Set Summary

MOVES

Format	Operation	Operands	Description
4	MOVi	gen,gen	Move a value.
2	MOVQi	short,gen	Extend and move a signed 4-bit constant.
7	MOVMi	gen,gen,disp	Move Multiple: disp bytes (1 to 16).
7	MOVZBW	gen,gen	Move with zero extension.
7	MOVZiD	gen,gen	Move with zero extension.
7	MOVXBW	gen,gen	Move with sign extension.
7	MOVXiD	gen,gen	Move with sign extension.
4	ADDR	gen,gen	Move Effective Address.

INTEGER ARITHMETIC

Format	Operation	Operands	Description
4	ADDi	gen,gen	Add.
2	ADDQi	short,gen	Add signed 4-bit constant.
4	ADDCi	gen,gen	Add with carry.
4	SUBi	gen,gen	Subtract.
4	SUBCi	gen,gen	Subtract with carry (borrow).
6	NEGi	gen,gen	Negate (2's complement).
6	ABSi	gen,gen	Take absolute value.
7	MULi	gen,gen	Multiply
7	QUOi	gen,gen	Divide, rounding toward zero.
7	REMi	gen,gen	Remainder from QUO.
7	DIVi	gen,gen	Divide, rounding down.
7	MODi	gen,gen	Remainder from DIV (Modulus).
7	MEIi	gen,gen	Multiply to Extended Integer.
7	DEIi	gen,gen	Divide Extended Integer.

PACKED DECIMAL (BCD) ARITHMETIC

Format	Operation	Operands	Description
6	ADDPi	gen,gen	Add Packed.
6	SUBPi	gen,gen	Subtract Packed.

INTEGER COMPARISON

Format	Operation	Operands	Description
4	CMPi	gen,gen	Compare.
2	CMPQi	short,gen	Compare to signed 4-bit constant.
7	CMPMi	gen,gen,disp	Compare Multiple: disp bytes (1 to 16).

LOGICAL AND BOOLEAN

Format	Operation	Operands	Description
4	ANDi	gen,gen	Logical AND.
4	ORi	gen,gen	Logical OR.
4	BICi	gen,gen	Clear selected bits.
4	XORi	gen,gen	Logical Exclusive OR.
6	COMi	gen,gen	Complement all bits.
6	NOTi	gen,gen	Boolean complement: LSB only.
2	Scondi	gen	Save condition code (cond) as a Boolean variable of size i.

Table 2. TI32000 Instruction Set Summary (Continued)

SHIFTS

Format	Operation	Operands	Description
6	LSHi	gen,gen	Logical Shift, left or right.
6	ASHi	gengen	Arithmetic Shift, left or right.
6	ROTi	gen,gen	Rotate, left or right.

BITS

Format	Operation	Operands	Description
4	TBITi	gen,gen	Test bit.
6	SBITi	gen,gen	Test and set bit.
6	SBITIi	gen,gen	Test and set bit, interlocked
6	CBITi	gen,gen	Test and clear bit.
6	CBITIi	gen,gen	Test and clear bit, interlocked.
6	IBITi	gen,gen	Test and invert bit.
8	FFSi	gen,gen	Find first set bit

BIT FIELDS

Bit fields are values in memory that are not aligned to byte boundaries. Examples are PACKED arrays and records used in Pascal. "Extract" instructions read and align a bit field. "Insert" instructions write a bit field from an aligned source.

Format	Operation	Operands	Description
8	EXTi	reg,gen,gen,disp	Extract bit field (array oriented).
8	INSi	reg,gen,gen,disp	Insert bit field (array oriented).
7	EXTSi	gen,gen,imm,imm	Extract bit field (short form).
7	INSSi	gen,gen,imm,imm	Insert bit field (short form).
8	CVTP	reg,gen,gen	Convert to Bit Field Pointer.

ARRAYS

Format	Operation	Operands	Description
8	CHECKi	reg,gen,gen	Index bounds check.
8	INDEXi	reg,gen,gen	Recursive indexing step for multiple-dimensional arrays.

STRINGS

String instructions assign specific functions to the Address-Data Registers:

R4 — Comparison Value
R3 — Translation Table Pointer
R2 — String 2 Pointer
R1 — String 1 Pointer
R0 — Limit Count

Options on all strong instructions are:

B (Backward): Decrement string pointers after each step rather than incrementing.
U (Until match): End instruction if String 1 entry matches R4.
W (While match): End instruction if String 1 entry does not match R4.

All string instructions end when R0 decrements to zero.

Table 2. TI32000 Instruction Set Summary (Continued)

Format	Operation	Operands	Descriptions
5	MOVSi	options	Move string 1 to String 2.
	MOVST	options	Move string, translating bytes.
5	CMPSi	options	Compare String 1 to String 2.
	CMPST	options	Compare, translating String 1 bytes.
5	SKPSi	options	Skip over String 1 entries.
	SKPST	options	Skip, translating bytes for Until/While.

JUMPS AND LINKAGE

Format	Operation	Operands	Description
3	JUMP	gen	Jump.
0	BR	disp	Branch (PC Relative).
0	Bcond	disp	Conditional branch.
3	CASEi	gen	Multiway branch.
2	ACBi	short,gen,disp	Add 4-bit constant and branch if non-zero.
3	JSR	gen	Jump to subroutine.
1	BSR	disp	Branch to subroutine.
1	CXP	disp	Call external procedure.
3	CXPD	gen	Call external procedure using descriptor.
1	SVC		Supervisor Call.
1	FLAG		Flag Trap.
1	BPT		Breakpoint Trap.
1	ENTER	[reg list],disp	Save registers and allocate stack frame (Enter Procedure).
1	EXIT	[reg list]	Restore registers and reclaim stack frame (Exit Procedure).
1	RET	disp	Return from subroutine.
1	RXP	disp	Return from external procedure call.
1	RETT	disp	Return from trap. (Privileged)
1	RETI		Return from interrupt. (Privileged)

Table 2. TI32000 Instruction Set Summary (Continued)

CPU REGISTER MANIPULATION

Format	Operation	Operands	Description
1	SAVE	[reg list]	Save Address-Data Registers.
1	RESTORE	[reg list]	Restore Address-Data registers.
2	LPRi	areg,gen	Load Dedicated Register. (Privileged if PSR or INTBASE)
2	SPRi	areg,gen	Store Dedicated Register. (Privileged if PSR or INTBASE)
3	ADJSPi	gen	Adjust Stack Pointer.
3	BISPSRi	gen	Set selected bits in PSR. (Privileged if not Byte length)
3	BICPSRi	gen	Clear selected bits in PSR. (Privileged if not Byte length)
5	SETCFG	[option list]	Set Configuration Register. (Privileged)

FLOATING POINT

Format	Operation	Operands	Description
11	MOVf	gen,gen	Move a Floating Point value.
9	MOVLF	gen,gen	Move and shorten a Long value to Standard.
9	MOVFL	gen,gen	Move and lengthen a Standard value to Long.
9	MOVif	gen,gen	Convert any integer to Standard or Long Floating.
9	ROUNDfi	gen,gen	Convert to integer by rounding.
9	TRUNCfi	gen,gen	Convert to integer by truncating, toward zero.
9	FLOOR fi	gen,gen	Convert to largest integer less than or equal to value.
11	ADDf	gen,gen	Add.
11	SUBf	gen,gen	Subtract.
11	MULf	gen,gen	Multiply.
11	DIVf	gen,gen	Divide.
11	CMPf	gen,gen	Compare.
11	NEGf	gen,gen	Negate.
11	ABSf	gen,gen	Take absolute value.
9	LFSR	gen	Load FSR.
9	SFSR	gen	Store FSR.

MEMORY MANAGEMENT

Format	Operation	Operands	Description
14	LMR	mreg,gen	Load Memory Management Register. (Privileged)
14	SMR	mreg,gen	Store Memory Management Register. (Privileged)
14	RDVAL	gen	Validate address for reading. (Privileged)
14	WRVAL	gen	Validate address for writing. (Privileged)
8	MOVSUi	gen,gen	Move a value from Operating System Space to User Space. (Privileged)
8	MOVUSi	gen,gen	Move a value from User Space to Operating System Space. (Privileged)

Table 2. TI32000 Instruction Set Summary (Concluded)

MISCELLANEOUS

Format	Operation	Operands	Description
1	NOP		No Operation.
1	WAIT		Wait for interrupt.
1	DIA		Diagnose. Single-byte "Branch to Self" for hardware breakpointing. Not for use in programming.

APPLICATION-SPECIFIC COPROCESSOR (ACU)

Format	Operation	Operands	Description
15.5	CCAL0c	gen,gen	ACU Calculate.
15.5	CCAL1c	gen,gen	
15.5	CCAL2c	gen,gen	
15.5	CCAL3c	gen,gen	
15.5	CMOV0c	gen,gen	ACU Move.
15.5	CMOV1c	gen,gen	
15.5	CMOV2c	gen,gen	
15.5	CCMPc	gen,gen	ACU Compare.
15.1	CCV0ci	gen,gen	ACU Convert.
15.1	CCV1ci	gen,gen	
15.1	CCV2ci	gen,gen	
15.1	CCV3ci	gen,gen	
15.1	CCV4DQ	gen,gen	
15.1	CCV5QD	gen,gen	
15.1	LCSR	gen	Load ACU Status Register.
15.1	SCSR	gen	Store ACU Status Register.
15.0	CATST0	gen	ACU Address/Test. (Privileged)
15.0	CATST1	gen	(Privileged)
15.0	LCR	creg,gen	Load ACU Register. (Privileged)
15.0	SCR	creg,gen	Store ACU Register. (Privileged)

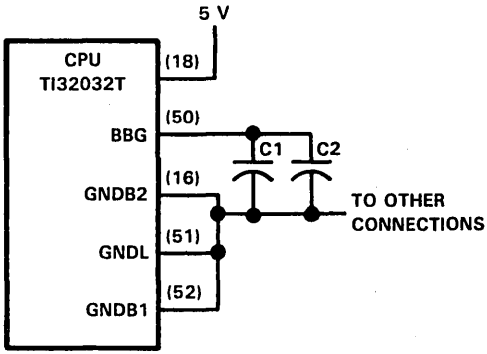


Figure 9. Recommended Supply Connections

3.1.3.2 Clocking

The TI32032T inputs clocking signals from the TI32201 Timing Control Unit(TCU), which presents two nonoverlapping phases of a single clock frequency. These phases are called CLK1 (pin 26) and CLK2 (pin 27). Their relationship to each other is shown in Figure 10.

Each positive edge of CLK1 defines a transition in the timing state (T-State) of the CPU. One T-State represents the execution of one microinstruction within the CPU, and/or one step of an external bus transfer. See the ac timing requirements in the TI32032T data sheet for complete specifications on CLK1 and CLK2.

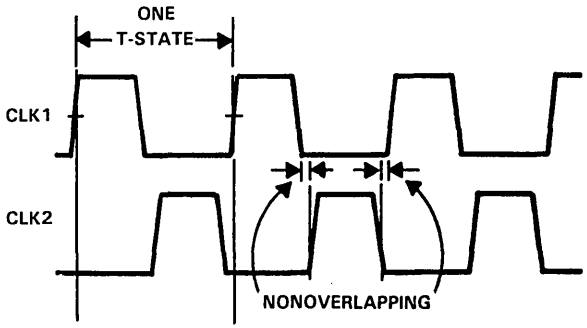


Figure 10. Clock Timing Relationships

Since the TCU presents signals with very fast transitions, it is recommended that the conductors carrying CLK1 and CLK2 be kept as short as possible, and that they not be connected anywhere except from the TCU to the CPU and, if present, the Memory Management Unit (MMU). A TTL Clock signal (CTTL) is provided by the TCU for all other clocking.

3.1.3.3 Resetting

The $\overline{RST}/\overline{ABT}$ pin serves both as a Reset for on-chip logic and as the Abort input for Memory-Managed systems. For its use as the Abort command, see section 3.1.3.5.4.

The CPU may be reset at any time by pulling the $\overline{RST}/\overline{ABT}$ pin low for at least 64 clock cycles. Upon detecting a reset, the CPU terminates instruction processing, resets its internal logic, and clears the Program Counter (PC) and Processor Status Register (PSR) to all zeros.

On application of power, $\overline{RST}/\overline{ABT}$ must be held low for at least 50 μs after V_{CC} is stable. This is to ensure that all on-chip voltages are completely stable before operation. Whenever a Reset is applied, it must remain active for not less than 64 clock cycles. The trailing (positive-going) edge must occur while CLK1 is high, and no later than 10 ns before the CLK1 trailing edge. See Figures 11 and 12.

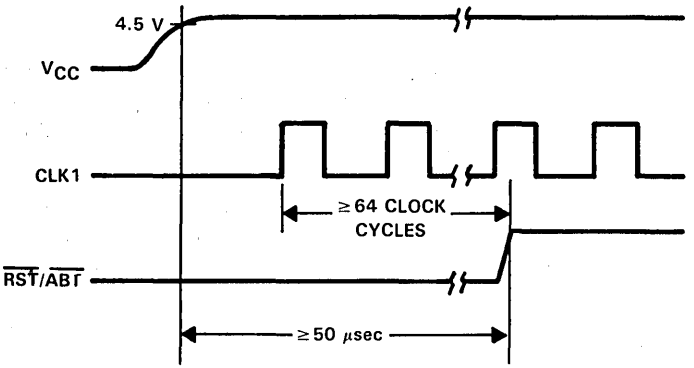


Figure 11. Power-On Reset Requirements

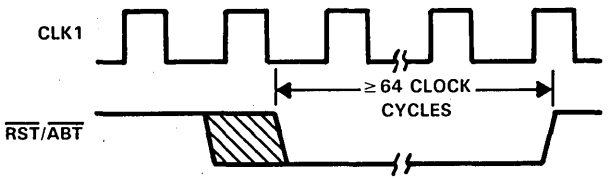


Figure 12. General Reset Timing

The TI32201 Timing Control Unit (TCU) provides circuitry to meet the reset requirements of the TI32032T CPU. Figure 13 shows the recommended connections for a non-Memory-Mananaged system. Figure 14 shows the connections for a Memory-Managed system.

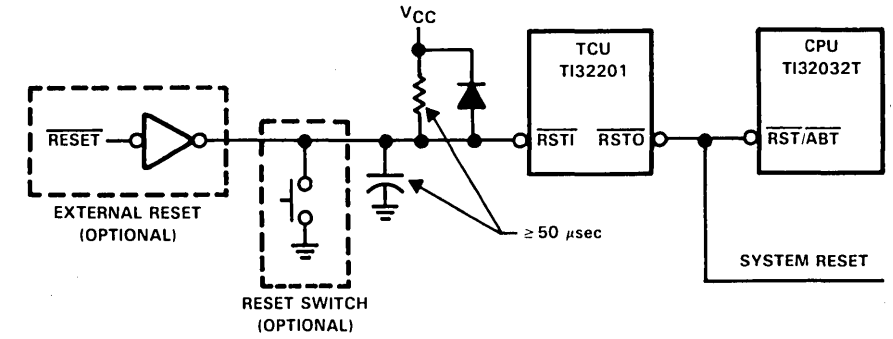


Figure 13. Recommended Reset Connections, Non-Memory-Managed System

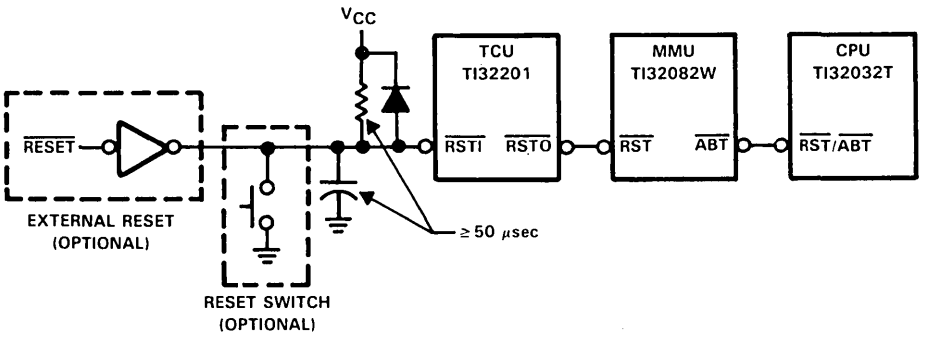


Figure 14. Recommended Reset Connections, Memory-Managed System

3.1.3.4 Bus Cycles

The TI32032T CPU has a strap option that defines the Bus Timing Mode as either with or without Address Translation. For details covering the use of the strap, refer to section 3.1.3.5.

The CPU will perform a bus cycle for one of the following reasons:

1. To write or read data, to or from memory or a peripheral interface device. Peripheral input and output are memory-mapped in the TI32000 family.
2. To fetch instructions into the 8-byte instruction queue. This happens whenever the bus would otherwise be idle and the queue is not already full.
3. To acknowledge an interrupt and allow external circuitry to provide a vector number, or to acknowledge completion of an interrupt service routine.
4. To transfer information to or from a Coprocessor.

In terms of bus timing, cases 1 through 3 above are identical. The only external difference between cases 1 through case 3 is the 4-bit code placed on the Bus Status pins (ST0-ST3). Coprocessor cycles differ in that separate control signals are applied. Refer to section 3.1.3.4.6

The sequence of events in a noncoprocessor bus cycle is shown in Figure 16 for a Read cycle and Figure 17 for a Write cycle. The cases shown assume that the selected memory or interface device is capable of communicating with the CPU at full speed. If it is not, then cycle extension may be requested through the RDY line. Refer to Section 3.1.3.4.1.

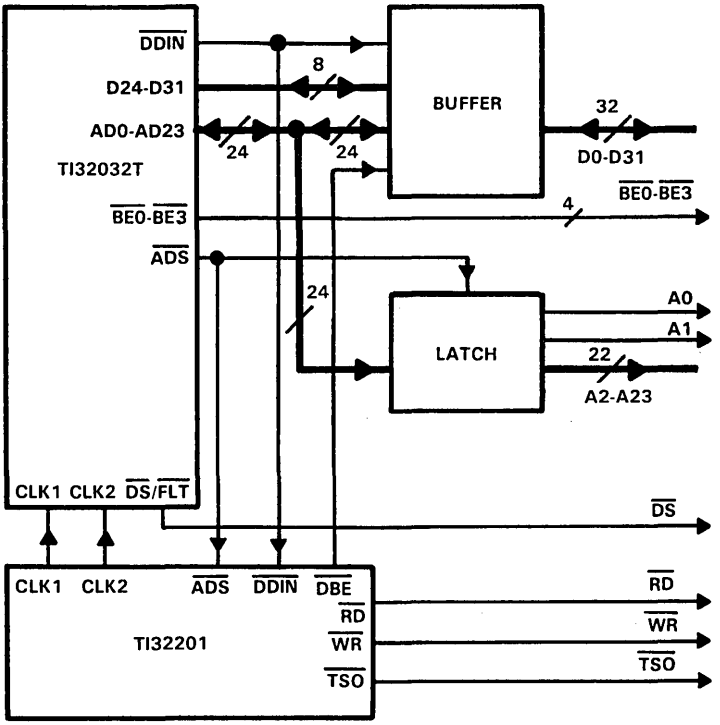


Figure 15. Bus Connections

A full-speed bus cycle is performed in four cycles of CLK1, labeled T1 through T4. Clock cycles not associated with a bus cycle are designated Ti (for "idle").

During T1, the CPU applies an address on pins AD0-AD23. It also provides a low-going pulse on the $\overline{ADS}$ pin, which serves the dual purpose of informing external circuitry that a bus cycle is starting and of providing control to an external latch for demultiplexing Address bits 0-23 from the AD0-AD23 pins. See Figure 15. Also during this time the status signals $\overline{DDIN}$, indicating the direction of the transfer, and $\overline{BE0}$-$\overline{BE3}$, indicating which of the four bus bytes are to be referenced, become valid.

During T2, the CPU switches the Data Bus AD0-AD31 to either accept or present data. It also starts the Data Strobe ($\overline{DS}$), signalling the beginning of the data transfer. Associated signals from the TI32201 Timing Control Unit are also activated at this time: $\overline{RD}$ (Read Strobe) or $\overline{WR}$ (Write Strobe), $\overline{TSO}$ (Timing State Output, indicating that T2 has been reached), and $\overline{DBE}$ (Data Buffer Enable).

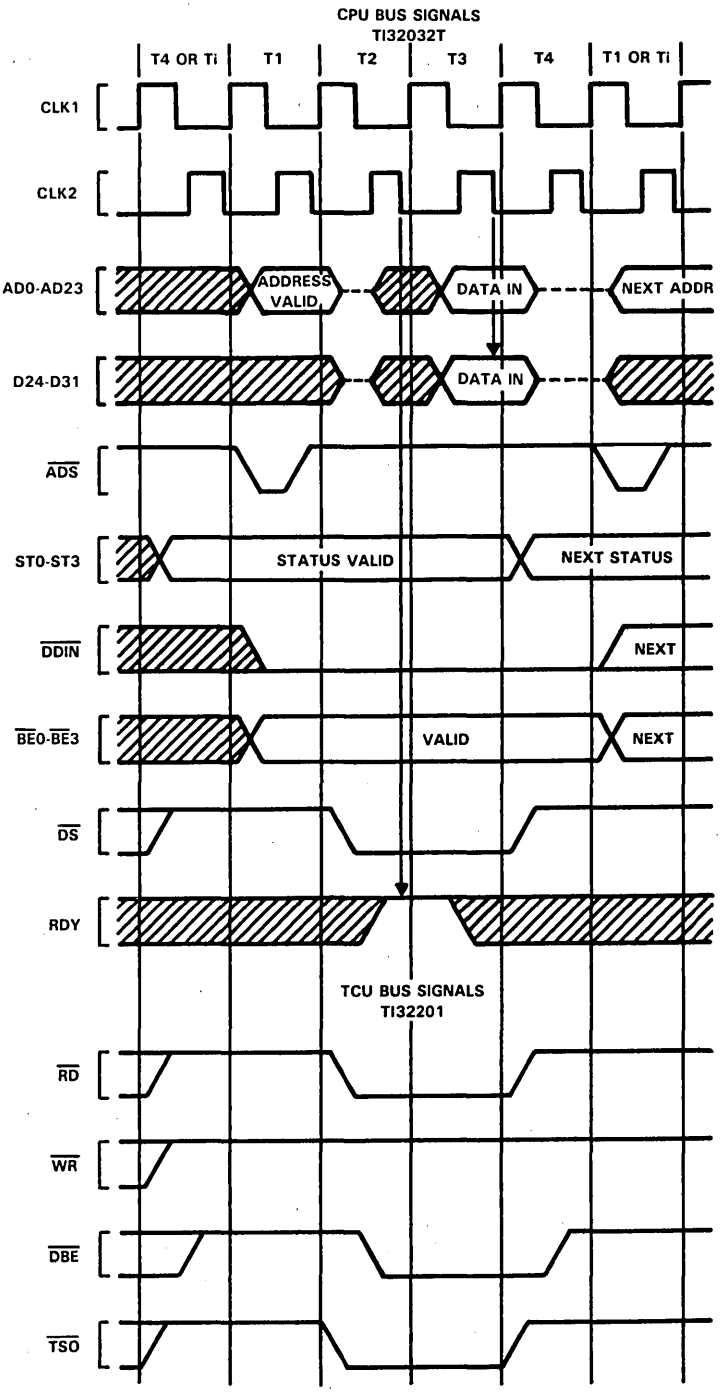


Figure 16. Read Cycle Timing

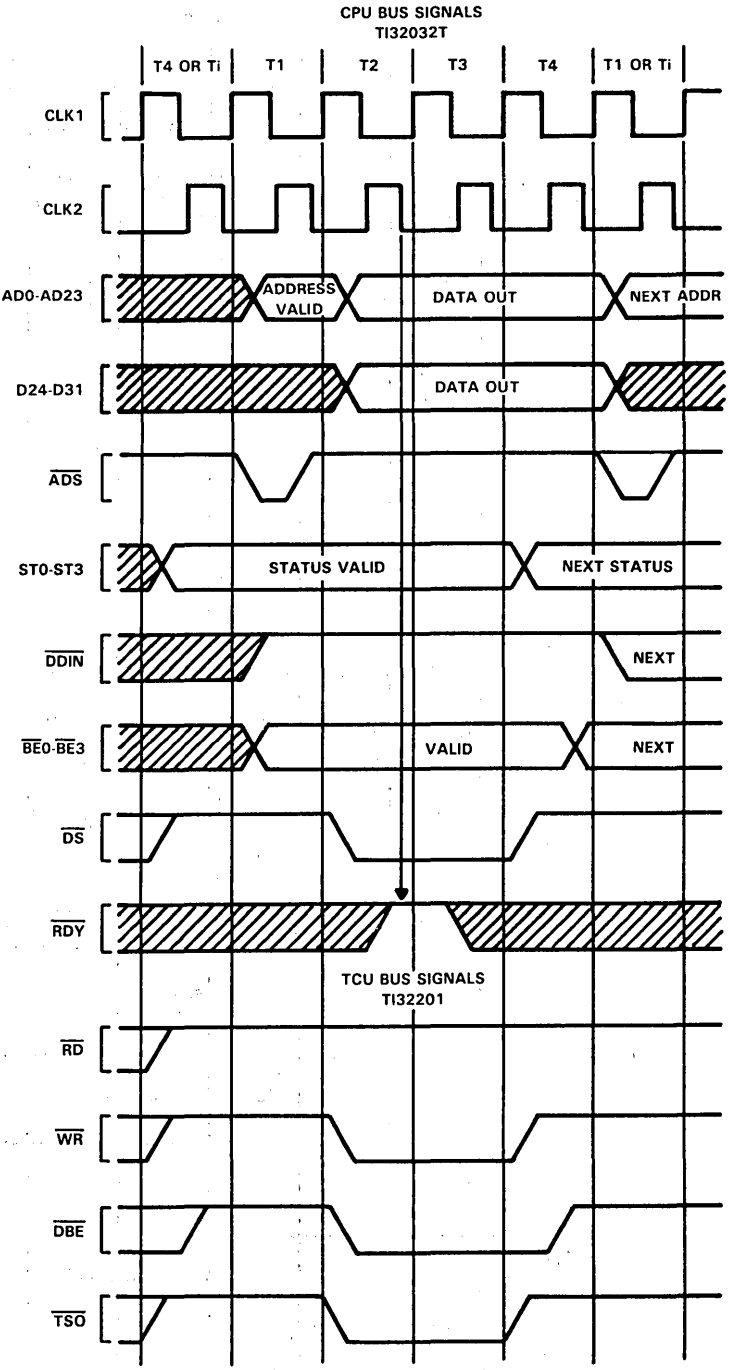


Figure 17. Write Cycle Timing

The T3 state provides for access time requirements, and it occurs at least once in a bus cycle. At the beginning of T3, on the rising edge of the CLK1 clock, the RDY line is sampled to determine whether the bus cycle will be extended (section 3.1.3.4.1).

If the CPU is performing a Read cycle, the Data Bus (AD0-AD31) is sampled at the falling edge of CLK2 in the last T3 state. However, data must be held at least until the beginning of T4. $\overline{DS}$ and $\overline{RD}$ are guaranteed not to go inactive before this point, so the rising edge of either of them may safely be used to disable the device providing the input data.

The T4 state finishes the bus cycle. At the beginning of T4, the $\overline{DS}$, $\overline{RD}$ or $\overline{WR}$, and $\overline{TSO}$ signals go inactive, and on the rising edge of CLK2, $\overline{DBE}$ goes inactive, having provided for necessary data hold times. Data during Write cycles remains valid from the CPU throughout T4. Note that the Bus Status lines (ST0-ST3) change at the beginning of T4, anticipating the following bus cycle (if any).

3.1.3.4.1 Cycle Extension

To allow sufficient strobe widths and access times for any speed of memory or peripheral device, the TI32032T provides for extension of a bus cycle. Any type of bus cycle except a coprocessor cycle can be extended.

In Figures 16 and 17, note that during T3 all bus control signals from the CPU and TCU are flat. Therefore, a bus cycle can be cleanly extended by causing the T3 state to be repeated. This is the purpose of the Ready (RDY) pin.

At the end of T2, on the falling edge of CLK2, the RDY line is sampled by the CPU. If RDY is high, the next T-states will be T3 and T4, ending the bus cycle. If RDY is low, an additional T3 state will be inserted after the initial T3 state and the RDY line will again be sampled on the falling edge of CLK2. Each additional T3 state after the first is referred to as a "Wait State". See Figure 18.

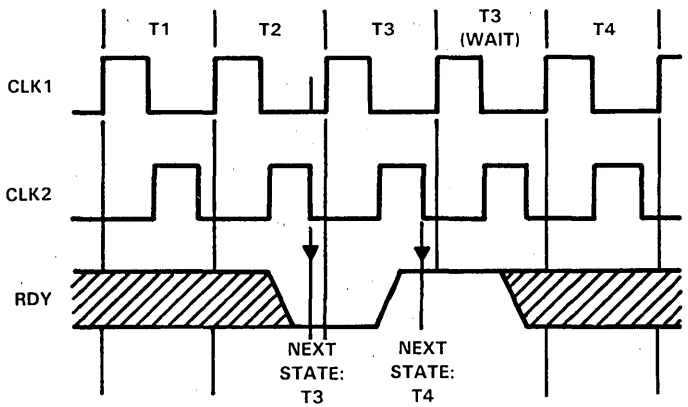


Figure 18. RDY Pin Timing

The RDY pin is driven by the TI32201 Timing Control Unit, which applies wait-states to the CPU as requested on three sets of pins:

1. $\overline{\text{CWAIT}}$ (Continuous Wait), which holds the CPU in wait-states until removed.
2. $\overline{\text{WAIT1}}$, $\overline{\text{WAIT2}}$, $\overline{\text{WAIT4}}$, $\overline{\text{WAIT8}}$ (Collectively $\overline{\text{WAITn}}$), which may be given a 4-bit binary value requesting a specific number of wait-states from 0-15.
3. $\overline{\text{PER}}$ (Peripheral), which inserts five additional wait-states and causes the TCU to reshape the $\overline{\text{RD}}$ and $\overline{\text{WR}}$ strobes. This provides the setup and hold times required by most MOS peripheral interface devices.

Combinations of these various Wait requests are both legal and useful. For details on their use, see section 3.6.

Figure 19 illustrates a typical Read cycle, with two wait-states requested through the TCU $\overline{\text{WAITn}}$ pins.

3.1.3.4.2 Bus Status

The TI32032T CPU presents 4 bits of Bus Status information on pins ST0-ST3. The various combinations on these pins indicate why the CPU is performing a bus cycle, or, if it is idle on the bus, then why it is idle.

Referring to Figures 16 and 17, note that Bus Status leads the corresponding Bus Cycle, going valid one clock cycle before T1, and changing to the next state at T4. This allows the system designer to fully decode the Bus Status and, if desired, latch the decoded signals before $\overline{\text{ADS}}$ initiates the Bus Cycle.

The Bus Status pins are interpreted as a 4-bit value, with ST0 the least significant bit. Their values decode as follows:

0000 The bus is idle because the CPU does not yet need access to the bus.

0001 The bus is idle because the CPU is executing the Wait instruction.

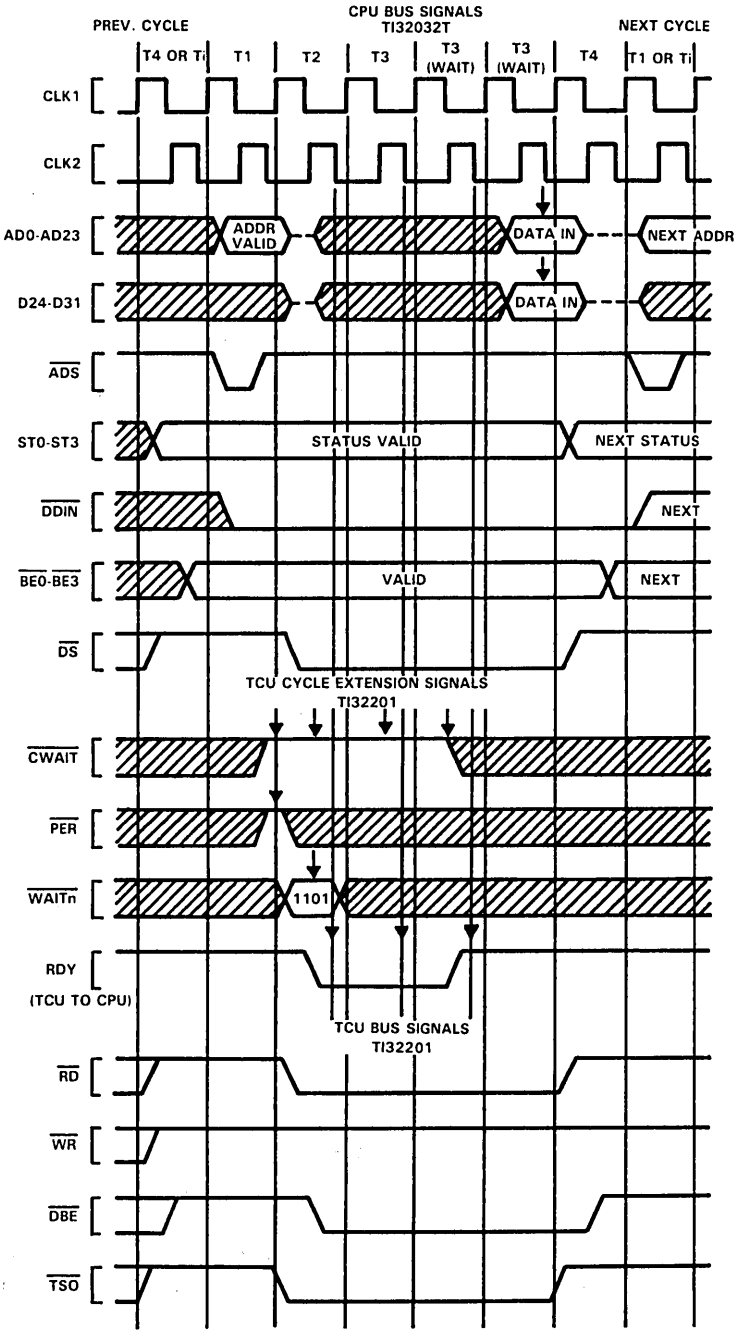
0010 (Reserved for future use.)

0011 The bus is idle because the CPU is waiting for a coprocessor to complete an instruction.

0100 Master Interrupt Acknowledge. The CPU is performing a read cycle.To acknowledge receipt of a Non-maskable Interrupt (on $\overline{\text{NMI}}$) it will read from address FFFF00_{16} but will ignore any data provided. To acknowledge receipt of a Maskable Interrupt (on $\overline{\text{INT}}$) it will read from address FFFF00_{16}, expecting a vector number to be provided from the Master TI32202 Interrupt Control Unit (ICU). If the vectoring mode selected by the last SETCFG instruction was nonvectored, then the CPU will ignore the value it has read and will use a default vector instead, having assumed that no TI32202 is present (section 3.1.3.4.5).

0101 Cascaded Interrupt Acknowledge. The CPU is reading a vector number from a Cascaded TI32202 Interrupt Control Unit. The address provided is the address of the TI32202 Hardware Vector register (section 3.1.3.4.5).

0110 Master End of Interrupt. The CPU is performing a Read cycle to indicate that it is executing a Return from Interrupt (RETI) instruction (section 3.1.3.4.5).



NOTE: Arrows on $\overline{\text{CWAIT}}$, $\overline{\text{PER}}$, $\overline{\text{WAITn}}$ indicate points at which the TCU samples. Arrows on AD0-AD15 and RDY indicate points at which the CPU samples.

Figure 19. Extended Cycle Example

0111 Cascaded End of Interrupt. The CPU is reading from a Cascaded Interrupt Control Unit to indicate that it is returning (through RETI) from an interrupt service routine requested by that unit (section 3.1.3.4.5).

1000 Sequential Instruction Fetch. The CPU is reading the next sequential word from the instruction stream into the Instruction Queue. It will do so whenever the bus would otherwise be idle and the queue is not already full.

1001 Nonsequential Instruction Fetch. The CPU is performing the first fetch of instruction code after the Instruction Queue is purged. This will occur as a result of any jump or branch, or any interrupt or trap, or execution of certain instructions.

1010 Data Transfer. The CPU is reading or writing an operand of instruction.

1011 Read RMW Operand. The CPU is reading an operand which will subsequently be modified and rewritten. If memory protection circuitry would not allow the following write cycle, it must abort this cycle.

1100 Read for Effective Address Calculation. The CPU is reading information from memory in order to determine the Effective Address of an operand. This will occur whenever an instruction uses the Memory Relative or External addressing mode.

1101 Transfer Coprocessor Operand. The CPU is either transferring an instruction operand to or from a coprocessor, or it is issuing the Operation Word of a coprocessor instruction (section 3.1.3.9.1)

1110 Read Coprocessor Status. The CPU is reading a Status Word from a coprocessor. This occurs after the coprocessor has signaled completion of an instruction. The transferred word tells the CPU whether a trap should be taken, and in some instructions it presents new values for the CPU Processor Status Register bits N,Z,L, or F (section 3.1.3.9.1).

1111 Broadcast Coprocessor ID. The CPU is initiating the execution of a coprocessor instruction. The ID Byte (first byte of the instruction) is sent to all Coprocessors, one of which will recognize it. From this point the CPU is communicating with only one coprocessor (section 3.1.3.9.1)

3.1.3.4.3 Data Access Sequences

The 24-bit address provided by the TI32032T is a byte address; that is, it uniquely identifies one of up to 16,777,216 eight-bit memory locations. An important feature of the TI32032T is that the presence of a 32-bit data bus imposes no restrictions on the data alignment; any data item, regardless of size, may be placed starting at any memory address. TheTI32032T provides special control signals, Byte Enable ($\overline{BE0}$-$\overline{BE3}$) which facilitate individual byte accessing on a 32-bit bus.

Memory is organized as four 8-bit banks, each bank receiving the double-word address (A2-A23)) in parallel. One bank, connected to Data Bus pins AD0 — AD7 is enabled when $\overline{BE0}$ is low. The second bank, connected to data bus pins AD8 — AD15, is enabled when $\overline{BE1}$ is low. The third and fourth banks are enabled by $\overline{BE2}$ and $\overline{BE3}$, repectively. See Figure 20.

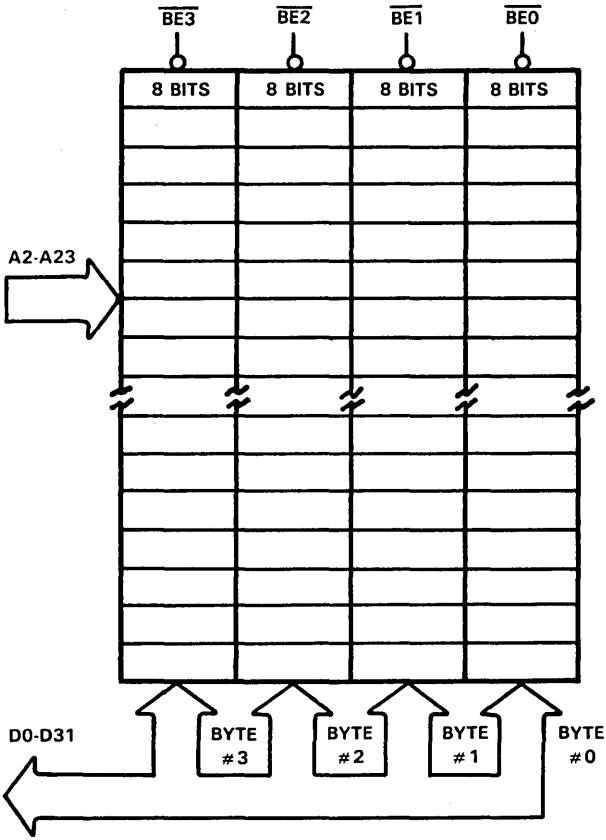


Figure 20. Memory Interface

Since operands do not need to be aligned with respect to the double-word bus access performed by the CPU, a given double-word access can contain one, two, three, or four bytes of the operand being addressed; these bytes can begin at various positions, as determined by A1 and A0.Table 3 lists the 10 resulting access types.

Accesses of operands requiring more than one bus cycle are performed sequentially, with no idle T-States separating them. The number of buscycles required to transfer an operand depends on its size and its alignment. Table 4 lists the bus cycles performed for each situation.

Bit Accesses. The Bit Instructions perform byte accesses to the byte containing the designated bit. The Test and Set Bit instruction (SBIT), for example, reads a byte, alters it, and rewrites it, having changed the contents of the one bit.

Bit Field Accesses. An access to a Bit Field in memory always generates a Double-Word transfer at the address containing the least significant bit of the field. The Double Word is read by an Extract Instruction; an Insert instruction reads a Double Word, modifies it, and rewrites it.

Table 3. Bus Access Types

TYPE	BYTES ACCESSED	A1,A0	$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$
1	1	00	1	1	1	0
2	1	01	1	1	0	1
3	1	10	1	0	1	1
4	1	11	0	1	1	1
5	2	00	1	1	0	0
6	2	01	1	0	0	1
7	2	10	0	0	1	1
8	3	00	1	0	0	0
9	3	01	0	0	0	1
10	4	00	0	0	0	0

Extending Multiply Accesses. The Extending Multiply Instruction (MEI) will return a result which is twice the size in bytes of the operands it reads. If the multiplicand is in memory, the most significant half of the result is written first (at the higher address), then the least significant half. This is done in order to support retry if this instruction is aborted.

3.1.3.4.4 Instruction Fetches

Instructions for the TI32032T CPU are "prefetched"; that is, they are input before being needed into the next available entry of the eight-byte Instruction Queue. The CPU performs two types of Instruction Fetch cycles: Sequential and Nonsequential. These can be distinguished from each other by their differing status combinations on pins ST0-ST3. (section 3.1.3.4.2)

A Sequential Fetch will be performed by the CPU whenever the Data Bus would otherwise be idle and the Instruction Queue is not currently full. Sequential Fetches are always type 8 Read cycles (Table 3).

A Nonsequential Fetch occurs as a result of any break in the normally sequential flow of a program. Any jump or branch instruction, a trap or an interrupt will cause the next Instruction Fetch cycle to be Nonsequential. In addition, certain instructions flush the instruction queue, causing the next instruction fetch to display Nonsequential status. Only the first bus cycle after a break displays Nonsequential status, and that cycle depends on the destination address.

3.1.3.4.5 Interrupt Control Cycles

Activating the $\overline{INT}$ or $\overline{NMI}$ pin on the CPU will initiate one or more bus cycles whose purpose is interrupt control rather than the transfer of instructions or data. Execution of the Return from Interrupt instruction (RETI) will also cause Interrupt Control bus cycles. These differ from instruction or data transfers only in the status presented on pins ST0-ST3. All Interrupt Control cycles are single-byte Read cycles.

This section describes only the Interrupt Control sequences associated with each interrupt and with the return from its service routine. For full details of the TI32032T interrupt structure, see section 3.1.3.8.

Table 4. Access Sequences

CYCLE	TYPE	ADDRESS	$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$	DATA BUS BYTE 3	BYTE 2	BYTE 1	BYTE 0
A. Word at address ending with 11									BYTE 1 | BYTE 0	← A
1.	4	A	0	1	1	1	Byte 0	X	X	X
2.	1	A + 1	1	1	1	0	X	X	X	Byte 1
B. Double word at address ending with 01							BYTE 3 | BYTE 2 | BYTE 1 | BYTE 0			← A
1.	9	A	0	0	0	1	Byte 2	Byte 1	Byte 0	X
2.	1	A + 3	1	1	1	0	X	X	X	Byte 3
C. Double word at address ending with 10							BYTE 3 | BYTE 2 | BYTE 1 | BYTE 0			← A
1.	7	A	0	0	1	1	Byte 1	Byte 0	X	X
2.	5	A + 2	1	1	0	0	X	X	Byte 3	Byte 2
D. Double word at address ending with 11							BYTE 3 | BYTE 2 | BYTE 1 | BYTE 0			← A
1.	4	A	0	1	1	1	Byte 0	X	X	X
2.	8	A + 1	1	0	0	0	X	Byte 3	Byte 2	Byte 1
E. Quad word at address ending with 00			BYTE 7 | BYTE 6 | BYTE 5 | BYTE 4 | BYTE 3 | BYTE 2 | BYTE 1 | BYTE 0							← A
1.	10	A	0	0	0	0	Byte 3	Byte 2	Byte 1	Byte 0
Other bus cycles (instruction prefetch or coprocessor) can occur here.										
2.	10	A + 4	0	0	0	0	Byte 7	Byte 6	Byte 5	Byte 4
F. Quad word at address ending with 01			BYTE 7 | BYTE 6 | BYTE 5 | BYTE 4 | BYTE 3 | BYTE 2 | BYTE 1 | BYTE 0							← A
1.	9	A	0	0	0	1	Byte 2	Byte 1	Byte 0	X
2.	1	A + 3	1	1	1	0	X	X	X	Byte 3
Other bus cycles (instruction prefetch or coprocessor) can occur here.										
3.	9	A + 4	0	0	0	1	Byte 6	Byte 5	Byte 4	X
4.	1	A + 7	1	1	1	0	X	X	X	Byte 7

Table 4. Access Sequences (Concluded)

CYCLE	TYPE	ADDRESS	$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$	DATA BUS BYTE 3	BYTE 2	BYTE 1	BYTE 0
G. Quad word at address ending with 10				BYTE 7	BYTE 6	BYTE 5	BYTE 4	BYTE 3	BYTE 2	BYTE 1	BYTE 0	← A
1.	7	A	0	0	1	1	Byte 1	Byte 0	X	X
2.	5	A + 2	1	1	0	0	X	X	Byte 3	Byte 2
Other bus cycles (instruction prefetch or coprocessor) can occur here.										
3.	7	A + 4	0	0	1	1	Byte 5	Byte 4	X	X
4.	5	A + 6	1	1	0	0	X	X	Byte 7	Byte 6
H. Quad word at address ending with 11				BYTE 7	BYTE 6	BYTE 5	BYTE 4	BYTE 3	BYTE 2	BYTE 1	BYTE 0	← A
1.	4	A	0	1	1	1	Byte 0	X	X	X
2.	8	A + 1	1	0	0	0	X	Byte 3	Byte 2	Byte 1
Other bus cycles (instruction prefetch or coprocessor) can occur here.										
1.	4	A + 4	0	1	1	1	Byte 4	X	X	X
2.	8	A + 5	1	0	0	0	X	Byte 7	Byte 6	Byte 5

X = Don't Care

Table 5. Interrupt Sequences

CYCLE	STATUS	ADDRESS	$\overline{DDIN}$	$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$	DATA BUS Byte 3	Byte 2	Byte 1	Byte 0
A. Nonmaskable Interrupt Control Sequences											
Interrupt Acknowledge											
1	0100	$FFFF00_{16}$	0	1	1	1	0	X	X	X	X
Interrupt Return											
None: Performed through Return from Trap (RETT) instruction.											
B. Nonvectored Interrupt Control Sequences											
Interrupt Acknowledge											
1	0100	$FFFE00_{16}$	0	1	1	1	0	X	X	X	X
Interrupt Return											
1	0110	$FFFE00_{16}$	0	1	1	1	0	X	X	X	X
C. Vectored Interrupt Sequences: Noncascaded											
Interrupt Acknowledge											
1	0100	$FFFE00_{16}$	0	1	1	1	0	X	X	X	Vector: Range: 0-127
Interrupt Return											
1	0110	$FFFE00_{16}$	0	1	1	1	0	X	X	X	Vector: Same as in Previous Int. Ack. Cycle

Table 5. Interrupt Sequences (Concluded)

CYCLE	STATUS	ADDRESS	$\overline{DDIN}$	$\overline{BE3}$	$\overline{BE2}$	$\overline{BE1}$	$\overline{BE0}$	DATA BUS Byte 3	Byte 2	Byte 1	Byte 0
D. Vectored Interrupt Sequences: Cascaded											
Interrupt Acknowledge											
1	0100	$FFFE00_{16}$	0	1	1	1	0	X	X	X	Cascade Index: range −16 to −1
(The CPU here uses the Cascade Index to find the Cascade Address.)											
2	0101	Cascade Address	0	See Note				Vector, range 9-255; on appropriate byte of data bus.			
Interrupt Return											
1	0110	$FFFE00_{16}$	0	1	1	1	0	X	X	X	Cascade Index: Same as in previous Int. Ack. Cycle
(The CPU here uses the Cascade Index to find the Cascade Address)											
2	0111	Cascade Address	0	See Note				X	X	X	X

X = Don't Care

Note: $\overline{BE0}$-$\overline{BE3}$ signals will be activated according to the cascaded ICU address. The cycle type can be 1, 2, 3 or 4, when reading the interrupt vector. the vector value can be in the range 0-255.

3.1.3.4.6 Coprocessor Communication

In addition to its use as the Address Translation strap (section 3.1.3.5.1), the $\overline{AT}/\overline{SPC}$ pin is used as the data strobe for coprocessor transfers.

In this role, it is referred to as Coprocessor Control ($\overline{SPC}$). In a Coprocessor bus cycle, data is transferred on the Data Bus (AD0-AD15), and the least significant two bits of CPU cycle status (ST0-ST1) are monitored by each coprocessor in order to determine the type of transfer being performed. $\overline{SPC}$ is bidirectional, but is driven by the CPU during all coprocessor bus cycles (section 3.1.3.9) (Figure 21).

Coprocessor Bus Cycles. A coprocessor bus cycle always takes exactly two clock cycles, labeled T1 and T4 (see Figures 22 and 23). During a Read cycle, $\overline{SPC}$ is activated at T1, data is sampled at T4, and $\overline{SPC}$ is removed. The Cycle Status pins lead the cycle by one clock period, and are sampled at the leading edge of $\overline{SPC}$. During a write cycle, the CPU applies data and activates $\overline{SPC}$ at T1, removing $\overline{SPC}$ at T4. The coprocessor latches status on the leading edge of $\overline{SPC}$ and latches data on the trailing edge.

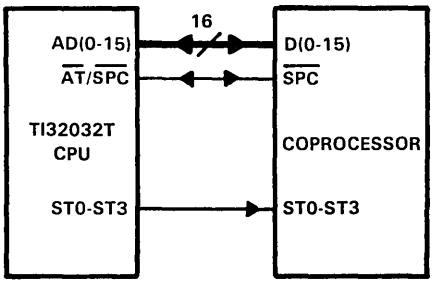


Figure 21. Coprocessor Connections

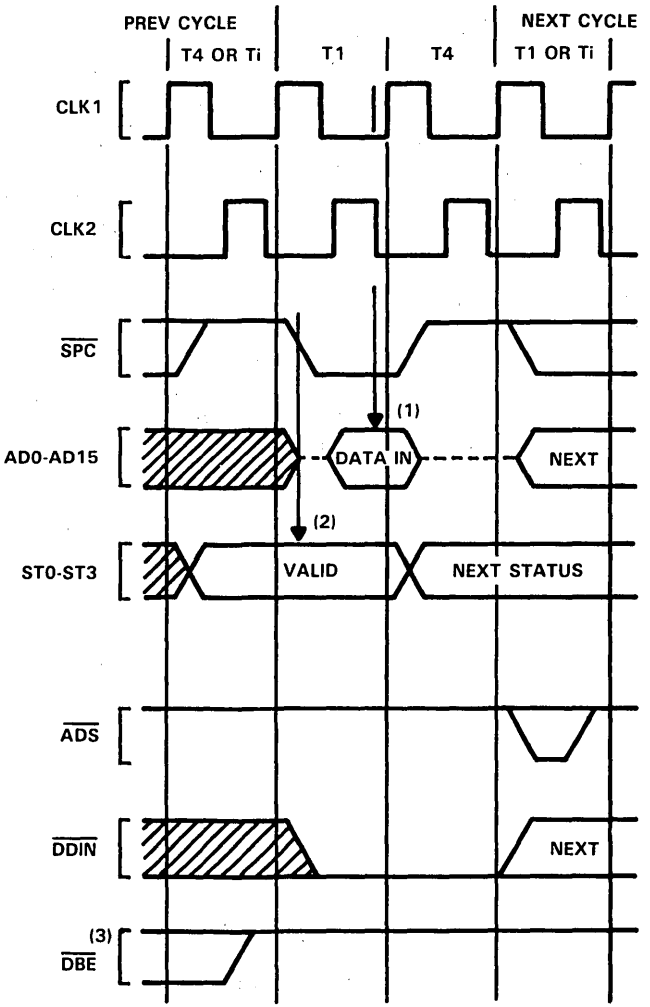
Since the CPU does not pulse the Address Strobe ($\overline{ADS}$), no bus signals are generated by the TI32201 Timing Control Unit. The direction of a transfer is determined by the sequence ("protocol") established by the instruction under execution; but the CPU indicates the direction on the $\overline{DDIN}$ pin for hardware debugging purposes.

Operand Transfer Sequences. A coprocessor operand is transferred in one or more coprocessor bus cycles. A Byte operand is transferred on the least significant byte of the Data Bus (AD0-AD7), and a Word operand is transferred on bits AD0-AD15. A Double-Word is transferred in a consecutive pair of bus cycles, least significant word first. A Quad-Word is transferred in two pairs of Coprocessor cycles, with other bus cycles possibly occurring between them. The word order is from least signficant word to most significant word.

Note that the TI32032T uses only the two least significant bytes of the data bus for coprocessor cycles. This is to maintain compatibility with existing coprocessors.

3.1.3.5 Memory-Management Option

The TI32032T CPU, in conjunction with the TI32082W Memory Management Unit (MMU), provides full support for address translation, memory protection, and memory allocation techniques up to and including Demand-Paged Virtual Memory.

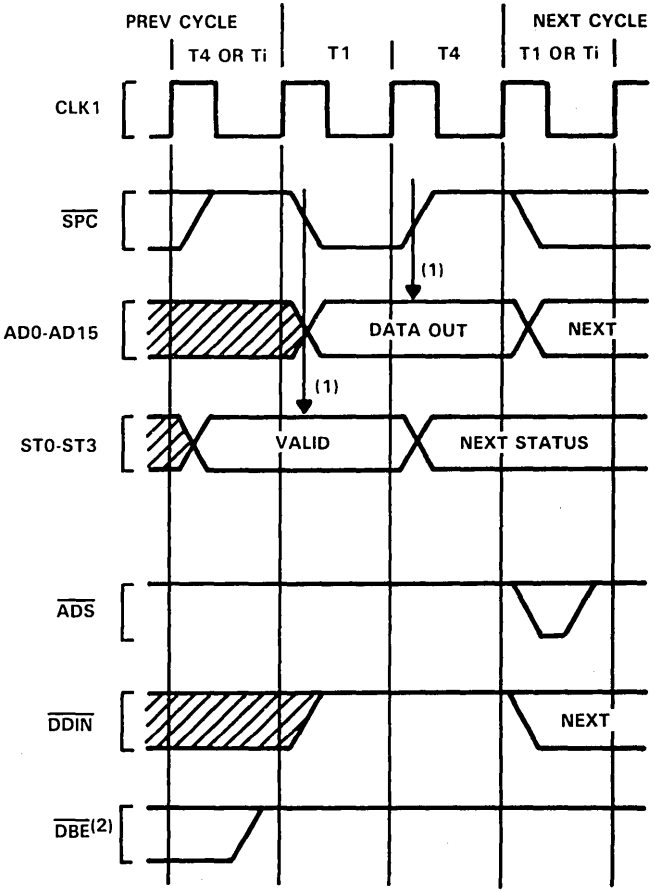


NOTES: 1. CPU samples Data Bus here.
2. Coprocessor samples CPU Status here.
3. $\overline{DBE}$ and all other TI32201 TCU bus signals remain inactive because no $\overline{ADS}$ pulse is received from the CPU.

Figure 22. CPU Read from Coprocessor

3.1.3.5.1 Address-Translation Strap

The Bus Interface Control section of the TI32032T CPU has two bus timing modes; with or without address-translation. The mode of operation is selected by the CPU by sampling the $\overline{AT}/\overline{SPC}$ (Address Translation/Coprocessor Control) pin on the rising edge of the family Reset (RST) pulse. If $\overline{AT}/\overline{SPC}$ is sampled as high, the bus timing is as previously described insection 3.1.3.4. If it is sampled as low, two changes occur:



NOTES: 1. Arrows indicate points at which the Coprocessor samples.
2. DBE, being provided by the TI32201 TCU, remains inactive due to the fact that no pulse is presented on ADS. TCU signals RD, WR and TSO also remain inactive.

Figure 23. CPU Write to Coprocessor

1. An extra clock cycle, Tmmu, is inserted into all bus cycles except coprocessor transfers.
2. The DS/FLT pin changes in function from a Data Strobe output (DS) to a Float-Command input (FLT).

The TI32082 MMU will itself pull the CPU AT/SPC pin low when it is reset. In non-Memory Managed systems, this pin should be pulled up to V_{CC} through a 10-kΩ resistor.

Note that the Address Translation strap does not specifically declare the presence of a TI32082W MMU, but only the presence of external address translation circuitry. MMU instructions will still trap as being undefined unless the SETCFG (Set Configuration) instruction is executed to declare the MMU instruction set valid.

Figures 24 and 25 illustrate the CPU activity during a read cycle and a write cycle in Address Translation mode. The additional T-State, Tmmu, is inserted between T1 and T2. During this time the CPU places AD0-AD23 into the 3-state (high-impedance state) mode, allowing the MMU to assert the translated address and issue the physical address strobe $\overline{PAV}$. T2 through T4 of the cycle are identical to their counterparts without Address Translation. Note that in order for the TI32082W MMU to operate correctly it must be set to the TI32032T mode by strapping A24 to ground during reset.

In this mode the bus lines AD16-AD23 are floated after the MMU address has been latched, since they are used by the CPU to transfer data. Figures 26 and 27 show a read cycle and a write cycle as generated by the TI32032T/TI32082W/TI32201

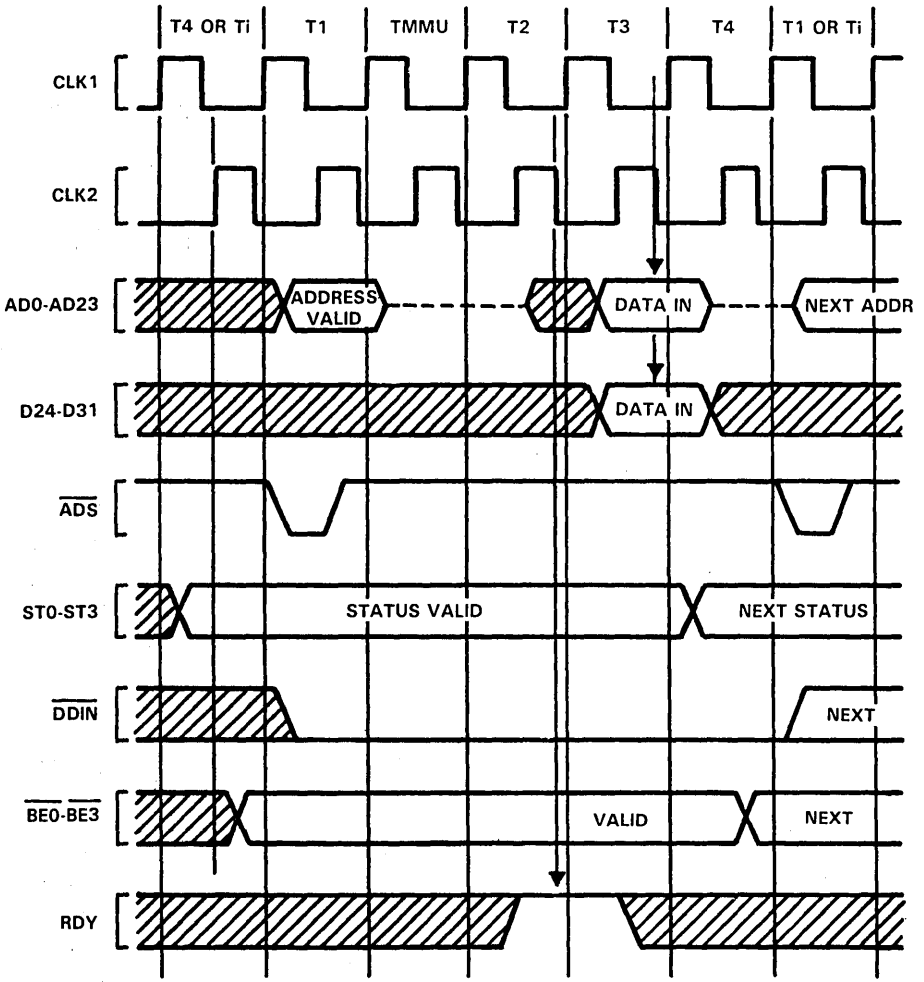


Figure 24. Read Cycle with Address Translation (CPU Action)

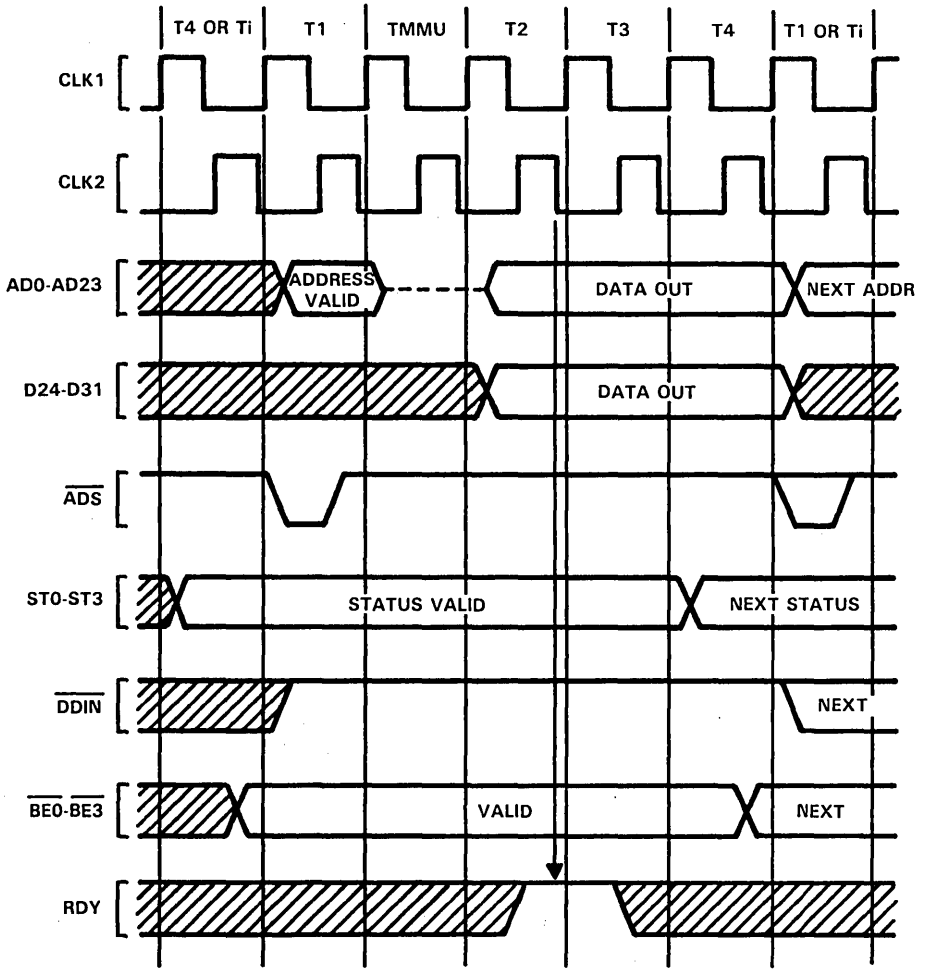


Figure 25. Write Cycle with Address Translation (CPU Action)

group. Note that with the CPU $\overline{ADS}$ signal going to the MMU, and with the MMU $\overline{PAV}$ signal substituting for $\overline{ADS}$ everywhere else, Tmmu through T4 look exactly like T1 through T4 in a non-Memory-Managed system. For the connection diagram, see Figure 28.

3.1.3.5.3 The $\overline{FLT}$ (Float) Pin

In Address Translation mode, the $\overline{DS}/\overline{FLT}$ pin is treated as the input command $\overline{FLT}$ (Float). Activating $\overline{FLT}$ during Tmmu causes the CPU to wait longer than Tmmu for address translation and validation. This feature is used occasionally by the TI32082 MMU in order to update its internal translation cache from page tables in memory, or to update certain status bits within them.

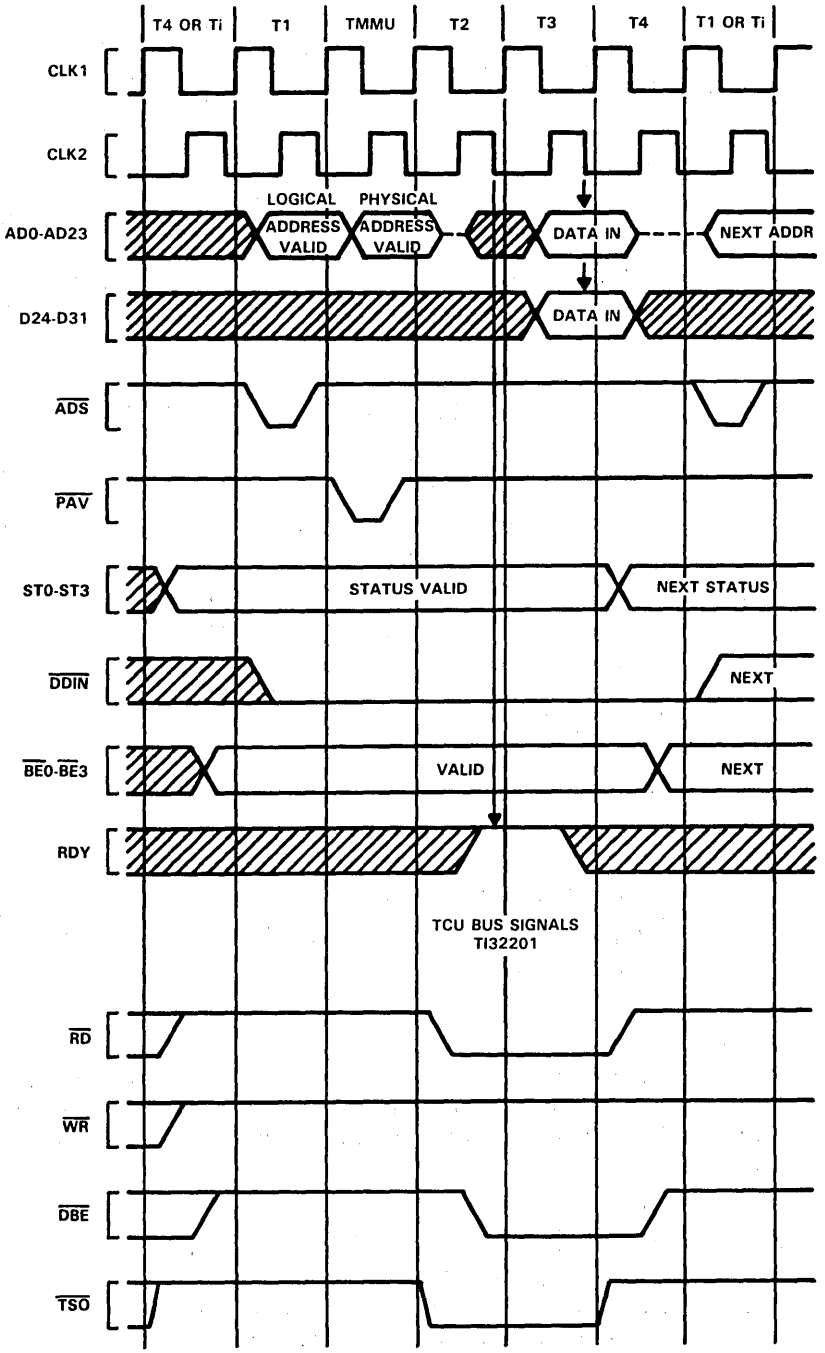


Figure 26. Memory-Managed Read Cycle

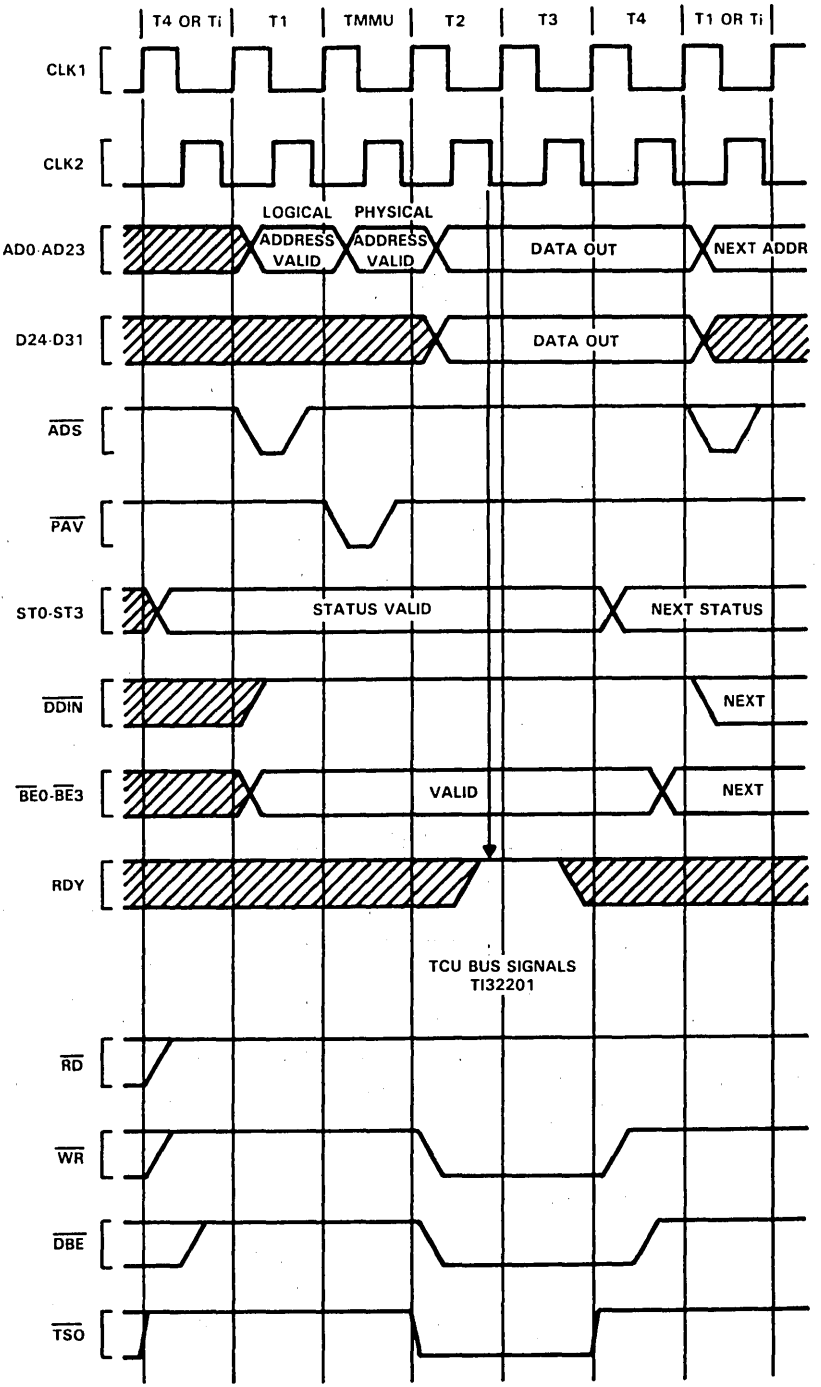


Figure 27. Memory-Managed Write Cycle

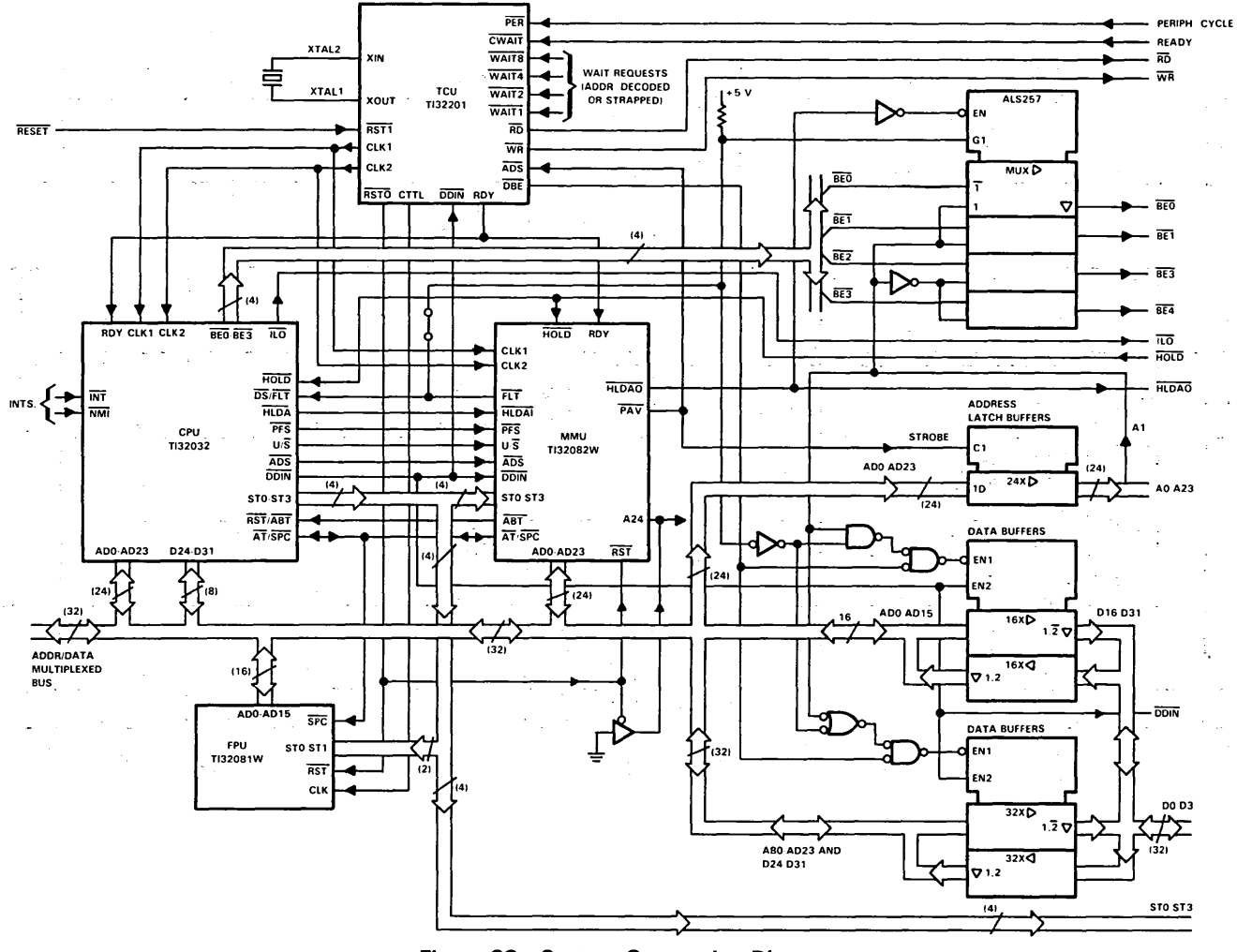


Figure 28. System Connection Diagram

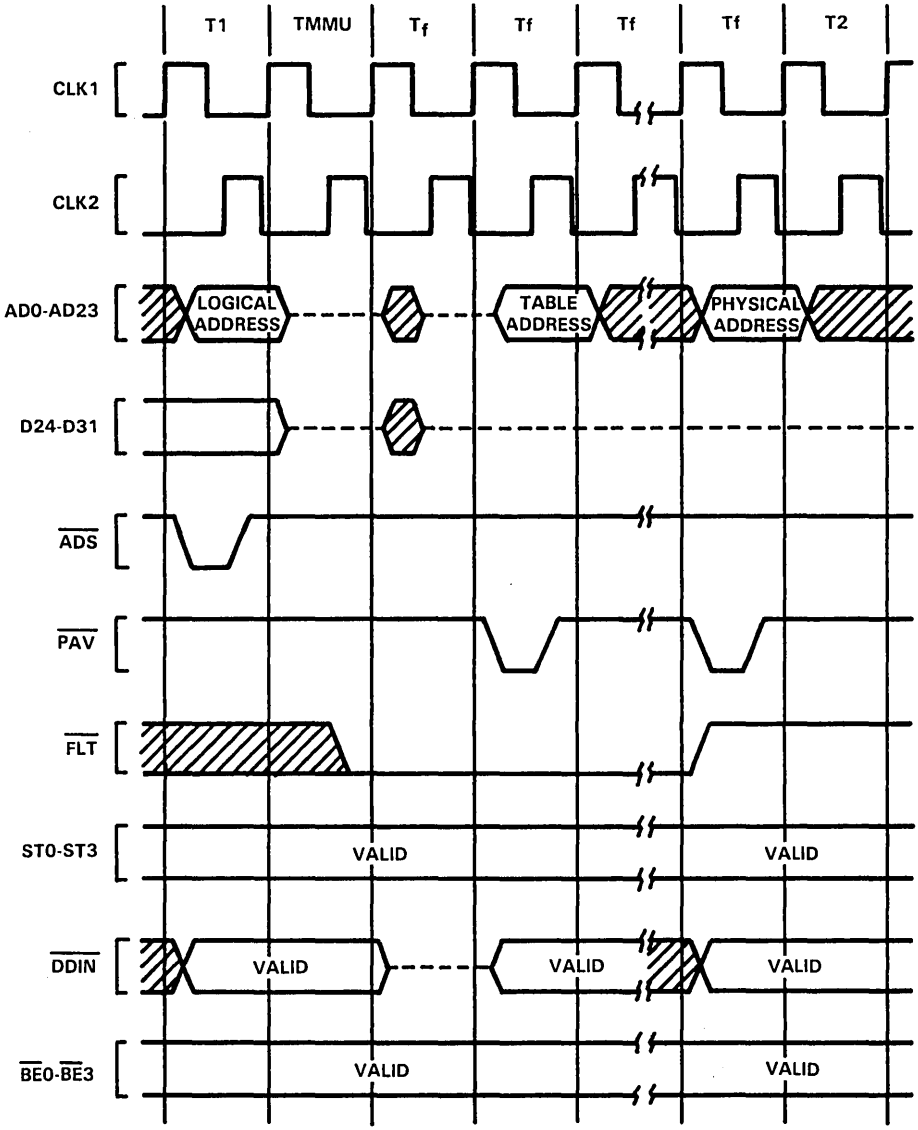


Figure 29. $\overline{\text{FLT}}$ Float Command Timing

Figure 29 shows the effects of $\overline{\text{FLT}}$. Upon sampling $\overline{\text{FLT}}$ low late in Tmmu, the CPU enters idle T-States (Tf) during which it:

1. Sets AD0-AD23, D24-D31, and $\overline{\text{DDIN}}$ to the 3-state (high-impedance) condition (Floating).
2. Suspends further internal processing of the current instruction. This ensures that the current instruction remains abortable with retry. (See $\overline{\text{RST}}$/$\overline{\text{ABT}}$ description, section 3.1.3.5.4)

Note that the AD0-AD23 pins may be briefly asserted during the first idle T-State. The above conditions remain in effect until $\overline{FLT}$ again goes high.

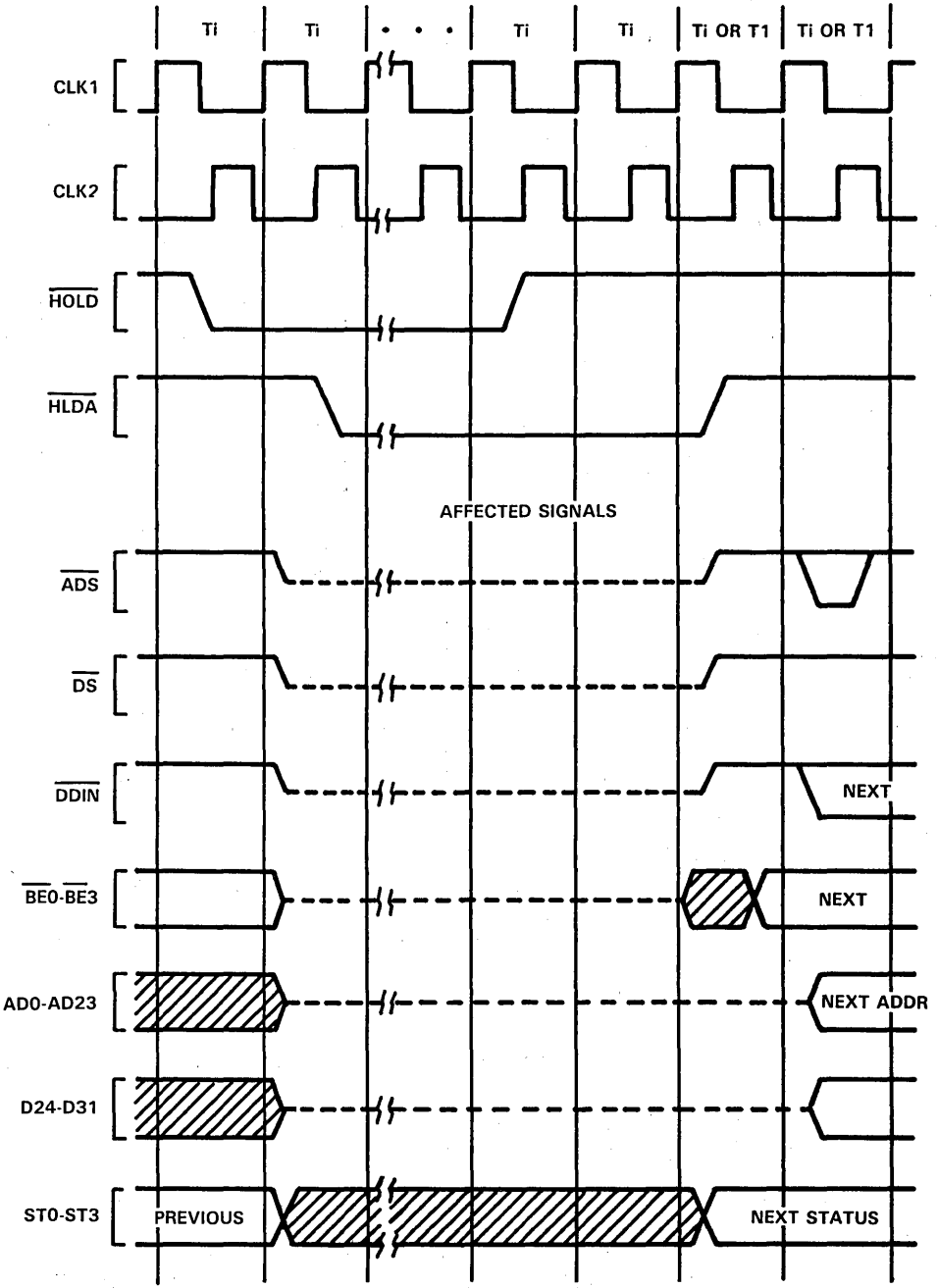


Figure 30. $\overline{HOLD}$ Timing, Bus Inititally Idle

3.1.3.5.4 Aborting Bus Cycles

The $\overline{RST}/\overline{ABT}$ pin, apart from its reset function (section 3.1.3.3), also serves as the means to "abort", or cancel, a bus cycle and the instruction,if any, which initiated it. An Abort request is distinguished from a Reset in that the $\overline{RST}/\overline{ABT}$ pin is held active for only one clock cycle.If $\overline{RST}/\overline{ABT}$ is pulled low during Tmmu or Tf, this signals that the cycle must be aborted. The CPU itself will enter T2 and then Ti, thereby terminating the cycle. Since it is the MMU $\overline{PAV}$ signal which triggers a physical cycle, the rest of the system remains unaware that a cycle was even started.

The TI32082W MMU will abort a bus cycle for either of two reasons:

1. The CPU is attempting to access a logical address which is not currently resident in physical memory. The referenced page must be brought into physical memory from mass storage to make it accessible to the CPU.
2. The CPU is attempting to perform an access which is not allowed due to the protection level assigned to that page.

When a bus cycle is aborted by the MMU, the instruction which caused it to occur is also aborted in such a manner that it is guaranteed to be reexecutable later. The information that is changed irrecoverably by such a partly executed instruction does not affect its reexecution.

The Abort Interrupt. Upon aborting an instruction, the CPU immediately performs an interrupt through the ABT vector in the Interrupt Table. The Return Address pushed on the Interrupt Stack is the address of the aborted instruction, such that a Return from Trap (RETT) instruction will automatically retry it.

The one exception to this sequence occurs if the aborted bus cycle was an instruction prefetch. If so, it is not yet certain that the aborted prefetch code is to be executed. Instead of causing an interrupt, the CPU only aborts the bus cycle, and stops prefetching. If the information in the instruction Queue runs out, meaning that the instruction will actually be executed, the ABT interrupt will occur, in effect aborting the instruction that was being fetched.

Hardware Considerations. In order to guarantee instruction retry, certain rules must be followed in applying an Abort to the CPU. These rules are followed by the TI32082W MMU.

1. If $\overline{FLT}$ has not been applied to the CPU, the Abort pulse must occur during or before Tmmu.
2. If $\overline{FLT}$ has been applied to the CPU, the Abort pulse must be applied before the T-State in which $\overline{FLT}$ goes inactive. The CPU will not actually respond to the Abort command until $\overline{FLT}$ is removed.
3. The Write half of a Read-Modify-Write operand access may not be aborted. The CPU guarantees that this will never be necessary for Memory Management functions by applying a special RMW status (Status Code 1011) during the Read half of the access. When the CPU presents RMW status, that cycle must be aborted if it would be illegal to write to any of the accessed addresses.

If $\overline{RST}/\overline{ABT}$ is pulsed at any time other than as indicated above, it will abort either the instruction currently under execution or the next instruction and will act as a very high-priority interrupt. However,the program which was running at the time is not guaranteed recoverable.

3.1.3.6 Bus Access Control

The TI32032T CPU has the capability of relinquishing its access to the bus upon request from a DMA device or another CPU. This capability is implemented on the $\overline{HOLD}$ (Hold Request) and $\overline{HLDA}$ (Hold Acknowledge) pins. By asserting $\overline{HOLD}$ low, an external device requests access to the bus. On receipt of $\overline{HLDA}$ from the CPU, the device may perform bus cycles, as the CPU at this point has set AD0-AD23, D24-D31, $\overline{ADS}$, $\overline{DDIN}$, and $\overline{BE3}$-$\overline{BE0}$ pins to the 3-state condition. To return control of the bus to the CPU, the device sets $\overline{HOLD}$ inactive, and the CPU acknowledges return of the bus by setting $\overline{HLDA}$ inactive.

How quickly the CPU releases the bus depends on whether it is idle on the bus at the time the $\overline{HOLD}$ request is made, as the CPU must always complete the current bus cycle. Figure 30 shows the timing sequence when the CPU is idle. In this case, the CPU grants the bus during the immediately following clock cycle. Figure 31 shows the sequence if the CPU is using the bus at the time that the $\overline{HOLD}$ request is made. If the request is made during or before the clock cycle shown (two clock cycles before T4), the CPU will release the bus during the clock cycle following T4. If the request occurs closer to T4, the CPU may already have decided to initiate another bus cycle. In that case it will not grant the bus until the next T4 state. Note that this situation will also occur if the CPU is idle on the bus but has initiated a bus cycle internally.

In Memory-Managed systems, the $\overline{HLDA}$ signal is connected in a daisy-chain through the TI32082W, such that the MMU can release the bus if it is using it.

3.1.3.7 Instruction Status

In addition to the four bits of Bus Cycle Status ($\overline{ST0}$-$\overline{ST3}$), the TI32032TCPU also presents Instruction Status information on three separate pins.These pins differ from ST0-ST3 in that they are synchronous to the CPU's internal instruction execution section rather than to its bus interface section.

$\overline{PFS}$ (Program Flow Status) is pulsed low as each instruction begins execution. It is intended for debugging purposes, and is used that way by the TI32082W MMU.

U/$\overline{S}$ originates from the U bit of the Processor Status Register, and indicates whether the CPU is currently running in User or Operating System mode. It is sampled by the MMU for mapping, protection, and debugging purposes. Although it is not synchronous to bus cycles, there are guarantees on its validity during any given bus cycle.

($\overline{ILO}$) (Interlocked Operation) is activated during a Set Bit Interlocked (SBITI) or (CBITI) Clear Bit, Interlocked instruction. It is made available to external bus arbitration circuitry in order to allow these instructions to implement the semaphore primitive operations for multiprocessor communication and resource sharing. As with the U/$\overline{S}$ pin, there are guarantees on it is validity during the operand accesses performed by the instructions.

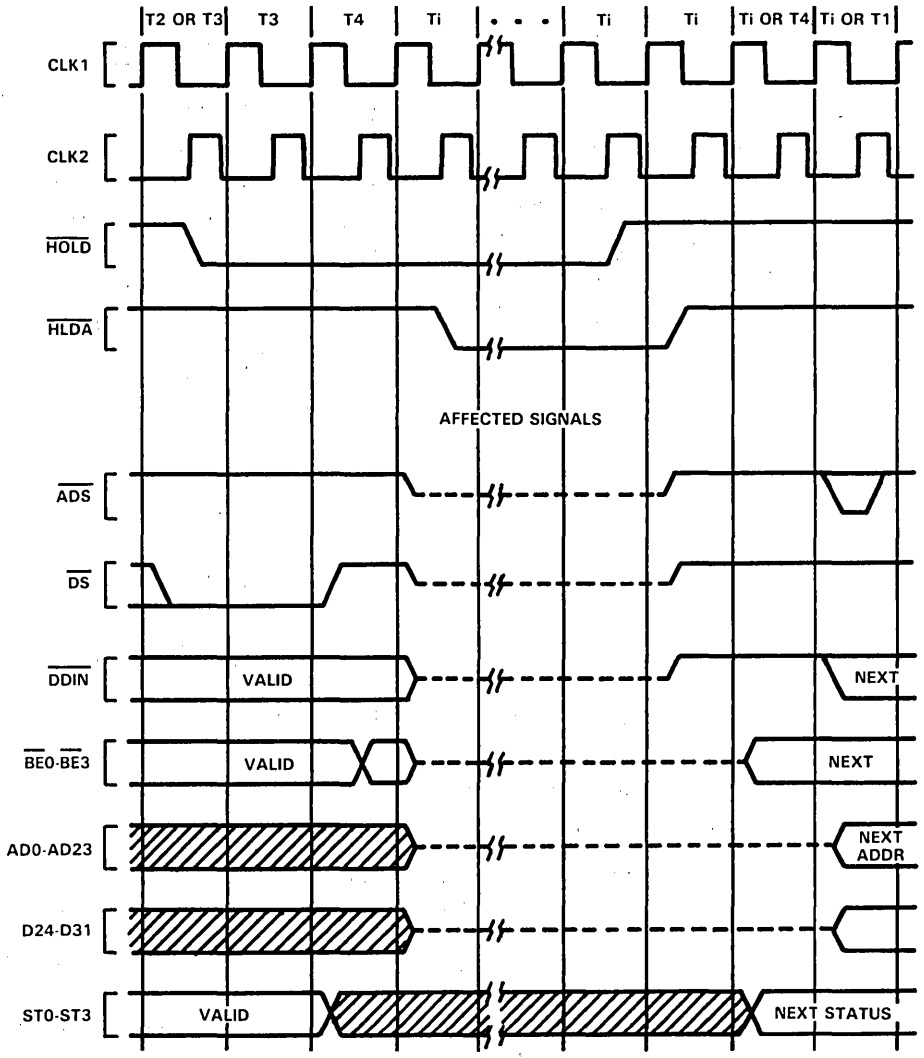


Figure 31. HOLD Timing, Bus Initially Not Idle

3.1.3.8 TI32032T Interrupt Structure

1. INT, on which maskable interrupts may be requested
2. NMI, on which nonmaskable interrupts may be requested, and
3. RST/ABT, which may be used to abort a bus cycle and any associated instruction. It generates an interrupt request if an instruction was aborted (section 3.1.3.5.4).

In addition, there is a set of internally generated "traps" which cause interrupt service to be performed as a result of exceptional conditions (e.g., attempted division by zero) or of specific instructions whose purpose is to cause a trap to occur (e.g., the Supervisor Call instruction).

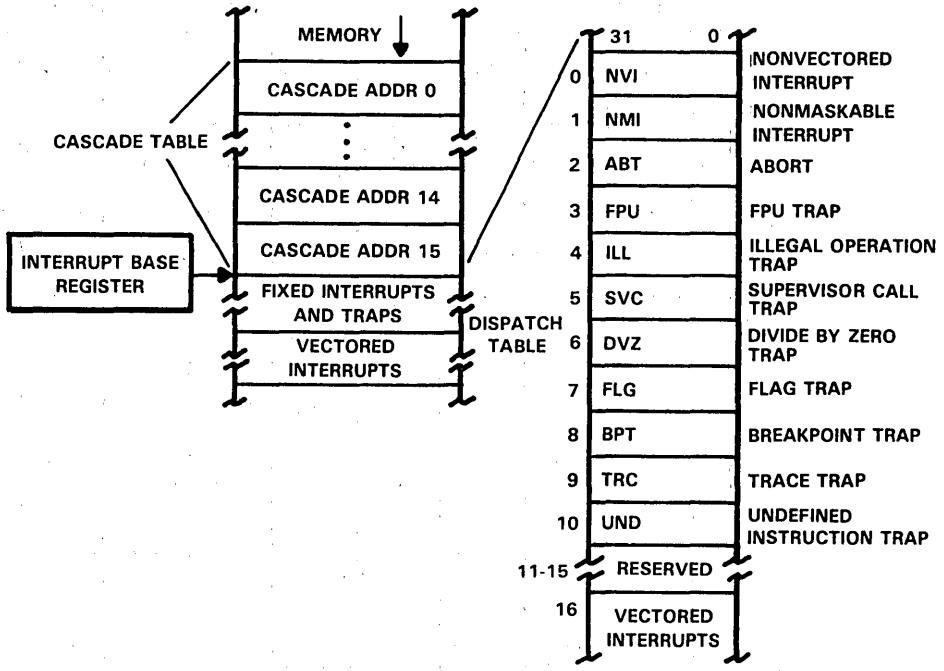


Figure 32. Interrupt Dispatch and Cascade Tables

3.1.3.8.1 General Interrupt/Trap Sequence

Upon receipt of an interrupt or trap request, the CPU goes through four major steps:

1. Adjustment of Registers. Depending on the source of the interrupt or trap, the CPU may restore and/or adjust the contents of the Program Counter (PC), the Processor Status Register (PSR), and the currently selected Stack Pointer (SP). A copy of the PSR is made, and the PSR is then set to reflect Operating System Mode and selection of the Interrupt Stack.
2. Saving Processor Status. The PSR copy is pushed onto the Interrupt Stack as a 16-bit quantity.
3. Vector Acquisition. A vector is either obtained from the Data Bus or is supplied by default.
4. Service Call. The Vector is used as an index into the Interrupt Dispatch Table, whose base address is taken from the CPU Interrupt Base (INTBASE) Register. See Figure 32. A 32-bit External Procedure Call is read from the table entry, and an External Procedure Call is performed using it. The MOD Register (16 bits) and Program Counter (32 bits) are pushed on the Interrupt Stack.

This process is illustrated in Figure 33, from the viewpoint of the programmer.

Full sequences of events in processing interrupts and traps may be found in section 3.1.3.8.7.

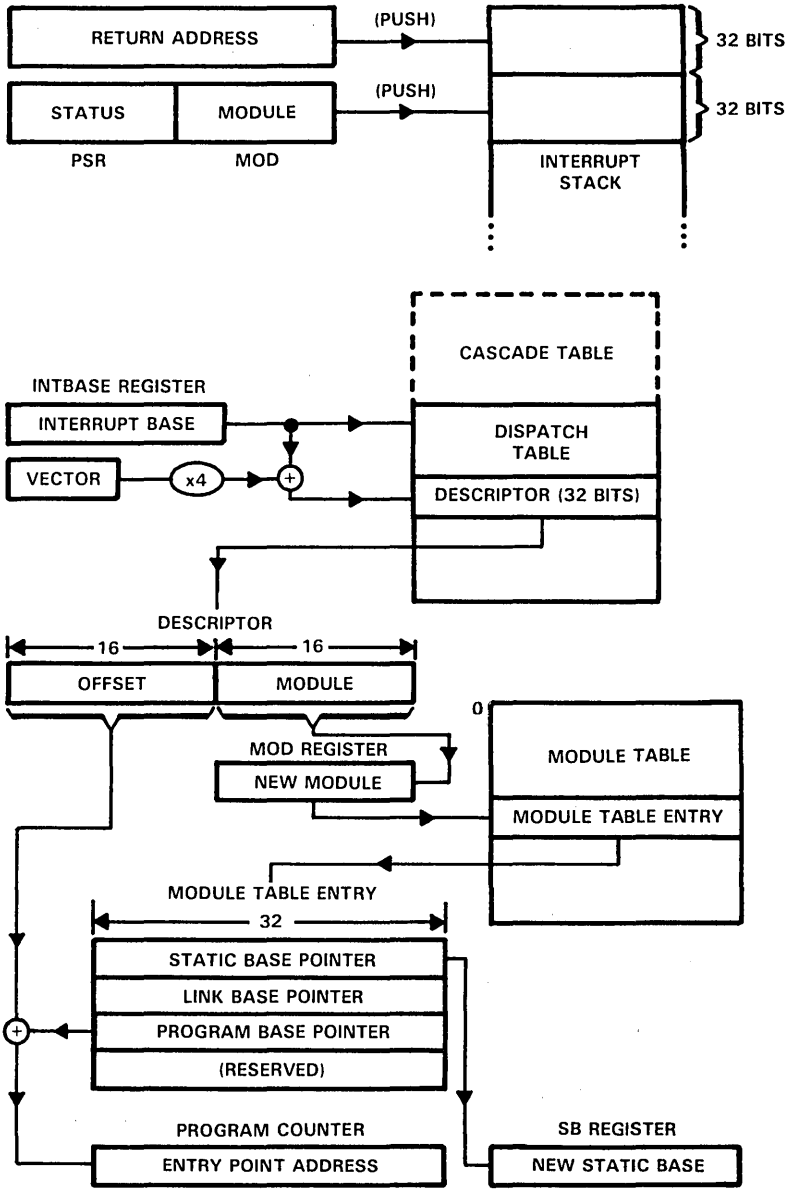


Figure 33. Interrupt/Trap Service Routine Calling Sequence

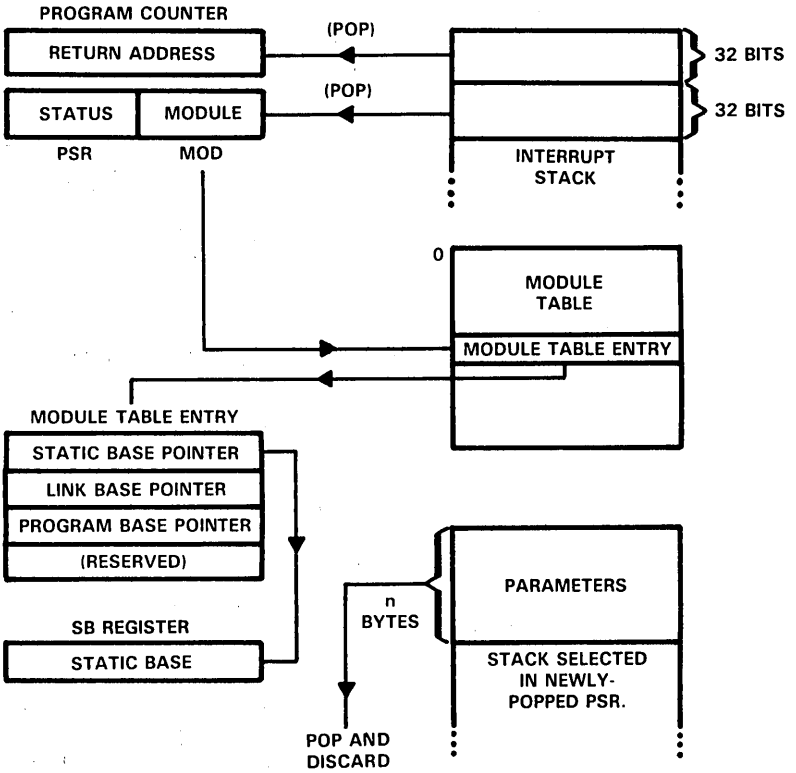


Figure 34. Return from Trap (RETTn) Instruction Flow

3.1.3.8.2 Interrupt/Trap Return

To return to an interrupted program, one of two instructions is used.The RETT (Return from Trap) instruction (Figure 34) restores the PSR, MOD, PC, and SB registers to their previous contents and, since traps are often used deliberately as a call mechanism for Operating System Mode procedures, it also discards a specified number of bytes from the original stack as surplus parameter space. RETT is used to return from any trap or interrupt except the Maskable Interrupt. For this, the RETI (Return from interrupt) instruction is used, which also informs any external Interrupt Control Units that interrupt service has completed. Since interrupts are generally asynchronous external events, RETI does not pop parameters. See Figure 35.

3.1.3.8.3 Maskable Interrupts ($\overline{\text{INT}}$ pin)

The $\overline{\text{INT}}$ pin is a level-sensitive input. A continuous low level is allowed for generating multiple interrupt requests. The input is maskable, and is therefore enabled to generate interrupt requests only while the Processor Status Register I bit is set. The I bit is

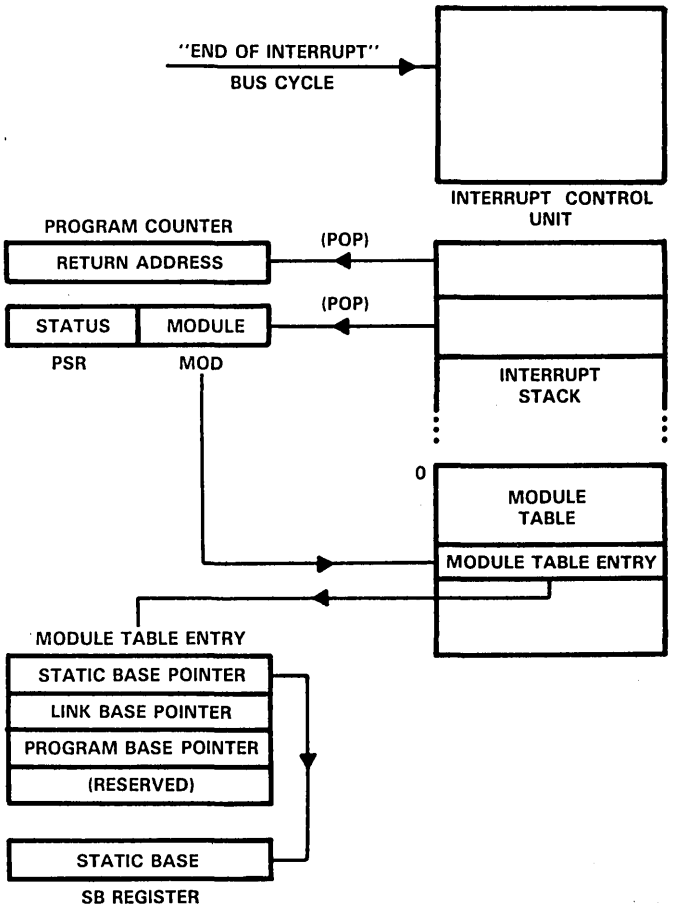


Figure 35. Return from Interrupt (RETI) Instruction Flow

automatically cleared during service of an $\overline{INT}$, $\overline{NMI}$, or Abort request, and is restored to its original setting upon return from the interrupt service routine via the RETT or RETI instruction.

The $\overline{INT}$ pin may be configured via the SETCFG instruction as either Nonvectored (CFG Register bit I = 0) or Vectored (CFG Register bit I = 1).

Nonvectored Mode. In the Nonvectored mode, an interrupt request on the $\overline{INT}$ pin will cause an Interrupt Acknowledge bus cycle, but the CPU will ignore any value read from the bus and use instead a default vector of zero. This mode is useful for small systems in which hardware interrupt prioritization is unnecessary.

Vectored Mode: Noncascaded Case. In the Vectored mode, the CPU uses a TI32202W Interrupt Control Unit (ICU) to prioritize up to 16 interrupt requests. See Figure 36. Upon receipt of an interrupt request on the $\overline{INT}$ pin, the CPU performs an "Interrupt Acknowledge, Master" bus cycle (section 3.1.3.4.2) reading a vector value from the

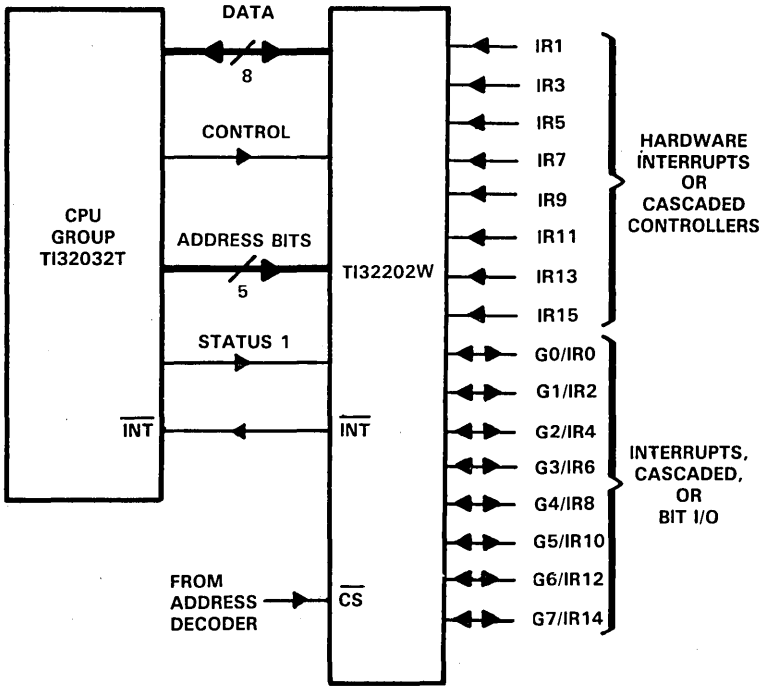


Figure 36. Interrupt Control Unit Connections (16 Levels)

low-order byte of the Data Bus. This vector is then used as an index into the Dispatch Table in order to find the External Procedure Descriptor for the proper interrupt service procedure. The service procedure eventually returns via the Return from Interrupt (RETI) instruction, which performs an End of Interrupt bus cycle, informing the ICU that it may reprioritize any interrupt requests still pending. The ICU provides the vector number again, which the CPU uses to determine whether it also needs to inform a Cascaded ICU (see below).

In a system with only one ICU (16 levels of interrupt), the vectors provided must be in the range of 0 through 127; that is, they must be positive numbers in eight bits. By providing a negative vector number, an ICU flags the interrupt source as being a Cascaded ICU (see below).

Vectored Mode: Cascaded Case. In order to allow up to 256 levels of interrupt, provision is made both in the CPU and in the TI32202W ICU to transparently support cascading. Figure 37 shows a typical cascaded configuration. Note that the Interrupt output from the Cascaded ICU goes to an Interrupt Request input of the Master ICU, which is the only ICU which drives the CPU $\overline{INT}$ pin.

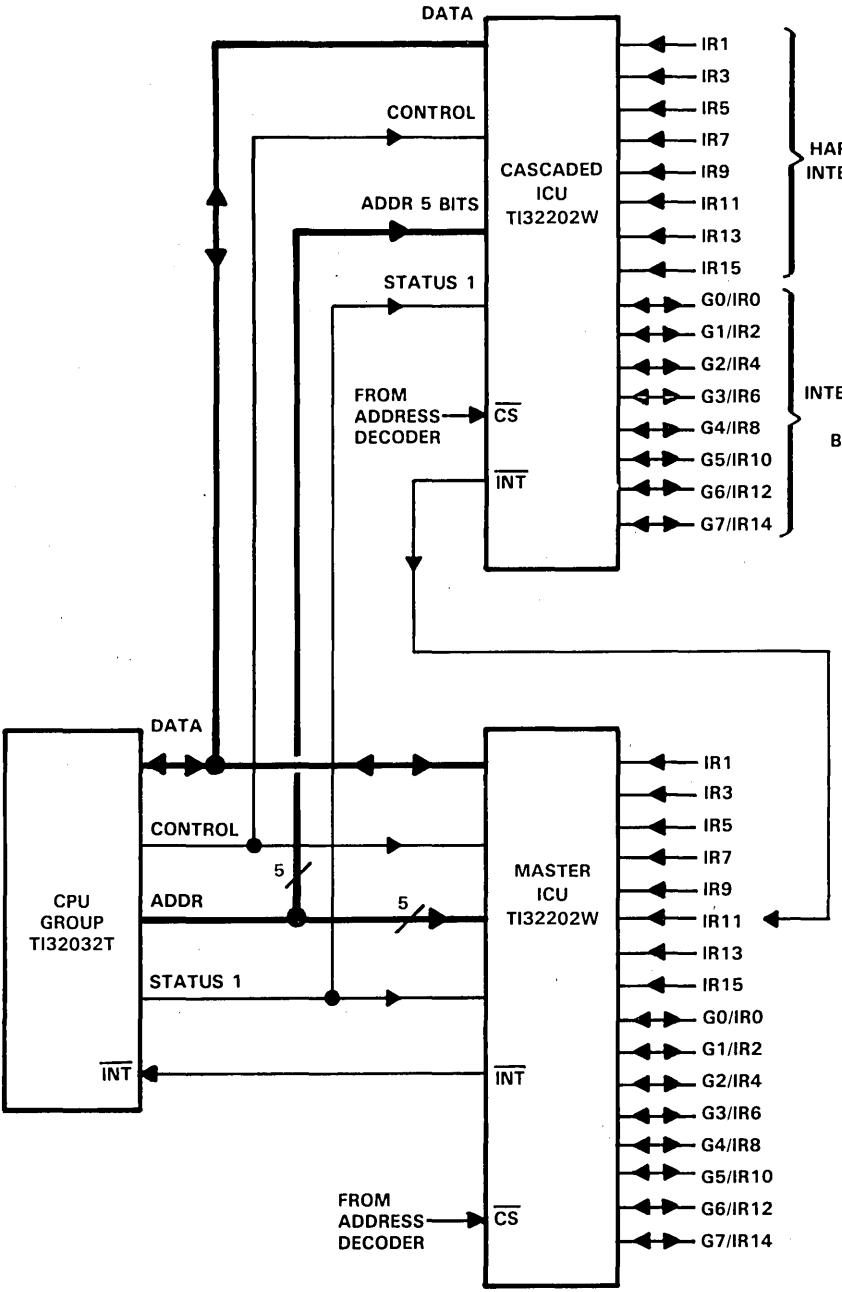


Figure 37. Cascaded Interrupt Control Unit Connections

In a system which uses cascading, two tasks must be performed upon initialization:

1. For each Cascaded ICU in the system, the Master ICU must be informed of the line number (0-15) on which it receives the cascaded requests.
2. A Cascade Table must be established in memory. The Cascade Table is located in a Negative direction from the location indicated by the CPU Interrupt Base (INTBASE) Register. Its entries are 32-bit addresses, pointing to the Vector Registers of each of up to 16 Cascaded ICUs.

Figure 32 illustrates the position of the Cascade Table. To find the Cascade Table entry for a Cascaded ICU, take its Master ICU line number (0-15) and subtract 16 from it, giving an index in the range −16 to −1. Multiply this value by 4 and add the resulting negative number to the contents of the INTBASE Register. The 32-bit entry at this address must be set to the address of the Hardware Vector Register of the Cascaded ICU. This is referred to as the "Cascade Address".

Upon receipt of an interrupt request from a Cascaded ICU, the Master ICU interrupts the CPU and provides the negative Cascade Table index instead of of a (positive) vector number. The CPU, seeing the negative value, uses it as an index into the Cascade Table and reads the Cascade Address from the referenced entry. Applying this address, the CPU performs an "Interrupt Acknowledge, Cascaded" bus cycle (section 3.1.3.4.2), reading the final vector value. This vector is interpreted by the CPU as an unsigned byte, and can therefore be in the range of 0 through 255.

In returning from a Cascaded Interrupt, the service procedure executes the Return from Interrupt (RETI) instruction, as it would for any Maskable Interrupt. The CPU performs an "End of Interrupt, Master" bus cycle, (section 3.1.3.4.2) whereupon the Master ICU again provides the negative Cascade Table index. The CPU, seeing a negative value, uses it to find the corresponding Cascade Address from the Cascade Table. Applying this address, it performs an "End of Interrupt, Cascaded" bus cycle, informing the Cascaded ICU of the completion of the service routine. The byte read from the Cascaded ICU is discarded.

3.1.3.8.4 Nonmaskable Interrupt ($\overline{\text{NMI}}$ pin)

The Nonmaskable Interrupt is triggered whenever a falling edge is detected on the $\overline{\text{NMI}}$ pin. The CPU performs an "Interrupt Acknowledge" bus cycle when processing of this interrupt actually begins. The Interrupt Acknowledge cycle differs from that provided for Maskable Interrupts in that the address presented is FFFF00_{16}. The vector value used for the Nonmaskable Interrupt is taken as 1, regardless of the value read from the bus.

The service procedure returns from the Nonmaskable Interrupt using the Return from Trap (RETT) instruction. No special bus cycles occur on return.

For the full sequence of events in processing the Nonmaskable Interrupt, see section 3.1.3.8.7.

3.1.3.8.5 Traps

A trap is an internally generated interrupt request caused as a direct and immediate result of the execution of an instruction. The Return Address pushed by any trap except

Trace (TRC) is the address of the first byte of the instruction during which the trap occurred. Traps do not disable interrupts, as they are not associated with external events. Traps recognized by the CPU are:

Trap (FPU):	An exceptional condition was detected by the TI32081 Floating Point Unit (FPU) or another coprocessor during the execution of a coprocessor instruction. This trap is requested via the Status Word returned as part of the coprocessor protocol (section 3.1.3.9.1).
Trap (ILL):	Illegal operation. A privileged operation was attempted while the CPU was in User Mode (PSR bit U = 1).
Trap (SVC):	The Supervisor Call (SVC) instruction was executed.
Trap (DVZ):	An attempt was made to divide an integer by zero. (The FPU trap is used for floating point division by zero.)
Trap (FLG):	The FLAG instruction detected a "1" in the CPU PSR F bit.
Trap (BPT):	The Breakpoint (BPT) instruction was exectuted.
Trap (TRC):	The instruction just completed is being traced.
Trap (UND):	An undefined opcode was encountered by the CPU.

A special case is the Trace Trap (TRC), which is enabled by setting the T bit in the Processor Status Register (PSR). At the beginning of each instruction, the T bit is copied into the PSR P ("Trace Pending") bit. If the P bit is set at the end of an instruction, then the trace trap is activated. If any other trap or interrupt request is made during a traced instruction, its entire service procedure is allowed to complete before the trace Trap occurs. Each interrupt and trap sequence handles the P bit for proper tracing, guaranteeing one and only Trace Trap per instruction, and guaranteeing that the Return Address pushed during a Trace Trap is always the address of the next instruction to be traced.

3.1.3.8.6 Prioritization

The TI32032T CPU internally prioritizes simultaneous interrupt and trap requests as follows:

1. Traps other than Trace (Highest priority)
2. Abort
3. Non-Maskable Interrupt
4. Maskable Interrupts
5. Trace Trap (Lowest priority)

3.1.3.8.7 Interrupt/Trap Sequences: Detailed Flow

For purposes of the following detailed discussion of interrupt and trap service sequences, a single sequence called "service" is defined in Table 6. Upon detecting any interrupt request or trap condition, the CPU first performs a sequence dependent upon the type of interrupt or trap. This sequence will include pushing the Processor Status Register and establishing a Vector and a Return Address. The CPU then performs the Service sequence.

Maskable/Nonmaskable Interrupt Sequence. This sequence is performed by the CPU when the $\overline{NMI}$ pin receives a falling edge, or the $\overline{INT}$ pin becomes active with the PSR I bit set. The interrupt sequence begins either at the next instruction boundary or, in the case of String instructions, at the next interruptible point during its execution:

1. If a String instruction was interrupted and not yet completed:
   a. Clear the Processor Status Register P Bit.
   b. Set "Return Address" to the address of the first byte of the interrupted instruction. Otherwise, set "Return Address" to the address of the next instruction.
2. Copy the Processor Status Register (PSR) into a temporary register, then clear PSR bits S,U,T,P, and I.
3. If interrupt is Nonmaskable:
   a. Read a byte from $FFFF00_{16}$, applying Status Code 0100 (Interrupt Acknowledge, Master). Discard the byte read.
   b. Set "Vector" to 1.
   c. Go to Step 8.
4. If the interrupt is Nonvectored:
   a. Read a byte from address $FFFF00_{16}$, applying Status Code 0100 (Interrupt Acknowledge, Master (section 3.1.3.4.2). Discard the byte read.
   b. Set "Vector" to 0.
   c. Go to Step 8.
5. Here the interrupt is Vectored. Read "Byte" from address $FFFE00_{16}$, applying Status Code 0100 (Interrupt Acknowledge, Master (section 3.1.3.4.2).
6. If "Byte" is greater than or equal to 0, then set "Vector" to "Byte" and go to Step 8.
7. If "Byte" is in the range -16 through -1, then the interrupt source is Cascaded. (More negative values are reserved for future use.) Perform the following:
   a. Read the 32-bit Cascade Address from memory. The address is calculated as INTBASE $+4 \times$ Byte.
   b. Read "Vector" applying the Cascade Address just read and Status Code 0101 (Interrupt Acknowledge) (section 3.1.3.4.2).
8. Push the PSR copy (from Step 2) onto the Interrupt Stack as a 16-bit value.
9. Perform Service (Vector, Return Address), Table 6.

Trap Sequence: Traps other than Trace.

1. Restore the currently selected Stack Pointer and the Processor Status Register to their original values at the start of the trapped instruction.
2. Set "Vector" to the value corresponding to the trap type:

FPU: Vector = 3
ILL: Vector = 4
SVC: Vector = 5
DVZ: Vector = 6
FLG: Vector = 7
BPT: Vector = 8
UND: Vector = 10

3. Copy the Processor Status Register (PSR) into a temporary register, then clear PSR bits S, U, P, and T.
4. Push the PSR copy onto the Interrupt Stack as a 16-bit value.
5. Set "Return Address" to the address of the first byte of the trapped instruction.
6. Perform Service (Vector, Return Address), Table 6.

Trace Trap Sequence

1. In the Processor Status Register (PSR), clear the P bit.
2. Copy the PSR into a temporary register, then clear PSR bits S, U, and T.
3. Push the PSR copy onto the Interrupt Stack as a 16-bit value.
4. Set "Vector" to 9.
5. Set "Return Address" to the address of the next instruction.
6. Perform Service (Vector, Return Address), Table 6.

Abort Sequence

1. Restore the currently selected Stack Pointer to its original contents at the beginning of the aborted instruction.
2. Clear the PSR P bit.
3. Copy the PSR into a temporary register, then clear PSR bits S, U, T, and I.
4. Push the PSR copy onto the Interrupt Stack as a 16-bit value.
5. Set "Vector" to 2.
6. Set "Return Address" to the address of the first byte of the aborted instruction.
7. Perform Service (Vector, Return Address), Table 6.

Table 6. Service Sequence
Invoked during all interrupt/trap sequences

Service (Vector, Return Address):

1) Read the 32-bit External Procedure Descriptor from the Interrupt Dispatch Table: address is Vector*4 + INTBASE Register contents.
2) Move the Module field of the Descriptor into the MOD Register.
3) Read the new Static Base pointer from the memory address contained in MOD, placing it into the SB Register.
4) Read the Program Base pointer from memory address MOD+8, and add to it the Offset field from the Descriptor, placing the result in the Program Counter.
5) Flush Queue: Non-sequentially fetch first instructin of Interrupt Routine.
6) Push MOD Register onto the Interrupt Stack as a 16-bit value. (The PSR has already been pushed as a 16-bit value.)
7) Push the Return Address onto the Interrupt Stack as a 32-bit quantity.

3.1.3.9 Coprocessor Instructions

The TI32032T CPU recognizes three groups of instructions as being executable by external coprocessors:

FLoating Point Instruction Set
Memory Management Instruction Set
Application-Specific Coprocessor Instruction Set

Each Coprocessor Instruction Set is validated by a bit in the Configuration Register (section 3.1.1.3). Any Coprocessor Instruction which does not have its corresponding Configuration Register bit set will trap as undefined, without any coprocessor communication attempted by the CPU. This allows software simulation of a nonexisting coprocessor.

3.1.3.9.1 Coprocessor Protocol

Coprocessor instructions have 3-byte Basic Instruction field, consisting of an ID Byte followed by an Operation Word. The ID Byte has three functions:

1. It identifies the instruction as being a coprocessor instruction.
2. It specifies which Coprocessor will execute it.
3. It determines the format of the following Operation Word of the instruction.

Upon receiving a coprocessor instruction, the CPU initiates the sequence outlined in Table 7. While applying Status Code 1111 (Broadcast ID) (section 3.1.3.4.2) the CPU transfers the ID Byte on the least significant byte of the Data Bus (AD0-AD7). All coprocessors input this byte and decode it. The coprocessor selected by the ID Byte is activated, and from this point the CPU is communicating only with it. If any other coprocessor protocol was in progress (e.g., an aborted coprocessor instruction), this transfer cancels it.

Table 7. Coprocessor Protocol

Status Combinations:
Send ID (ID): Code 1111
Xfer Operand (OP): Code 1101
Read Status (ST): Code 1110

STEP	STATUS	ACTION
1	ID	CPU Send ID Byte.
2	OP	CPU Sends Operation Word.
3	OP	CPU Sends Required Operands.
4	—	Coprocessor Starts Execution. CPU Pre-Fetches.
5	—	Coprocessor Pulses $\overline{SPC}$ low.
6	ST	CPU Reads Status Word. (Trap? Alter Flags?)
7	OP	CPU Reads Results (If Any).

The CPU next sends the Operation Word while applying Status Code 1101 (Transfer Coprocessor Operand) (section 3.1.3.4.2). Upon receiving it, the Coprocessor decodes it, and at this point the CPU and the Coprocessor are aware of the number of operands to be transferred and their sizes. The Operation Word is swapped on the Data Bus: that is, bits 0-7 appear on pins AD8-AD15 and bits 8-15 appear on pins AD0-AD7.

Using the Addressing Mode fields within the Operation Word, the CPU starts fetching operands and issuing them to the Coprocessor. To do so, it references any Addressing Mode extensions which may be appended to the Coprocessor instruction. Since the CPU is solely responsible for memory accesses, these extensions are not sent to the coprocessor. The Status Code applied is 1101 (Transfer Coprocessor Operand) (section 3.1.3.4.2).

After the CPU has issued the last operand, the Coprocessor starts the actual execution of the instruction. Upon completion, it will signal the CPU by pulsing $\overline{SPC}$ low. To allow for this and for the address strap translation function, $\overline{AT}/\overline{SPC}$ is normally held high only by a pull-up device of approximately 5 kΩ inside the CPU.

While the coprocessor is executing the instruction, the CPU is free to prefetch instructions into its queue. If it fills the queue before the coprocessor finishes, the CPU will wait, applying Status Code 0011 (Waiting for Coprocessor) (section 3.1.3.4.2).

Upon receiving the pulse on $\overline{SPC}$, the CPU uses $\overline{SPC}$ to read a Status Word from the coprocessor, applying Status Code 1110 (Read Coprocessor Status) (section 3.1.3.4.2). This word has the format shown in Figure 38. If the Q bit ("Quit", Bit 0) is set, this indicates that an error has been detected by the coprocessor. The CPU will not continue the protocol, but will immediately trap through the FPU vector in the Interrupt Table. If the instruction being performed is CMPf and the Q bit is not set, the CPU loads Processor Status Register (PSR) bits N,Z, and L from the corresponding bits in the Status Word. The TI32081 FPU always sets the L bit to zero.

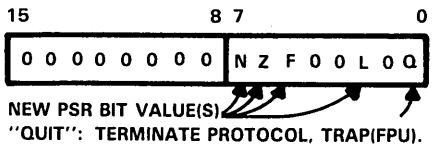


Figure 38. Coprocessor Status Word Format

The last step in the protocol is for the CPU to read a result, if any, and transfer it to the destination. The Read cycles from the Coprocessor Processor are performed by the CPU while applying Status Code 1101 (Transfer Coprocessor Operand) (section 3.1.3.4.2).

An exception to the protocol above is the LMR (Load Memory Management Register) instruction, and a corresponding Application-Specific Coprocessor instruction (LCR: Load ACU Register). In executing these instructions, the protocol ends after the CPU has issued the last operand. The CPU does not wait for an acknowledgment from the coprocessor, and it does not read status.

3.1.3.9.2 Floating-Point Instructions

Table 8 gives the protocols followed for each Floating-Point instruction. The instructions are referenced by their mnemonics. For the bit encodings of each instruction, see the Appendix.

Table 8. Floating-Point Instruction Protocols

MNEMONIC	OPERAND 1 CLASS	OPERAND 2 CLASS	OPERAND 1 ISSUED	OPERAND 2 ISSUED	RETURNED VALUE TYPE AND DEST.	PSR BITS AFFECTED
ADDf	read.f	rmw.f	f	f	f to Op. 2	none
SUBf	read.f	rmw.f	f	f	f to Op. 2	none
MULf	read.f	rmw.f	f	f	f to Op. 2	none
DIVf	read.f	rmw.f	f	f	f to Op. 2	none
MOVf	read.f	write.f	f	N/A	f to Op. 2	none
ABSf	read.f	write.f	f	N/A	f to Op. 2	none
NEGf	read.f	write.f	f	N/A	f to Op. 2	none
CMPf	read.f	read.f	f	f	N/A	N,Z,L
FLOORfi	read.f	write.i	f	N/A	i to Op. 2	none
TRUNCfi	read.f	write.i	f	N/A	i to Op. 2	none
ROUNDfi	read.f	write.i	f	N/A	i to Op. 2	none
MOVFL	read.F	write.L	F	N/A	L to Op. 2	none
MOVLF	read.L	write.F	L	N/A	F to Op. 2	none
MOVif	read.i	write.f	i	N/A	f to Op. 2	none
LFSR	read.D	N/A	D	N/A	N/A	none
SFSR	N/A	write.D	N/A	N/A	D to Op. 2	none

Note:
D = Double Word
i = Integer size (B, W, D) specified in mnemonic.
f = Floating-Point type (F,L) specified in mnemonic.
N/A = Not Applicable to this instruction.

The Operand Class Columns give the Access Class for each general operand, defining how the addressing modes are interpreted (see TI32000 Programmer's Reference Manual).

The Operand Issued Columns show the sizes of the operands issued to the Floating Point Unit by the CPU:

"D" indicates a 32-bit double word.

"i" indicates that the instruction specifies an integer size for the operand (B = Byte, W = Word, D = Double Word).

"f" indicates that the instruction specifies a floating-point size for the operand (F = 32-bit Standard Floating, L = 64-bit Long Floating).

The Returned Value Type and Destination column gives the size of any returned value and where the CPU places it. The PSR Bits Affected column indicates which PSR bits, if any, are updated from the Coprocessor Status Word (Figure 38).

Any operand indicated as being of type "f" will not cause a transfer if the Register Addressing Mode is specified. This is because the Floating Point Registers are physically on the Floating Point Unit and are therefore available without CPU assistance.

3.1.3.9.3 Memory Management Instructions

Table 9 gives the protocols for memory management instructions. Encodings for these instructions may be found in the Appendix.

In executing the RDVAL and WRVAL instructions, the CPU calculates and issues the 32-bit Effective Address of the single operand. The CPU then performs a single-byte read cycle from that address, allowing the MMU to safely abort the instruction if the necessary information is not currently in physical memory. Upon seeing the memory cycle complete, the MMU continues the protocol, and returns the validation result in the F bit of the Coprocessor Status Word.

The size of the Memory Management operand is always a 32-bit double word. For further details of the Memory Management Instruction set, see the *TI32000 Programmer's Manual* and the Appendix.

Table 9. Memory Management Instruction Protocols

MNEMONIC	OPERAND 1 CLASS	OPERAND 2 CLASS	OPERAND 1 ISSUED	OPERAND 2 ISSUED	RETURNED VALUE TYPE AND DEST.	PSR BITS AFFECTED
RDVAL†	addr	N/A	D	N/A	N/A	F
WRVAL†	addr	N/A	D	N/A	N/A	F
LMR†	read.D	N/A	D	N/A	N/A	none
SMR†	write.D	N/A	N/A	N/A	D to Op. 1	none

Note:
In the RDVAL and WRVAL instructions, the CPU issues the address as a Double Word, and performs a single-byte Read cycle from that memory address. For details, see the TI32000 Programmer's Reference Manual and the TI32081W Memory Management Unit Data Sheet.
D = Double Word
† = Privileged Instruction; will trap if CPU is in User Mode.
N/A = Not Applicable to this instruction.

3.1.3.9.4 Application Specific Coprocessor Instructions

Provided in the TI32032T is the capability of communicating with a user-defined, "Application-Specific" Coprocessor. The instruction set provided for an Application-Specific Coprocessor defines the instruction formats, the operand classes, and the communication protocol. Left to the user are the interpretations of the op code fields, the programming model of the Application-Specific Coprocessor, and the actual types of data transfered. The protocol specifies only the size of an operand, not its data type.

Table 10 lists the relevant information for the Application-Specific Coprocessor instruction set. The designation "c" is used to represent an operand which can be a 32-bit ("D") or 64-bit ("Q") quantity in any format: the size is determined by the suffix on the mnemonic. Similarly, an "i" indicates an integer size (Byte, Word, Double Word) selected by the corresponding mnemonic suffix.

Any operand indicated as being of type "c" will not cause a transfer if the register addressing mode is specified. It is assumed in this case that the coprocessor is already holding the operand internally.

For the instruction encodings, see the Appendix.

Table 10. Application-Specific Coprocessor Instruction Protocols

MNEMONIC	OPERAND 1 CLASS	OPERAND 2 CLASS	OPERAND 1 ISSUED	OPERAND 2 ISSUED	RETURNED VALUE TYPE AND DEST.	PSR BITS AFFECTED
CCAL0c	read.c	rmw.c	c	c	c to Op. 2	none
CCAL1c	read.c	rmw.c	c	c	c to Op. 2	none
CCAL2c	read.c	rmw.c	c	c	c to Op. 2	none
CCAL3c	read.c	rmw.c	c	c	c to Op. 2	none
CMOV0c	read.c	write.c	c	N/A	c to Op. 2	none
CMOV1c	read.c	write.c	c	N/A	c to Op. 2	none
CMOV2c	read.c	write.c	c	N/A	c to Op. 2	none
CCMPc	read.f	read.c	c	c	N/A	N,Z,L
CCV0ci	read.c	write.i	c	N/A	i to Op. 2	none
CCV1ci	read.c	write.i	c	N/A	i to Op. 2	none
CCV2ci	read.c	write.i	c	N/A	i to Op. 2	none
CCV3ic	read.i	write.c	i	N/A	c to Op. 2	none
CCV4DQ	read.D	write.Q	D	N/A	Q to Op. 2	none
CCV5QD	read.Q	write.D	Q	N/A	D to Op. 2	none
LCSR	read.D	N/A	D	N/A	N/A	none
SCSR	N/A	write.D	N/A	N/A	D to Op. 2	none
CATST0†	addr	N/A	D	N/A	N/A	F
CATST1†	addr	N/A	D	N/A	N/A	F
LCR†	read.D	N/A	D	N/A	N/A	none
SCR†	write.D	N/A	N/A	N/A	D to Op. 1	none

NOTE:
D = Double Word
i = Integer size (B, W, D) specified in mnemonic.
c = ACU size (D:32 bits or Q:64 bits) specified in mnemonic.
† = Privileged instruction: will trap if CPU is in User Mode.
N/A = Not Applicable to this instruction.

3.2 TI32016T Microprocessor

3.2.1 Programming Model

The TI32000 microprocessor family architecture includes 16 registers on the TI32016T Central Processing Unit (CPU) (Figure 1).

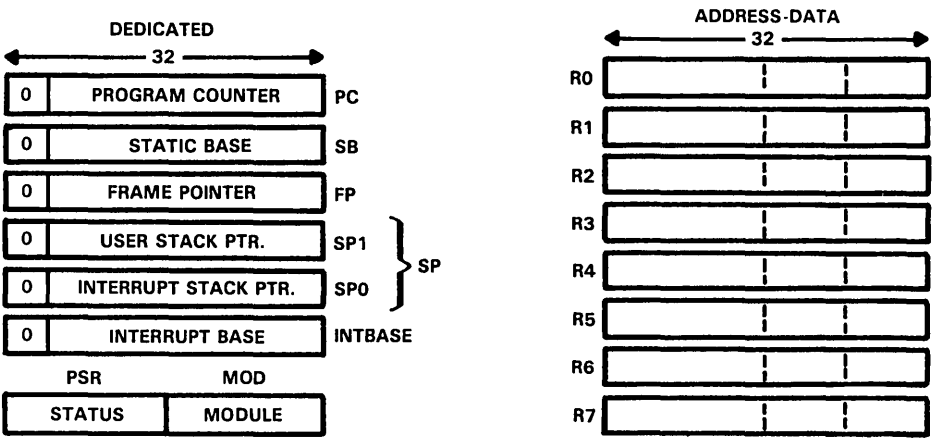


Figure 1. Address-Data and Dedicated Registers

3.2.1.1 Address-Data Registers

The TI32016T contains eight registers (R0 through R7) for meeting high-speed general-storage requirements, such as for holding temporary variables and addresses. These registers are free for any use by the programmer. Each is 32 bits in length. If an Address-Data register is specified for an operand that is 8- or 16-bits long, only the low part (8 or 16 bit section) of the register is used and the high part is not referenced or modified.

3.2.1.2 Dedicated Registers

The eight dedicated registers of the TI32016T are assigned specific functions.

PC: The PROGRAM COUNTER Register is a pointer to the first byte of the instruction currently being executed. The PC Register is used to reference memory in the program section. In the TI32016T CPU, the upper 8 (most significant) bits of this register are always zero.

SP0,SP1: The function of the STACK POINTER Registers is as follows (1) The SP0 register points to the lowest address of the last item stored in the Interrupt Stack. This stack is normally used only by the operating system. It is primarily used for storing

temporary data, and holding return information for operating system subroutines and Interrupt and Trap service routines. (2) The SP1 register points to the lowest address of the last item stored on the User Stack. This stack is used by normal user programs to hold temporary data and subroutine return information.

In this document, reference is made to the SP register. The terms "SP register" or "SP" refer to either SP0 or SP1, depending on the setting of the S bit in the Processor Status Register (PSR). If the S bit in the PSR is 0, then SP refers to SP0. If the S bit in the PSR is 1, then SP refers to SP1. In the TI32016T CPU, the upper 8 (most significant) bits of these registers are always zero.

Stacks in the TI32000 microprocessor family grow downward in memory. A Push operation predecrements the Stack Pointer by the operand length. A Pop operation post increments the Stack Pointer by the operand length.

FP: The FRAME POINTER Register is used by a procedure to access parameters and local variables on the stack. The FP Register is set up on procedure entry with the ENTER instruction and stored on procedure termination with the EXIT instruction.

The FP Register holds the address in memory occupied by the old contents of the Frame Pointer. In the TI32016T CPU, the upper 8 (most significant) bits of this register are always zero.

SB: The STATIC BASE Register points to the global variables of the software module. This register is used to support relocatable global variables for software modules. The SB Register holds the lowest address in memory occupied by the global variables of a module. In the TI32016T CPU, the upper 8 (most significant) bits of this register are always zero.

INTBASE: The INTERRUPT BASE Register holds the address of the dispatch table for interrupts and traps (section 3.2.3.8). The INTBASE register holds the lowest address in memory occupied by the dispatch table. In the TI32016T CPU, the upper 8 (most significant) bits of this register are always zero.

MOD: The MODULE Register holds the address of the module descriptor of the currently executing software module. The MOD register is 16 bits long, therefore the module table must be contained within the first 64k bytes of memory.

PSR: The PROCESSOR STATUS Register holds the status codes for the TI32016T microprocessor. The PSR, as shown in Figure 2, is 16 bits long, divided into two 8-bit halves. The low-order 8 bits are accessible to all programs, but the high-order eight bits are accessible only to programs executing in Operating System Mode.

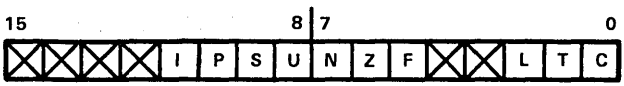


Figure 2. Processor Status Register

C: C bit indicates that a carry or borrow occurred after an addition or subtraction instruction. It can be used with the ADDC and SUBC instructions to perform multiple-precision integer arithmetic calculations. It may have a setting of 0 (no carry or borrow) or 1 (carry or borrow).

T: The T bit causes program tracing. If this bit is a 1, a Trace Trap (TRC) is executed after every instruction (section 3.2.3.8.5).

L: The L bit is altered by comparison instructions. In a comparison instruction, the L bit is set to "1" if the second operand is less than the first operand, and when both operands are interpreted as unsigned integers. Otherwise, it is set to "0". In floating point comparisons, this bit is always cleared.

F: The F bit is a general condition flag, which is altered by many instructions (e.g., integer arithmetic instructions use it to indicate overflow).

Z: The Z bit is altered by comparison instructions. In a comparison instruction, the Z bit is set to "1" if the second operand is equal to the first operand; otherwise it is set to "0".

N: The N bit is altered by comparison instructions. In a comparison instruction, the N bit is set to "1" if both operands are interpreted as signed integers and the second operand is less than the first operand. Otherwise, it is set to "0".

U: If the U bit is "1", no privileged instructions may be executed. If the U bit is "0", then all instructions may be executed. When U = 0, the TI32016T is said to be in the Operating System Mode; when U = 1, the TI32016T is said to be in User Mode. A User Mode program is restricted from executing certain instructions and accessing certain registers which could interfere with the operating system. For example, a User Mode program is prevented from changing the setting of the flag used to indicate its own privilege mode. An Operating System Mode program is assumed to be a trusted part of the operating system, hence it has no such restrictions.

S: The S bit specifies whether the SP0 register or SP1 register is used as the Stack Pointer. The S bit is automatically cleared on interrupts and traps. It may have a setting of 0 (use SP0 register) or 1 (use SP1 register).

P: The P bit prevents a TRC trap from occurring more than once for an instruction (section 3.2.3.8.5). It may have a setting of 0 (no trace pending) or 1 (trace pending).

I: When the I bit is "1", all interrupts will be accepted (section 3.2.3.8). If the I bit is "0", only the NMI interrupt is accepted. Trap enables are not affected by this bit.

3.2.1.3 Configuration Register (CFG)

Within the Control section of the TI32016T CPU is a 4-bit CFG register that declares the presence of certain external devices. It is referenced by only one instruction, SETCFG, which is intended to be executed only as part of system initialization after reset. The format of the CFG Register is shown in Figure 3.

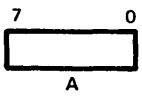
Figure 3. CFG Register

The CFG I bit declares the presence of external interrupt vectoring circuitry, specifically, the TI32202 Interrupt Control Unit. If the CFGI bit is "1", interrupts requested through the $\overline{INT}$ pin are "Vectored". If it is "0", these interrupts are "nonvectored" (section 3.2.3.8).

The F,M, and C bits declare the presence of the Floating Point Unit (FPU), Memory Management Unit (MMU), and Application-Specific Coprocessors. If these bits are 0s, the corresponding instructions are trapped as being undefined.

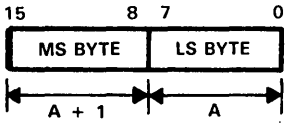
3.2.1.4 Memory Organization

The main memory of the TI32016T is a uniform linear address space. Memory locations are numbered sequentially starting at 0 and ending at $2^{24}-1$. The number specifying a memory location is called an address.The contents of each memory location is a byte consisting of 8 bits. Unless otherwise noted, diagrams in this document show data stored in memory with the lowest address on the right and the highest address on the left. In addition, when data is shown vertically, the lowest address is at the top of a diagram and the highest address is at the bottom of the diagram. When bits are numbered in a diagram, the least significant bit is given the number 0, and is shown at the right of the diagram. Bits are numbered in increasing significance and toward the left.



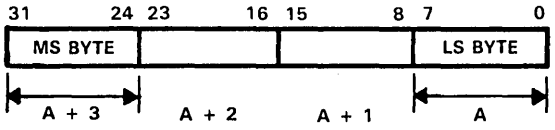
Byte at Address A.

Two contiguous bytes are called a word. Except where noted (section 3.2.2.1), the least significant byte of a word is stored at the lower address, and the most significant byte of the word is stored at the next higher address. In memory, the address of a word is the address of its least significant byte, and a word may start at any address.



Word at Address A.

Two contiguous words are called a double word. Except where noted (section 3.2.2.1), the least significant word of a double word is stored at the lowest address and the most significant word of the double word is stored at the address two greater. In memory, the address of a double word is the address of its least significant byte, and a double word may start at any address.



Double-word at Address A.

Although memory is addressed as bytes, it is actually organized as words. Therefore, words and double words that are aligned to start at even addresses (multiples of two) are accessed more quickly than words and double words that are not so aligned.

3.2.1.5 Dedicated Tables

Two of the TI32016T dedicated registers (MOD and INTBASE) serve as pointers to dedicated tables in memory (section 3.2.3.8).

The INTBASE register points to the Interrupt Dispatch and Cascade tables.

The MOD Register contains a pointer into the Module Table whose entries are called Module Descriptors. A Module Descriptor contains four pointers, three of which are used by the TI32016T. At any time, the MOD register contains the address of the Module Descriptor for the currently running module. It is automatically updated by the Call External Procedure instructions (CXP and CXPD).

The format of a Module Descriptor is shown in Figure 4. The Static Base entry contains the address of static data assigned to the running module. It is loaded into the CPU Static Base register by the CXP and CXPD instructions. The Program Base entry contains the address of the first byte of instruction code in the module. Since a module may have multiple entry points, the Program Base pointer serves only as a reference to find them.

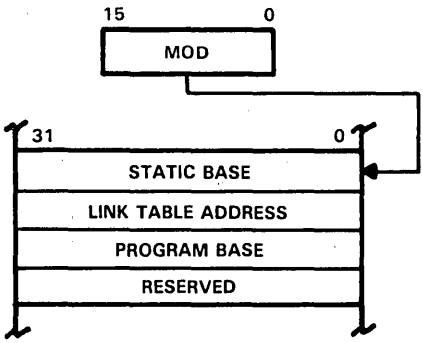


Figure 4. Module Descriptor Format

The Link Table Address points to the Link Table for the currently running module. The Link Table provides the information needed for:

1. Sharing variables between modules. Such variables are accessed through the Link Table via the External addressing mode.
2. Transferring control from one module to another. This is done via the Call External Procedure (CXP) instruction.

The format of a Link Table is shown in Figure 5. A Link Table Extry for an external variable contains the 32-bit address of that variable. An entry for an external procedure contains two 16-bit fields: Module and Offset. The Module field contains the new MOD register contents for the module being entered. The Offset field is an unsigned number giving the position of the entry point relative to the new module's Program Base pointer.

For further details of the functions of these tables, refer to the *TI32000 Programmer's Reference Manual.*

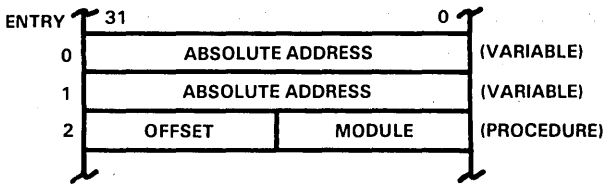


Figure 5. A Sample Link Table

3.2.2 Instruction Set

3.2.2.1 General Instruction Format

Figure 6 shows the general format of a TI32000 instruction. The Basic Instruction is one to three bytes long and contains the Opcode and up to two 5-bit General Addressing Mode (*gen*) fields. Following the Basic Instruction field is a set of optional extensions, which may appear depending on the instruction and the addressing modes selected. Index Byts apear when either or both *gen* fields specify Scaled Index. In this case, the *gen* field specifies only the Scale Factor (1, 2, 4, or 8) and the Index Byte specifies which Address-Data Register to use as the index and which addressing mode calculation to perform before indexing. See Figure 7.

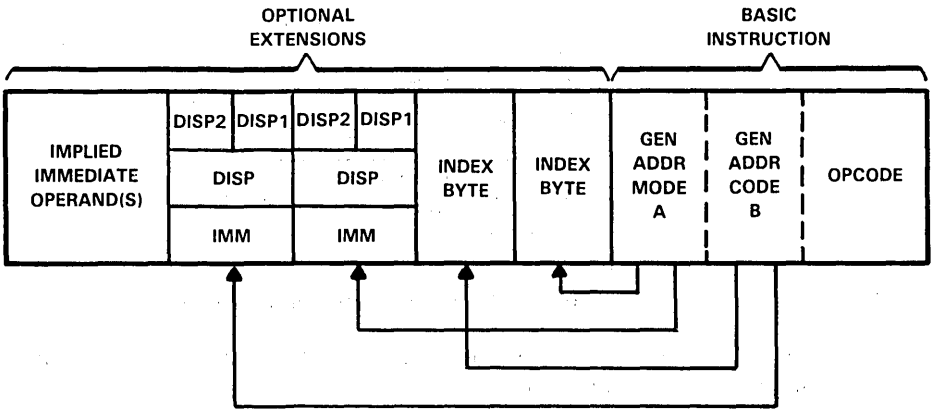


Figure 6. General Instruction Format

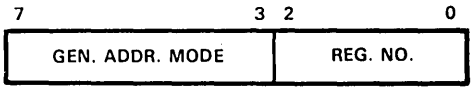


Figure 7. Index Byte Format

Following Index Bytes come any displacements (addressing constants) or immediate values associated with the selected addressing modes. Each Displacement/Immediate (disp/imm) field may contain one or two displacements, or one immediate value. The size of a disp field is encoded within the top bits of that field, as shown in Figure 8, with the remaining bits interpreted as a signed (two's complement) value.The size of an immediate value is determined from the opcode field. Both disp and imm fields are stored most significant byte first. Note that this is different from the memory representation of data (section 3.2.1.4).

Some instructions require additional, "implied" immediates and/or displacements, apart from those associated with addressing modes. Any such extensions appear at the end of the instruction, in the order that they appear within the list of operands in the instruction definition (section 3.2.2.3).

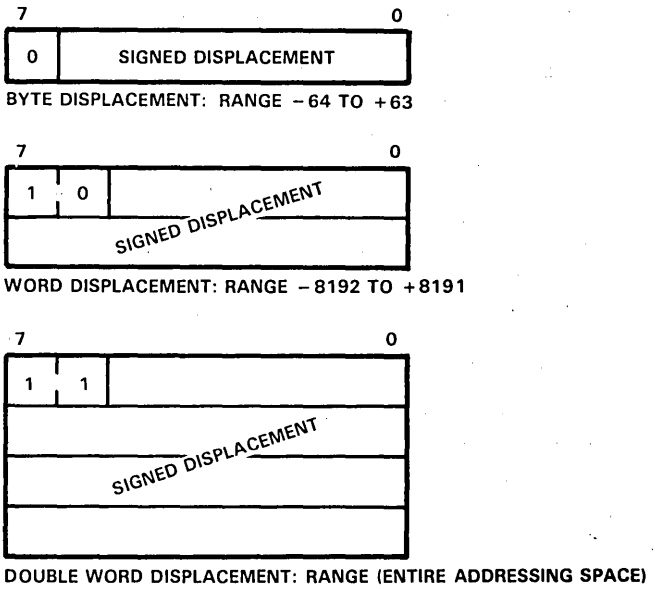


Figure 8. Displacement Encodings

3.2.2.2 Addressing Modes

The TI32016T CPU generally accesses an operand by calculating its Effective Address based on information available when the operand is to be accessed. The method to be used in performing this calculation is specified by the programmer as an "addressing mode"

Addressing modes in the TI32016T are designed to optimally support high-level language accesses to variables. In nearly all cases, a variable access requires only one addressing mode within the instruction that acts upon that variable. Extraneous data movement is therefore minimized.

TI32016T Addressing Modes fall into nine basic types:

1. Register — The operand is available in one of the eight Address-Data Registers. In certain coprocessor instructions, an auxiliary set of eight registers may be referenced instead.
2. Register Relative — An Address-Data Register contains an address to which is added a displacement value from the instruction, yielding the Effective Address of the operand in memory.
3. Memory Space — Identical to Register Relative, except that the register used is one of the dedicated registers: PC, SP, SB, or FP. These registers point to data areas generally needed by high-level languages.
4. Memory Relative — A pointer variable is found within the memory space pointed to by the SP, SB, or FP register. A displacement is added to that pointer to generate the Effective Address of the operand.
5. Immediate — The operand is encoded within the instruction. This addressing mode is not allowed if the operand is to be written.
6. Absolute — The address of the operand is specified by a displacement field in the instruction.
7. External — A pointer value is read from a specified entry of the current Link Table. To this pointer value is added a displacement, yielding the Effective Address of the operand.
8. Top-of-Stack — The currently selected Stack Pointer (SP0 or SP1) specifies the location of the operand. The operand is pushed or popped, depending on whether it is written or read.
9. Scaled Index — Although encoded as an addressing mode, Scaled Indexing is an option on any addressing mode except Immediate or another Scaled Index. It has the effect of calculating an Effective Address, then multiplying any Address-Data Register by 1, 2, 4, or 8 and adding it to the total, yielding the final Effective Address of the operand.

Table 1 is a brief summary of the addressing modes. For a complete description of their actions, see the *TI32000 Programmer's Reference Manual.*

3.2.2.3 Instruction Set Summary

Table 2 presents a brief description of the TI32016T instruction set. The Format Column refers to the Instruction Format Tables (See Appendix). The Instruction Column gives the instruction as coded in assembly language, and the Description column provides a short description of the function provided by that instruction. Further details of the exact operations performed by each instruction may be found in the *TI32000 Programmer's Reference Manual.*

Notations:

i = integer length suffix:
B = Byte
W= Word
D = Double Word

f = Floating Point length suffix:
F = Standard Floating
L = Long Floating

gen = General operand: Any addressing mode can be specified.

short = A 4-bit value encoded within the Basic Instruction (see Appendix for encoding).

imm = Implied immediate operand. An 8-bit value appended after any addressing extensions.

disp = displacement (addressing constant): 8, 16, or 32 bits. All three lengths equal.

reg = Any Address-Data register: R0-R7.

areg = Any Dedicated address register: SP, SB, FP, MOD, INTBASE, PSR, US (bottom 8 PSR bits).

mreg = Any Memory Management Status/Control Register.

creg = An Application-Specific Coprocessor Register (Implementation Dependent).

cond = Any condition code, encoded as the 4-bit field within the Basic Instruction (see Appendix for encodings).

3.2.3 Functional Description

3.2.3.1 Power and Grounding

The TI32016T requires a single 5-V power supply, applied on pin 18 (V_{CC}). See DC specifications in the TI32016T data sheet.

Grounding connections are made on two pins. Logic Ground (GNDL, pin 24) is the common pin for on-chip logic, and Buffer Ground (GNDB, pin 25) is the common pin for the output drivers. For optimal noise immunity, it is recommended that GNDB and GNDL be connected together through a single conductor. All other ground connections should be made to the common line as shown in Figure 9.

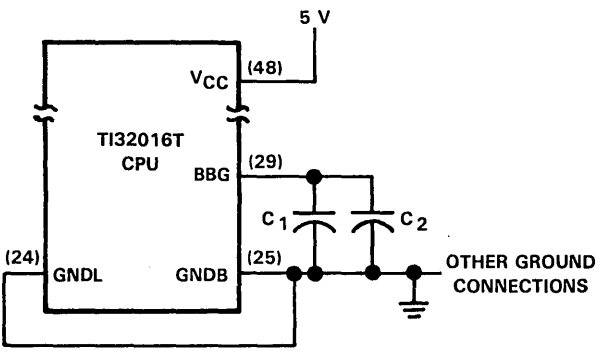


Figure 9. Recommended Supply Connections

Table 1. TI32000 Addressing Modes

ENCODING	MODE	ASSEMBLER SYNTAX	EFFECTIVE ADDRESS
Register			
00000	Register 0	R0 or F0	None: Operand is in the specified register
00001	Register 1	R1 or F1	
00010	Register 2	R2 or F2	
00011	Register 3	R3 or F3	
00100	Register 4	R4 or F4	
00101	Register 5	R5 or F5	
00110	Register 6	R6 or F6	
00111	Register 7	R7 or F7	
Register Relative			
01000	Register 0 relative	disp(R0)	Disp + Register.
01001	Register 1 relative	disp(R1)	
01010	Register 2 relative	disp(R2)	
01011	Register 3 relative	disp(R3)	
01100	Register 4 relative	disp(R4)	
01101	Register 5 relative	disp(R5)	
01110	Register 6 relative	disp(R6)	
01111	Register 7 relative	disp(R7)	
Memory Relative			
10000	Frame memory relative	disp2(disp1(FP))	Disp 2 + Pointer; Pointer found at address Disp1 + Register. "SP" is either SP0 or SP1, as selected in PSR.
10001	Stack memory relative	disp2(disp1(SP))	
10010	Static memory relative	disp2(disp1(SB))	
Immediate			
10100	Immediate	value	None: Operand is input from instruction queue.
Absolute			
10101	Absolute	@disp	Disp.
External			
10110	External	EXT (disp1) + disp2	Disp2 + Pointer; Pointer is found at Link Table Entry number Disp1.
Top of Stack			
10111	Top of Stack	TOS	Top of current stack, using either User or Interrupt Stack Pointer, as selected in PSR. Automatic Push/Pop included.

Table 1. TI32000 Addressing Modes (Continued)

ENCODING	MODE	ASSEMBLER SYNTAX	EFFECTIVE ADDRESS
Memory Space			
11000	Frame memory	disp(FP)	Disp + Register, "SP" is either SP0 or SP1, as selected in PSR.
11001	Stack memory	disp(SP)	
11010	Static memory	disp(SB)	
11011	Program memory	* + disp	
Scaled Index			
11100	Index, bytes	mode[Rn:B]	EA (mode) + Rn.
11101	Index, words	mode[Rn:W]	EA (mode) + 2 × Rn.
11110	Index, double words	mode[Rn:D]	EA (mode) + 4 × Rn.
11111	Index, quad words	mode[Rn:Q]	EA (mode) + 8 × Rn. "Mode" and "n" are contained within the Index Byte. EA(mode) denotes the effective address generated using mode.

Table 2. TI32000 Instuction Set Summary

MOVES

Format	Operation	Operands	Description
4	MOVi	gen,gen	Move a value.
2	MOVQi	short,gen	Extend and move a signed 4-bit constant.
7	MOVMi	gen,gen,disp	Move Multiple: disp bytes (1 to 16).
7	MOVZBW	gen,gen	Move with zero extension.
7	MOVZiD	gen,gen	Move with zero extension.
7	MOVXBW	gen,gen	Move with sign extension.
7	MOVXiD	gen,gen	Move with sign extension.
4	ADDR	gen,gen	Move Effective Address.

INTEGER ARITHMETIC

Format	Operation	Operands	Description
4	ADDi	gen,gen	Add.
2	ADDQi	short,gen	Add signed 4-bit constant.
4	ADDCi	gen,gen	Add with carry.
4	SUBi	gen,gen	Subtract.
4	SUBCi	gen,gen	Subtract with carry (borrow).
6	NEGi	gen,gen	Negate (2's complement).
6	ABSi	gen,gen	Take absolute value.
7	MULi	gen,gen	Multiply
7	QUOi	gen,gen	Divide, rounding toward zero.
7	REMi	gen,gen	Remainder from QUO.
7	DIVi	gen,gen	Divide, rounding down.
7	MODi	gen,gen	Remainder from DIV (Modulus).
7	MEIi	gen,gen	Multiply to Extended Integer.
7	DEIi	gen,gen	Divide Extended Integer.

PACKED DECIMAL (BCD) ARITHMETIC

Format	Operation	Operands	Description
6	ADDPi	gen,gen	Add Packed.
6	SUBPi	gen,gen	Subtract Packed.

INTEGER COMPARISON

Format	Operation	Operands	Description
4	CMPi	gen,gen	Compare.
2	CMPQi	short,gen	Compare to signed 4-bit constant.
7	CMPMi	gen,gen,disp	Compare Multiple: disp bytes (1 to 16).

LOGICAL AND BOOLEAN

Format	Operation	Operands	Description
4	ANDi	gen,gen	Logical AND.
4	ORi	gen,gen	Logical OR.
4	BICi	gen,gen	Clear selected bits.
4	XORi	gen,gen	Logical Exclusive OR.
6	COMi	gen,gen	Complement all bits.
6	NOTi	gen,gen	Boolean complement: LSB only.
2	Scondi	gen	Save condition code (cond) as a Boolean variable of size i.

Table 2. TI32000 Instruction Set Summary (Continued)

SHIFTS

Format	Operation	Operands	Description
6	LSHi	gen,gen	Logical Shift, left or right.
6	ASHi	gengen	Arithmetic Shift, left or right.
6	ROTi	gen,gen	Rotate, left or right.

BITS

Format	Operation	Operands	Description
4	TBITi	gen,gen	Test bit.
6	SBITi	gen,gen	Test and set bit.
6	SBITIi	gen,gen	Test and set bit, interlocked
6	CBITi	gen,gen	Test and clear bit.
6	CBITIi	gen,gen	Test and clear bit, interlocked.
6	IBITi	gen,gen	Test and invert bit.
8	FFSi	gen,gen	Find first set bit

BIT FIELDS

Bit fields are values in memory that are not aligned to byte boundaries. Examples are PACKED arrays and records used in Pascal. "Extract" instructions read and align a bit field. "Insert" instructions write a bit field from an aligned source.

Format	Operation	Operands	Description
8	EXTi	reg,gen,gen,disp	Extract bit field (array oriented).
8	INSi	reg,gen,gen,disp	Insert bit field (array oriented).
8	INSi	reg,gen,gen,disp	Insert bit field (array oriented).
7	EXTSi	gen,gen,imm,imm	Extract bit field (short form).
7	INSSi	gen,gen,imm,imm	Insert bit field (short form).
8	CVTP	reg,gen,gen	Convert to Bit Field Pointer.

ARRAYS

Format	Operation	Operands	Description
8	CHECKi	reg,gen,gen	Index bounds check.
8	INDEXi	reg,gen,gen	Recursive indexing step for multiple-dimensional arrays.

STRINGS

String instructions assign specific functions to the Address-Data Registers:

R4 – Comparison Value

R3 – Translation Table Pointer

R2 – String 2 Pointer

R1 – String 1 Pointer

R0 – Limit Count

Options on all strong instructions are:

B (Backward): Decrement string pointers after each step rather than incrementing.

U (Until match): End instruction if String 1 entry matches R4.

W (While match): End instruction if String 1 entry does not match R4.

All string instructions end when R0 decrements to zero.

Table 2. TI32000 Instruction Set Summary (Continued)

Format	Operation	Operands	Descriptions
5	MOVSi	options	Move string 1 to String 2.
	MOVST	options	Move string, translating bytes.
5	CMPSi	options	Compare String 1 to String 2.
	CMPST	options	Compare, translating String 1 bytes.
5	SKPSi	options	Skip over String 1 entries.
	SKPST	options	Skip, translating bytes for Until/While.

JUMPS AND LINKAGE

Format	Operation	Operands	Description
3	JUMP	gen	Jump.
0	BR	disp	Branch (PC Relative).
0	Bcond	disp	Conditional branch.
3	CASEi	gen	Multiway branch.
2	ACBi	short,gen,disp	Add 4-bit constant and branch if non-zero.
3	JSR	gen	Jump to subroutine.
1	BSR	disp	Branch to subroutine.
1	CXP	disp	Call external procedure.
3	CXPD	gen	Call external procedure using descriptor.
1	SVC		Supervisor Call.
1	FLAG		Flag Trap.
1	BPT		Breakpoint Trap.
1	ENTER	[reg list],disp	Save registers and allocate stack frame (Enter Procedure).
1	EXIT	[reg list]	Restore registers and reclaim stack frame (Exit Procedure).
1	RET	disp	Return from subroutine.
1	RXP	disp	Return from external procedure call.
1	RETT	disp	Return from trap. (Privileged)
1	RETI		Return from interrupt. (Privileged)

Table 2. TI32000 Instruction Set Summary (Continued)

CPU REGISTER MANIPULATION

Format	Operation	Operands	Description
1	SAVE	[reg list]	Save Address-Data Registers.
1	RESTORE	[reg list]	Restore Address-Data registers.
2	LPRi	areg,gen	Load Dedicated Register. (Privileged if PSR or INTBASE)
2	SPRi	areg,gen	Store Dedicated Register. (Privileged if PSR or INTBASE)
3	ADJSPi	gen	Adjust Stack Pointer.
3	BISPSRi	gen	Set selected bits in PSR. (Privileged if not Byte length)
3	BICPSRi	gen	Clear selected bits in PSR. (Privileged if not Byte length)
5	SETCFG	[option list]	Set Configuration Register. (Privileged)

FLOATING POINT

Format	Operation	Operands	Description
11	MOVf	gen,gen	Move a Floating Point value.
9	MOVLF	gen,gen	Move and shorten a Long value to Standard.
9	MOVFL	gen,gen	Move and lengthen a Standard value to Long.
9	MOVif	gen,gen	Convert any integer to Standard or Long Floating.
9	ROUNDfi	gen,gen	Convert to integer by rounding.
9	TRUNCfi	gen,gen	Convert to integer by truncating, toward zero.
9	FLOOR fi	gen,gen	Convert to largest integer less than or equal to value.
11	ADDf	gen,gen	Add.
11	SUBf	gen,gen	Subtract.
11	MULf	gen,gen	Multiply.
11	DIVf	gen,gen	Divide.
11	CMPf	gen,gen	Compare.
11	NEGf	gen,gen	Negate.
11	ABSf	gen,gen	Take absolute value.
9	LFSR	gen	Load FSR.
9	SFSR	gen	Store FSR.

MEMORY MANAGEMENT

Format	Operation	Operands	Description
14	LMR	mreg,gen	Load Memory Management Register. (Privileged)
14	SMR	mreg,gen	Store Memory Management Register. (Privileged)
14	RDVAL	gen	Validate address for reading. (Privileged)
14	WRVAL	gen	Validate address for writing. (Privileged)
8	MOVSUi	gen,gen	Move a value from Operating System Space to User Space. (Privileged)
8	MOVUSi	gen,gen	Move a value from User Space to Operating System Space. (Privileged)

Table 2. TI32000 Instruction Set Summary (Concluded)

MISCELLANEOUS

Format	Operation	Operands	Description
1	NOP		No Operation.
1	WAIT		Wait for interrupt.
1	DIA		Diagnose. Single-byte "Branch to Self" for hardware breakpointing. Not for use in programming.

APPLICATION-SPECIFIC COPROCESSOR (ACU)

Format	Operation	Operands	Description
15.5	CCAL0c	gen,gen	ACU Calculate.
15.5	CCAL1c	gen,gen	
15.5	CCAL2c	gen,gen	
15.5	CCAL3c	gen,gen	
15.5	CMOV0c	gen,gen	ACU Move.
15.5	CMOV1c	gen,gen	
15.5	CMOV2c	gen,gen	
15.5	CCMPc	gen,gen	ACU Compare.
15.1	CCV0ci	gen,gen	ACU Convert.
15.1	CCV1ci	gen,gen	
15.1	CCV2ci	gen,gen	
15.1	CCV3ic	gen,gen	
15.1	CCV4DQ	gen,gen	
15.1	CCV5QD	gen,gen	
15.1	LCSR	gen	Load ACU Status Register.
15.1	SCSR	gen	Store ACU Status Register.
15.0	CATST0	gen	ACU Address/Test. (Privileged)
15.0	CATST1	gen	(Privileged)
15.0	LCR	creg,gen	Load ACU Register. (Privileged)
15.0	SCR	creg,gen	Store ACU Register. (Privileged)

In addition to V_{CC} and GND, the TI32016T CPU uses an internally generated negative voltage. It is necessary to filter this voltage externally by attaching a pair of capacitors (Figure 9) from the BBG pin to ground. Recommended values for these are:

C1: 1 μF, Tantalum
C2: 1000 pF, low inductance. This should be either a disc or monolithic ceramic capacitor.

3.2.3.2 Clocking

The TI32016T inputs clocking signals from the TI32201 Timing Control Unit (TCU), which presents two nonoverlapping phases of a single clock frequency. These phases are called CLK1 (pin 26) and CLK2 (pin 27). Their relationship to each other is shown in Figure 10.

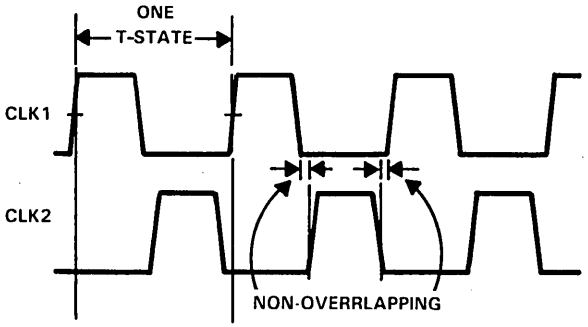


Figure 10. Clock Timing Relationships

Each positive edge of CLK1 defines a transition in the timing state (T-State) of the CPU. One T-State represents the execution of one microinstruction within the CPU, and/or one step of an external bus transfer. See the ac timing characteristics in the TI32016T Data Sheet for complete specifications on CLK1 and CLK2.

Since the TCU presents signals with very fast transitions, it is recommended that the conductors carrying CLK1 and CLK2 be kept as short as possible, and that they not be connected anywhere except from the TCU to the CPU and, if present, the Memory Management Unit (MMU). A TTL Clock signal (CTTL) is provided by the TCU for all other clocking.

3.2.3.3 Resetting

The $\overline{RST}/\overline{ABT}$ pin serves both as a Reset for on-chip logic and as the Abort input for Memory-Managed systems. For its use as the Abort command, see section 3.2.3.5.4.

The CPU may be reset at any time by pulling the $\overline{RST}/\overline{ABT}$ pin low for at least 64 clock cycles. Upon detecting a reset, the CPU terminates instruction processing, resets its internal logic, and clears the Program Counter (PC) and Processor Status Register (PSR) to all zeros.

On application of power, $\overline{RST}/\overline{ABT}$ must be held low for at least 50 μs after V_{CC} is stable. This is to ensure that all on-chip voltages are completely stable before operation.

Whenever a Reset is applied, it must remain active for not less than 64 clock cycles. The trailing (positive-going) edge must occur while CLK1 is high, and no later than 10 ns before the CLK1 trailing edge. See Figures 11 and 12.

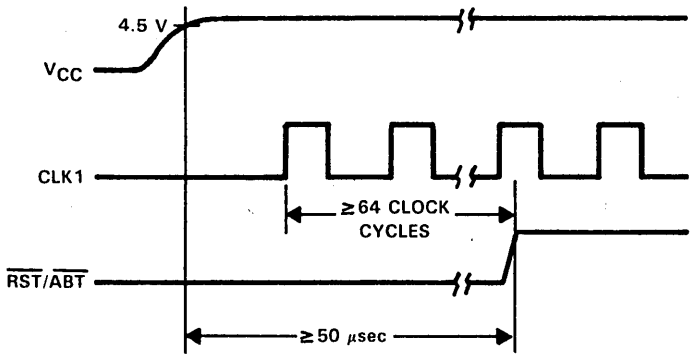


Figure 11. Power-On Reset Requirements

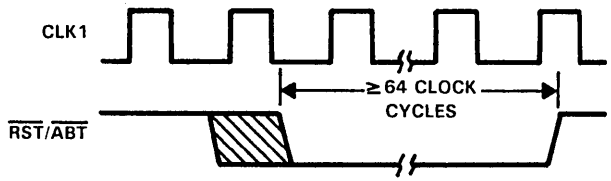


Figure 12. General Reset Timing

The TI32201 Timing Control Unit (TCU) provides circuitry to meet the reset requirements of the TI32016T CPU. Figure 13 shows the recommended connections for a nonmemory-mananaged system. Figure 14 shows the connections for a memory-managed system.

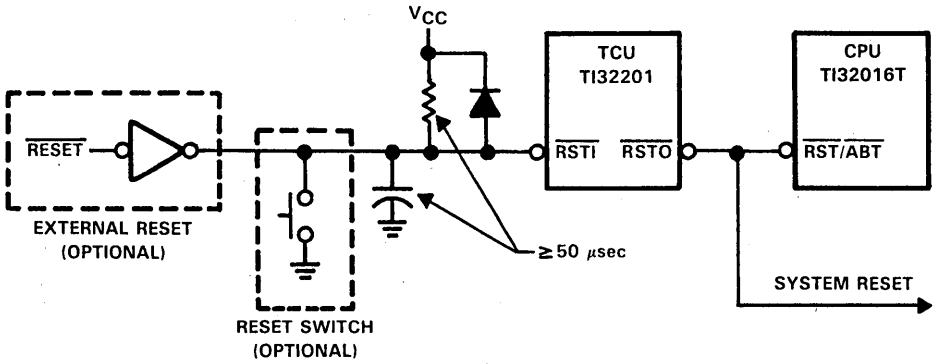


Figure 13. Recommended Reset Connections, Non-Memory-Managed System

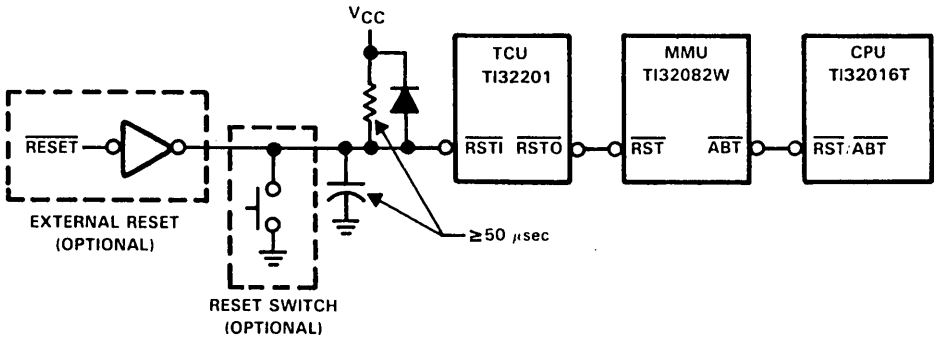


Figure 14. Recommended Reset Connections, Memory-Managed System

3.2.3.4 Bus Cycles

The TI32016T CPU has a strap option that defines the Bus Timing Mode as either with or without Address Translation. For details covering the use of the strap, refer to section 3.2.3.5.

The CPU will perform a bus cycle for one of the following reasons:

1. To write or read data, to or from memory or a peripheral interface device. Peripheral input and output are memory-mapped in the TI32000 family.
2. To fetch instructions into the 8-byte instruction queue. This happens whenever the bus would otherwise be idle and the queue is not already full.
3. To acknowledge an interrupt and allow external circuitry to provide a vector number, or to acknowledge completion of an interrupt service routine.
4. To transfer information to or from a Coprocessor.

In terms of bus timing, cases 1 through 3 above are identical. The only external difference between cases 1 through case 3 is the 4-bit code placed on the Bus Status pins (ST0-ST3). Coprocessor cycles differ in that separate control signals are applied. Refer to section 3.2.3.4.6.

The sequence of events in a noncoprocessor bus cycle is shown in Figure 16 for a Read cycle and Figure 17 for a Write cycle. The cases shown assume that the selected memory or interface device is capable of communicating with the CPU at full speed. If it is not, then cycle extension may be requested through the RDY line. Refer to section 3.2.3.4.1.

A full-speed bus cycle is performed in four cycles of CLK1, labeled T1 through T4. Clock cycles not associated with a bus cycle are designated Ti (for "idle").

During T1, the CPU applies an address on pins AD0-AD15 and A16-A23. It also provides a low-going pulse on the $\overline{ADS}$ pin, which serves the dual purpose of informing external circuitry that a bus cycle is starting and of providing control to an external latch for demultiplexing Address bits 0-15 from the AD0-AD15 pins. See Figure 15. Also during this time the status signals $\overline{DDIN}$, indicating the direction of the transfer, and $\overline{HBE}$, indicating whether the high byte (AD8-AD15) is to be referenced, become valid.

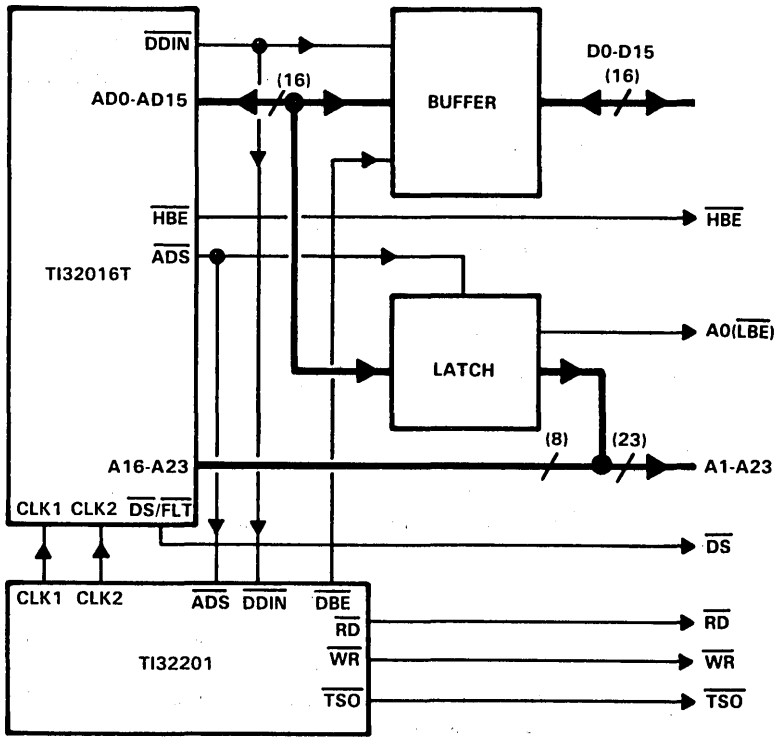


Figure 15. Bus Connections

During T2, the CPU switches the Data Bus AD0-AD15 to either accept or present data. Note that the signals A16-A23 remain valid, and need not be latched . It also starts the Data Strobe ($\overline{DS}$), signaling the beginning of the data transfer. Associated signals from the TI32201 Timing Control Unit are also activated at this time: $\overline{RD}$ (Read Strobe) or $\overline{WR}$ (Write Strobe), $\overline{TSO}$ (Timing State Output, indicating that T2 has been reached), and $\overline{DBE}$ (Data Buffer Enable).

The T3 state provides for access time requirements, and it occurs at least once in a bus cycle. At the beginning of T3, on the rising edge of the CLK1 clock, the RDY line is sampled to determine whether the bus cycle will be extended (section 3.2.3.4.1).

If the CPU is performing a Read cycle, the Data Bus (AD0-AD15) is sampled at the falling edge of CLK2 in the last T3 state. Data must, however, be held at least until the beginning of T4. $\overline{DS}$ and $\overline{RD}$ are guaranteed not to go inactive before this point, so the rising edge of either of them may safely be used to disable the device providing the input data.

The T4 state finishes the bus cycle. At the beginning of T4, the $\overline{DS}$, $\overline{RD}$ or $\overline{WR}$, and $\overline{TSO}$ signals go inactive, and on the rising edge of CLK2, $\overline{DBE}$ goes inactive, having provided for necessary data hold times. Data during Write cycles remains valid from the CPU throughout T4. Note that the Bus Status lines (ST0—ST3) change at the beginning of T4, anticipating the following bus cycle (if any).

3.2.3.4.1 Cycle Extension

To allow sufficient strobe widths and access times for any speed of memory or peripheral device, the TI32016T provides for extension of a bus cycle. Any type of bus cycle except a coprocessor cycle can be extended.

In Figures 16 and 17, note that during T3 all bus control signals from the CPU and TCU are flat. Therefore, a bus cycle can be cleanly extended by causing the T3 state to be repeated. This is the purpose of the Ready (RDY) pin.

At the end of T2, on the falling edge of CLK2, the RDY line is sampled by the CPU. If RDY is high, the next T-states will be T3 and T4, ending the bus cycle. If RDY is low, An additional T3 state will be inserted after the initial T3 state and the RDY line will again be sampled on the falling edge of CLK2. Each additional T3 state after the first is referred to as a "Wait State". See Figure 18.

The RDY pin is driven by the TI32201 Timing Control Unit, which applies wait-states to the CPU as requested on three sets of pins:

1. $\overline{\text{CWAIT}}$ (Continuous Wait) holds the CPU in wait-states until removed.
2. $\overline{\text{WAIT1}}$, $\overline{\text{WAIT2}}$, $\overline{\text{WAIT4}}$, $\overline{\text{WAIT8}}$ (Collectively $\overline{\text{WAITn}}$) may be given a four-bit value requesting a specific number of wait-states from 0—15.
3. $\overline{\text{PER}}$ (Peripheral) inserts five additional wait-states and causes the TCU to reshape the $\overline{\text{RD}}$ and $\overline{\text{WR}}$ strobes. This provides the setup and hold times required by most MOS peripheral interface devices.

Combinations of these various Wait requests are both legal and useful. For details on their use, see section 3.6.

Figure 19 illustrates a typical Read cycle, with two wait-states requested through the TCU $\overline{\text{WAITn}}$ pins.

3.2.3.4.2 Bus Status

The TI32016T CPU presents 4 bits of Bus Status information on pins ST0-ST3. The various combinations on these pins indicate why the CPU is performing a bus cycle, or, if it is idle on the bus, then why it is idle.

Referring to Figures 16 and 17, note that Bus Status leads the corresponding Bus Cycle, going valid one clock cycle before T1, and changing to the next state at T4. This allows the system designer to fully decode the Bus Status and, if desired, latch the decoded signals before $\overline{\text{ADS}}$ initiates the Bus Cycle.

The Bus Status pins are interpreted as a 4-bit value, with ST0 the least significant bit. Their values decode as follows:

0000 The bus is idle because the CPU does not yet need access to the bus.
0001 The bus is idle because the CPU is executing the WAIT instruction.
0010 (Reserved for future use.)
0011 The bus is idle because the CPU is waiting for a coprocessor to complete an instruction.

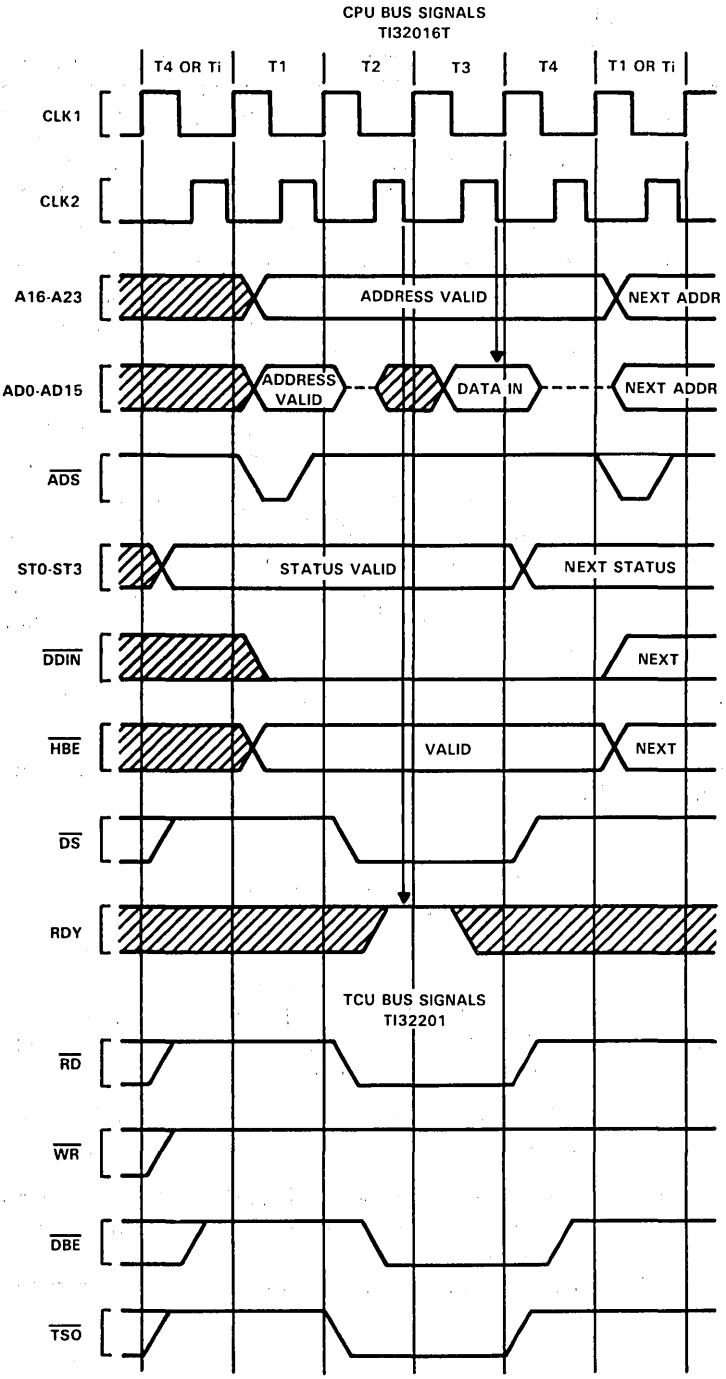


Figure 16. Read Cycle Timing

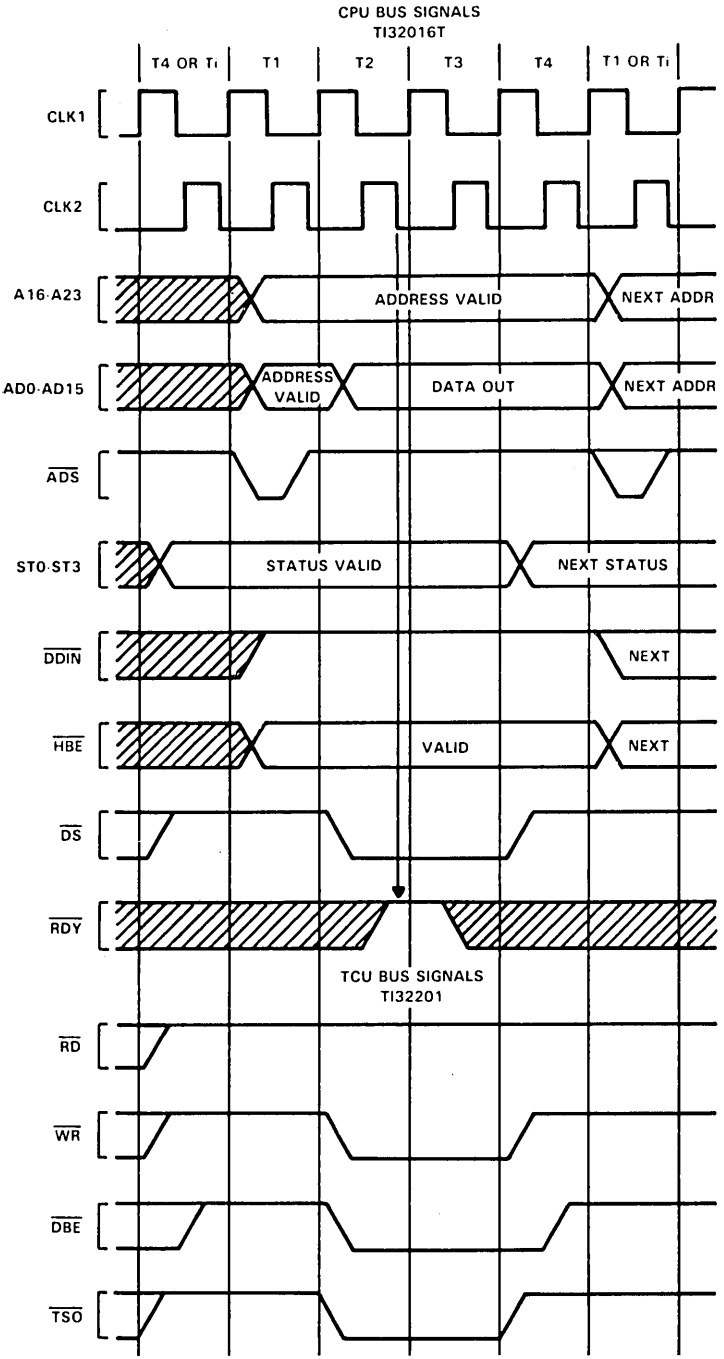


Figure 17. Write Cycle Timing

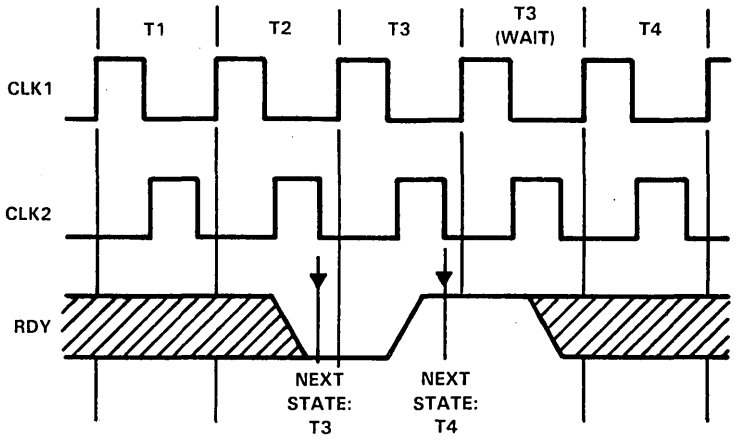


Figure 18. RDY Pin Timing

0100 Master Interrupt Acknowledge. The CPU is performing a read cycle. To acknowledge receipt of a Nonmaskable Interrupt (on $\overline{\text{NMI}}$) it will read from address FFFF00_{16} but will ignore any data provided. To acknowledge receipt of a Maskable Interrupt (on $\overline{\text{INT}}$) it will read from address FFFF00_{16}, expecting a vector number to be provided from the Master TI32202 Interrupt Control Unit (ICU). If the vectoring mode selected by the last SETCFG instruction was nonvectored, then the CPU will ignore the value it has read and will use a default vector instead, having assumed that no TI32202 is present (section 3.2.3.4.5).

0101 Cascaded Interrupt Acknowledge. The CPU is reading a vector number from a Cascaded TI32202 Interrupt Control Unit. The address provided is the address of the TI32202 Hardware Vector register (section 3.2.3.4.5).

0110 Master End of Interrupt. The CPU is performing a Read cycle to indicate that it is executing a Return from Interrupt (RETI) instruction (section 3.2.3.4.5).

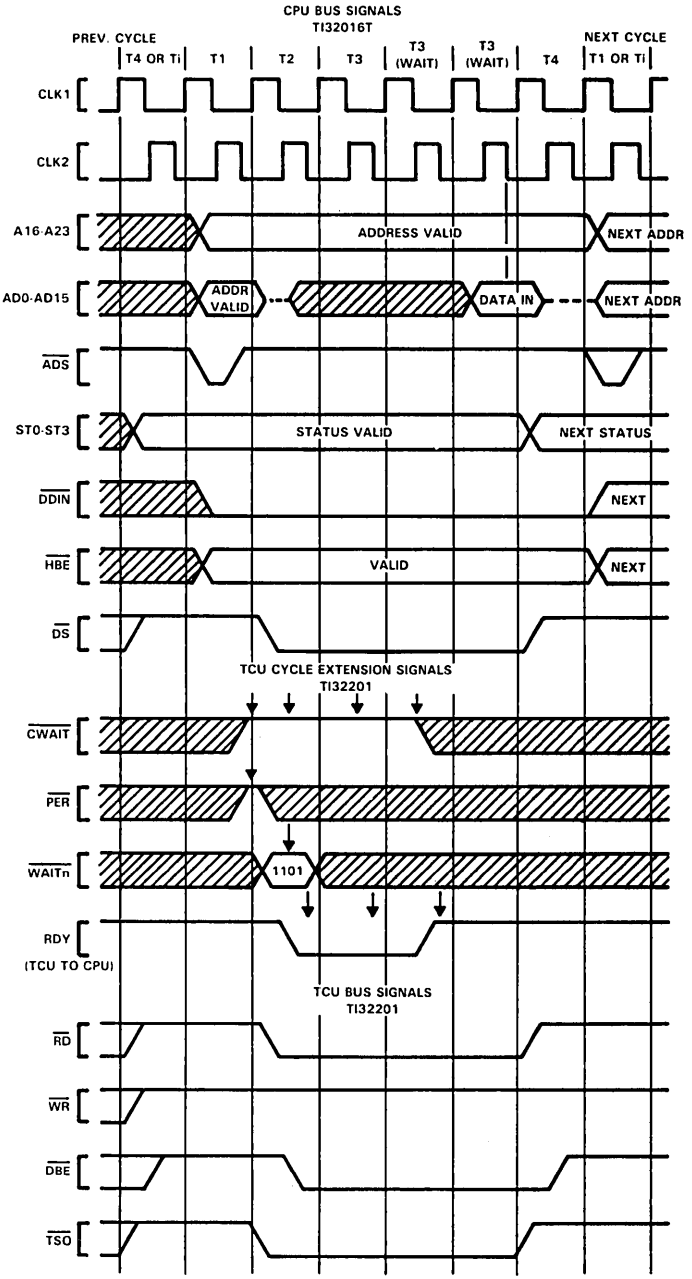
0111 Cascaded End of Interrupt. The CPU is reading from a Cascaded Interrupt Control Unit to indicate that it is returning (through RETI) from an interrupt service routine requested by that unit (section 3.2.3.4.5).

1000 Sequential Instruction Fetch. The CPU is reading the next sequential word from the instruction stream into the Instruction Queue. It will do so whenever the bus would otherwise be idle and the queue is not already full.

1001 Nonsequential Instruction Fetch. The CPU is performing the first fetch of instruction code after the Instruction Queue is purged. This will occur as a result of any jump or branch, or any interrupt or trap, or execution of certain instructions.

1010 Data Transfer. The CPU is reading or writing an operand of an instruction

1011 Read RMW Operand. The CPU is reading an operand which will subsequently be modified and rewritten. If memory protection circuitry would not allow the following write cycle, it must abort this cycle.



NOTE: Arrows on $\overline{\text{CWAIT}}$, $\overline{\text{PER}}$, $\overline{\text{WAITn}}$ indicate points at which the TCU samples. Arrows on AD0-AD15 and RDY indicate points at which the CPU samples.

Figure 19. Extended Cycle Example

1100 Read for Effective Address Calculation. The CPU is reading information from memory in order to determine the Effective Address of an operand. This will occur whenever an instruction uses the Memory Relative or External addressing mode.

1101 Transfer Coprocessor Operand. The CPU is either transferring an instruction operand to or from a coprocessor, or it is issuing the Operation Word of a coprocessor instruction (section 3.2.3.9.1)

1110 Read Coprocessor Status. The CPU is reading a Status Word from a coprocessor. This occurs after the coprocessor has signaled completion of an instruction. The transferred word tells the CPU whether a trap should be taken, and in some instructions it presents new values for the CPU Processor Status Register bits N, Z, L, or F (section 3.2.3.9.1).

1111 Broadcast Coprocessor ID. The CPU is initiating the execution of a coprocessor instruction. The ID Byte (first byte of the instruction) is sent to all coprocessors, one of which will recognize it. From this point the CPU is communicating with only one coprocessor (section 3.2.3.9.1).

3.2.3.4.3 Data Access Sequences

The 24-bit address provided by the TI32016T is a byte address; that is,it uniquely identifies one of up to 16,777,216 eight-bit memory locations. An important feature of the TI32016T is that the presence of a 16-bit data bus imposes no restrictions on the data alignment; any data item, regardless of size, may be placed starting at any memory address. TheTI32016T provides a special control signal, High Byte Enable ($\overline{HBE}$) which facilitates individual byte addressing on a 16-bit bus. Memory is organized as two 8-bit banks, each bank receiving the word address (A1-A23)) in parallel. One bank, connected to Data Bus pins AD0-AD7 is enabled to respond to even byte addresses; i.e., when the least significant address bit (A0) is low. The second bank, connected to data bus pins AD8-AD15, is enabled when $\overline{HBE}$ is low. See Figure 20.

Any bus cycle falls into one of three categories: Even Byte Access, Odd Byte Access, and Even Word Access. All accesses to any data type are made up of sequences of these cycles. Table 3 lists the state of A0 and $\overline{HBE}$ for each category.

Accesses of operands requiring more than one bus cycle are performed sequentially, with no idle T-States separating them. The number of bus cycles required to transfer an operand depends on its size and its alignment. Table 4 lists the bus cycles performed for each situation.

Bit Accesses. The Bit Instructions perform byte accesses to the byte containing the designated bit. The Test and Set Bit instruction (SBIT), for example, reads a byte, alters it, and rewrites it, having changed the contents of the one bit.

Bit Field Accesses. An access to a Bit Field in memory always generates a double word transfer at the address containing the least significant bit of the field. The double word is read by an Extract Instruction; an Insert instruction reads a double word, modifies it, and rewrites it.

Extending Multiply Accesses. The Extending Multiply Instruction (MEI) will return a result which is twice the size in bytes of the operands it reads. If the multiplicand is

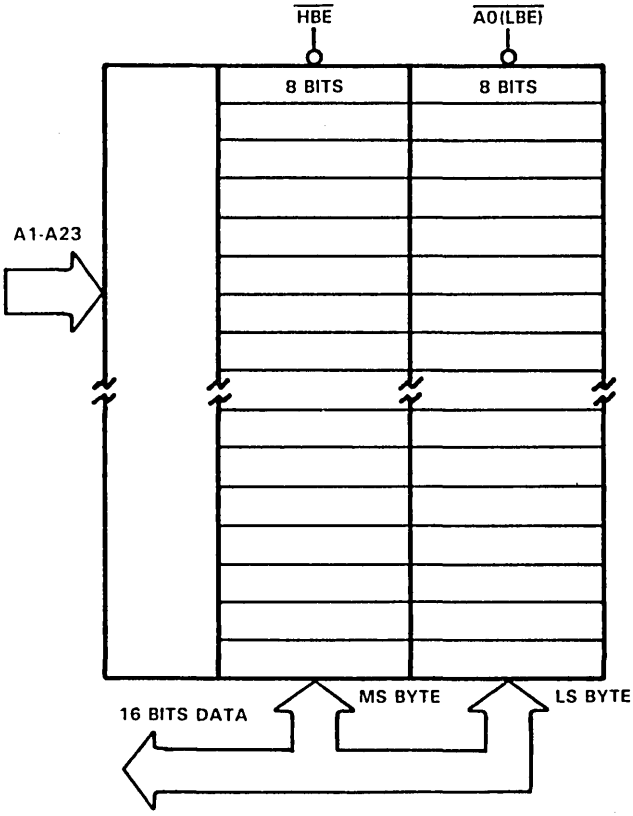


Figure 20. Memory Interface

in memory, the most significant half of the result is written first (at the higher address), then the least significant half. This is done in order to support retry if this instruction is aborted.

3.2.3.4.4 Instruction Fetches.

Instructions for the TI32016T CPU are "prefetched"; that is, they are input before being needed into the next available entry of the 8-byte Instruction Queue. The CPU performs two types of Instruction Fetch cycles: Sequential and Nonsequential. These can be distinguished from each other by their differing status combinations on pins ST0-ST3 (section 3.2.3.4.2)

Table 3. Bus Cycle Categories

CATEGORY	$\overline{HBE}$	A0
Even Byte	1	0
Odd Byte	0	1
Even Word	0	0

Table 4. Access Sequences

CYCLE	TYPE	ADDRESS	$\overline{HBE}$	A0	HIGH BUS	LOW BUS
A. Odd Word Access Sequence						
					BYTE 1 | BYTE 0 ← A	
1	Odd Byte	A	0	1	Byte 0	Don't Care
2	Even Byte	A+1	1	0	Don't Care	Byte 1
B. Even Double-Word Access Sequence						
					BYTE 3 | BYTE 2 | BYTE1 | BYTE 0 ← A	
1	Even Word	A	0	0	Byte 1	Byte 0
2	Even Word	A+2	0	0	Byte 3	Byte 2
C. Odd Double-Word Access Sequence						
					BYTE 3 | BYTE 2 | BYTE 1 | BYTE 0 ← A	
1	Odd Byte	A	0	1	Byte 0	Don't Care
2	Even Word	A+1	0	0	Byte 2	Byte 1
3	Even Byte	A+3	1	0	Don't Care	Byte 3
D. Even Quad-Word Access Sequence						
					BYTE 7 | BYTE 6 | BYTE 5 | BYTE 4 | BYTE 3 | BYTE 2 | BYTE 1 | BYTE 0 ← A	
1	Even Word	A	0	0	Byte 1	Byte 0
2	Even Word	A+2	0	0	Byte 3	Byte 2
Other bus cycles (instruction prefetch or coprocessor) can occur here.						
3	Even Word	A+4	0	0	Byte 5	Byte 4
4	Even Word	A+6	0	0	Byte 7	Byte 6
E. Odd Quad-Word Access Sequence						
					BYTE 7 | BYTE 6 | BYTE 5 | BYTE 4 | BYTE 3 | BYTE 2 | BYTE 1 | BYTE 0 ← A	
1	Odd Byte	A	0	1	Byte 0	Don't Care
2	Even Word	A+1	0	0	Byte 2	Byte 1
3	Even Byte	A+3	1	0	Don't Care	Byte 3
Other bus cycles (instruction prefetch or coprocessor) can occur here.						
4	Odd Byte	A+4	0	1	Byte 4	Don't Care
5	Even Word	A+5	0	0	Byte 6	Byte 5
6	Even Byte	A+7	1	0	Don't Care	Byte 7

A Sequential Fetch will be performed by the CPU whenever the Data Bus would otherwise be idle and the Instruction Queue is not currently full. Sequential Fetches are always Even Word Read cycles (Table 3).

A Nonsequential Fetch occurs as a result of any break in the normally sequential flow of a program. Any jump or branch instruction, a trap or an interrupt will cause the next Instruction Fetch cycle to be Nonsequential. In addition, certain instructions flush the instruction queue, causing the next instruction fetch to display Nonsequential status. Only the first bus cycle after a break displays Nonsequential status, and that cycle is either an Even Word Read or an Odd Byte Read, depending on whether the destination address is even or odd.

3.2.3.4.5 Interrupt Control Cycles

Activating the $\overline{INT}$ or $\overline{NMI}$ pin on the CPU will initiate one or more bus cycles whose purpose is interrupt control rather than the transfer of instructions or data. Execution of the Return from Interrupt instruction (RETI) will also cause Interrupt Control bus cycles. These differ from instruction or data transfers only in the status presented on pins ST0-ST3. All Interrupt Control cycles are single-byte read cycles.

This section describes only the Interrupt Control sequences associated with each interrupt and with the return from its service routine. For full details of the TI32016T interrupt structure, see section 3.2.3.8.

3.2.3.4.6 Coprocessor Communication

In addition to its use as the Address Translation strap (section 3.2.3.5.1), the $\overline{AT}/\overline{SPC}$ pin is used as the Data Strobe for coprocessor transfers. In this role, it is referred to as Coprocessor Control ($\overline{SPC}$). In a coprocessor bus cycle, data is transferred on the Data Bus (AD0-AD15), and the least significant two bits of CPU cycle status (ST0-ST1) are monitored by each coprocessor in order to determine the type of transfer being performed. $\overline{SPC}$ is bidirectional, but is driven by the CPU during all coprocessor bus cycles (section 3.2.3.9) (Figure 21).

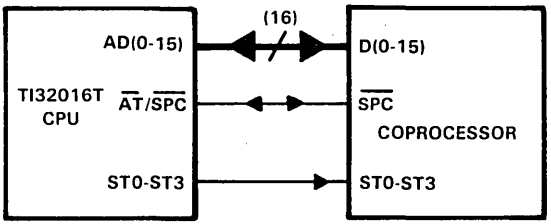


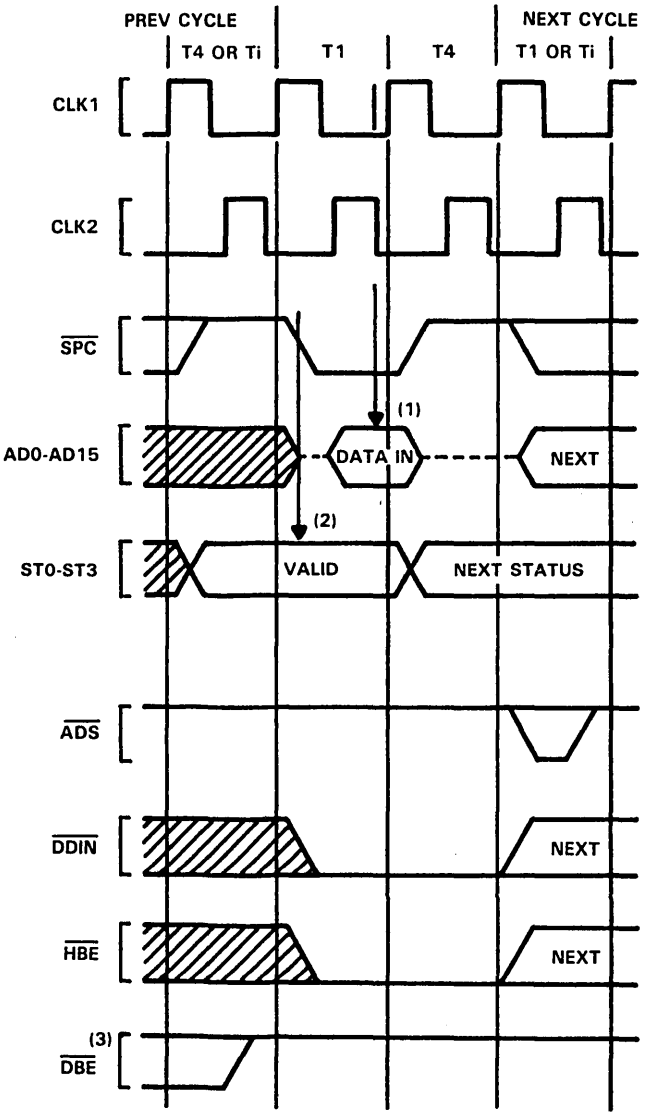
Figure 21. Coprocessor Connections

Coprocessor Bus Cycles. A coprocessor bus cycle always takes exactly two clock cycles, labeled T1 and T4 (see Figures 22 and 23). During a Read cycle, $\overline{SPC}$ is activated at T1, data is sampled at T4, and $\overline{SPC}$ is removed. The Cycle Status pins lead the cycle by one clock period, and are sampled at the leading edge of $\overline{SPC}$. During a Write cycle, the CPU applies data and activates $\overline{SPC}$ at T1, removing $\overline{SPC}$ at T4.

Table 5. Interrupt Sequences

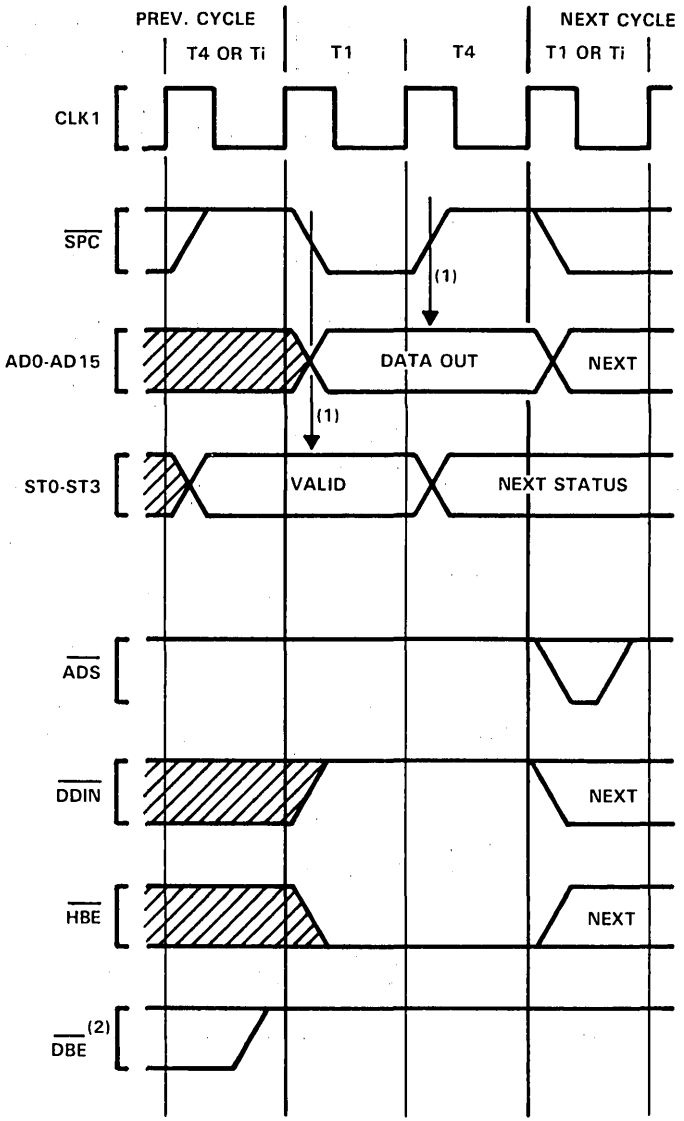
CYCLE	STATUS	ADDRESS	$\overline{DDIN}$	$\overline{HBE}$	A0	HIGH BUS	LOW BUS
A. Non-Maskable Interrupt Control Sequences.							
Interrupt Acknowledge							
1	0100	$FFFF00_{16}$	0	1	0	Don't Care	Don't Care
Interrupt Return							
None: Performed through Return from Trap (RETT) instruction.							
B. Non-Vectored Interrupt Control Sequences.							
Interrupt Acknowledge							
1	0100	$FFFE00_{16}$	0	1	0	Don't Care	Don't Care
Interrupt Return							
None: Performed through Return from Trap (RETT) instruction.							
C. Vectored Interrupt Sequences: Non-Cascaded.							
Interrupt Acknowledge							
1	0100	$FFFE00_{16}$	0	1	0	Don't Care	Vector: Range: 0—127
Interrupt Return							
1	0110	$FFFE00_{16}$	0	1	0	Don't Care	Vector: Same as in Previous Int. Ack. Cycle
D. Vectored Interrupt Sequences: Cascaded.							
Interrupt Acknowledge							
1	0100	$FFFE00_{16}$	0	1	0	Don't Care	Cascade Index: range −16 to −1
(The CPU here uses the Cascade Index to find the Cascade Address.)							
2	0101	Cascade Address	0	1 or† 0	0 or† 1	Vector, range 0—255; on appropriate half of Data Bus for even/odd address	
Interrupt Return							
1	0110	$FFFE00_{16}$	0	1	0	Don't Care	Cascade Index: same as in previous Int. Ack. Cycle
(The CPU here uses the Cascade Index to find the Cascade Address.)							
2	0111	Cascade Address	0	1 or 0†	0 or 1†	Don't Care	Don't Care

† If the Cascaded ICU Address is Even (A0 is low), then the CPU applies $\overline{HBE}$ high and reads the vector number from bits 0—7 of the Data Bus. If the address is Odd (A0 is high), then the CPU applies $\overline{HBE}$ low and reads the vector number from bits 8—15 of the Data Bus. The vector number may be in the range 0—255.



NOTES: 1. CPU samples Data Bus here.
2. Coprocessor samples CPU Status here.
3. $\overline{DBE}$ and all other TI32201 TCU bus signals remain inactive because no $\overline{ADS}$ pulse is received from the CPU.

Figure 22. CPU Read from Coprocessor



NOTES: 1. Arrows indicate points at which the Coprocessor samples.
2. $\overline{DBE}$, being provided by the TI32201 TCU, remains inactive due to the fact that no pulse is presented on $\overline{ADS}$. $\overline{TCU}$ signals $\overline{RD}$, $\overline{WR}$ and $\overline{TSO}$ also remain inactive.

Figure 23. CPU Write to Coprocessor

The coprocessor latches status on the leading edge of $\overline{SPC}$ and latches data on the trailing edge. Since the CPU does not pulse the Address Strobe ($\overline{ADS}$), no bus signals are generated by the TI32201 Timing Control Unit. The direction of a transfer is determined by the sequence ("protocol") established by the instruction under execution; but the CPU indicates the direction on the $\overline{DDIN}$ pin for hardware debugging purposes.

Operand Transfer Sequences. A coprocessor operand is transferred in one or more coprocessor bus cycles. A Byte operand is transferred on the least significant byte of the Data Bus (AD0-AD7), and a Word operand is transferred on bits AD0-AD15. A double word is transferred in a consecutive pair of bus cycles, least significant word first. A quad word is transferred in two pairs of coprocessor cycles, with other bus cycles possibly occurring between them. The word order is from least signficant word to most significant word.

3.2.3.5 Memory-Management Option

The TI32016T CPU, in conjunction with the TI32082W Memory Management Unit (MMU), provides full support for address translation, memory protection, and memory allocation techniques up to and including Demand-Paged Virtual Memory.

3.2.3.5.1 Address-Translation Strap

The Bus Interface Control section of the TI32016T CPU has two bus timing modes; with or without address-translation. The mode of operation is selected by the CPU by sampling the $\overline{AT}/\overline{SPC}$ (Address Translation/Coprocessor Control) pin on the rising edge of the Reset ($\overline{RST}$) pulse. If $\overline{AT}/\overline{SPC}$ is sampled as high, the bus timing is as previously described in section 3.2.3.4. If it is sampled as low, two changes occur:

1. An extra clock cycle, Tmmu, is inserted into all bus cycles except coprocessor transfers.
2. The $\overline{DS}/\overline{FLT}$ pin changes in function from a Data Strobe output ($\overline{DS}$) to a Float Command input ($\overline{FLT}$). The TI32082 MMU will itself pull the CPU $\overline{AT}/\overline{SPC}$ pin low when it is reset. In Nonmemory Managed systems, this pin should be pulled up to V_{CC} through a 10 kΩ resistor.

Note that the Address Translation strap does not specifically declare the presence of a TI32082W MMU, but only the presence of external address translation circuitry. MMU instructions will still trap as being undefined unless the SETCFG (Set Configuration) instruction is executed to declare the MMU instruction set valid.

3.2.3.5.2 Translated Bus Timing

Figures 24 and 25 illustrate the CPU activity during a read cycle and a write cycle in Address Translation mode. The additional T-State, Tmmu, is inserted between T1 and T2. During this time the CPU places AD0-AD15 and A16-A23 into the 3-state (high-impedance state) mode, allowing the MMU to assert the translated address and issue the physical address strobe $\overline{PAV}$. T2 through T4 of the cycle are identical to their counterparts without Address Translation, with the exception that the CPU address lines A16-A23 remain in the 3-state condition. This allows the MMU to continue asserting the translated address on those pins.

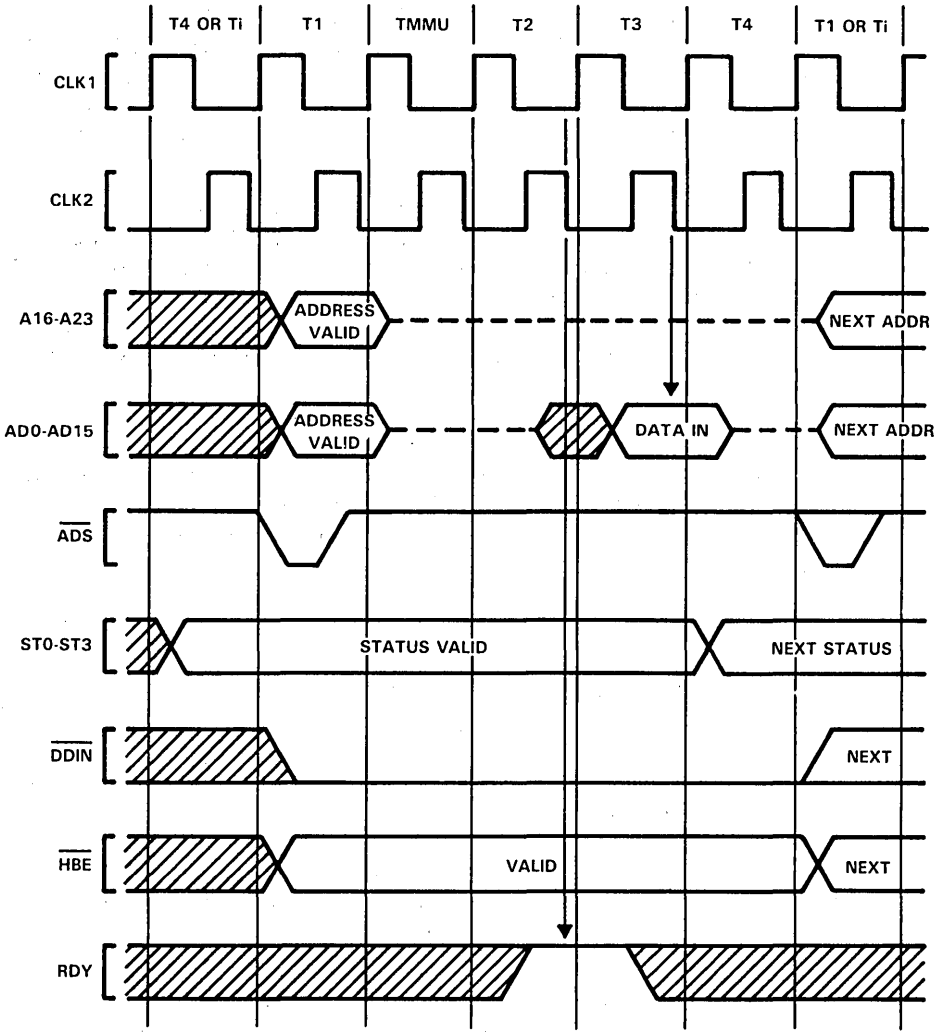


Figure 24. Read Cycle with Address Translation (CPU Action)

Figures 26 and 27 show a read cycle and a write cycle as generated by the TI32016T/TI32082W/TI32201 group. Note that with the CPU $\overline{ADS}$ signal going to the MMU, and with the MMU $\overline{PAV}$ signal substituting for $\overline{ADS}$ every where else, Tmmu through T4 look exactly like T1 through T4 in a nonmemory managed system. For the connection diagram (Figure 28).

3.2.3.5.3 The $\overline{FLT}$ (Float) Pin

In Address Translation mode, the $\overline{DS}$/$\overline{FLT}$ pin is treated as the input command $\overline{FLT}$ (Float). Activating $\overline{FLT}$ during Tmmu causes the CPU to wait longer than Tmmu for

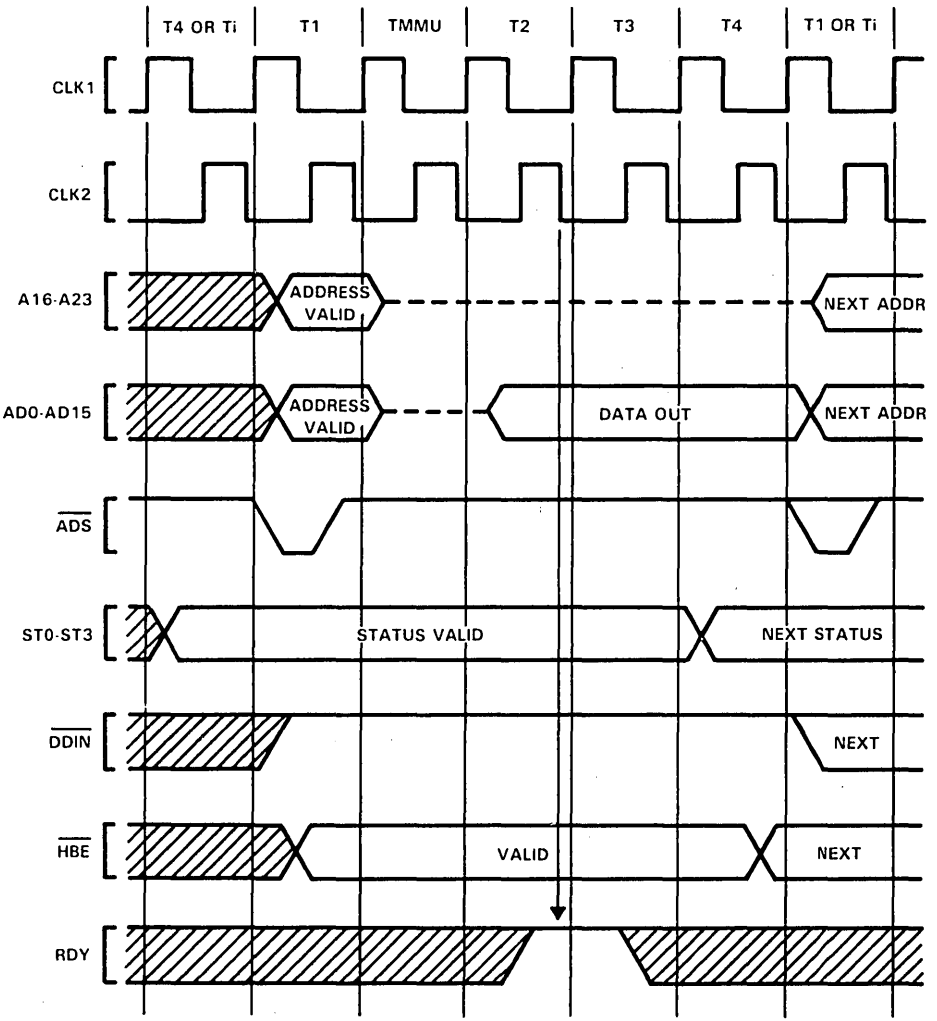


Figure 25. Write Cycle with Address Translation (CPU Action)

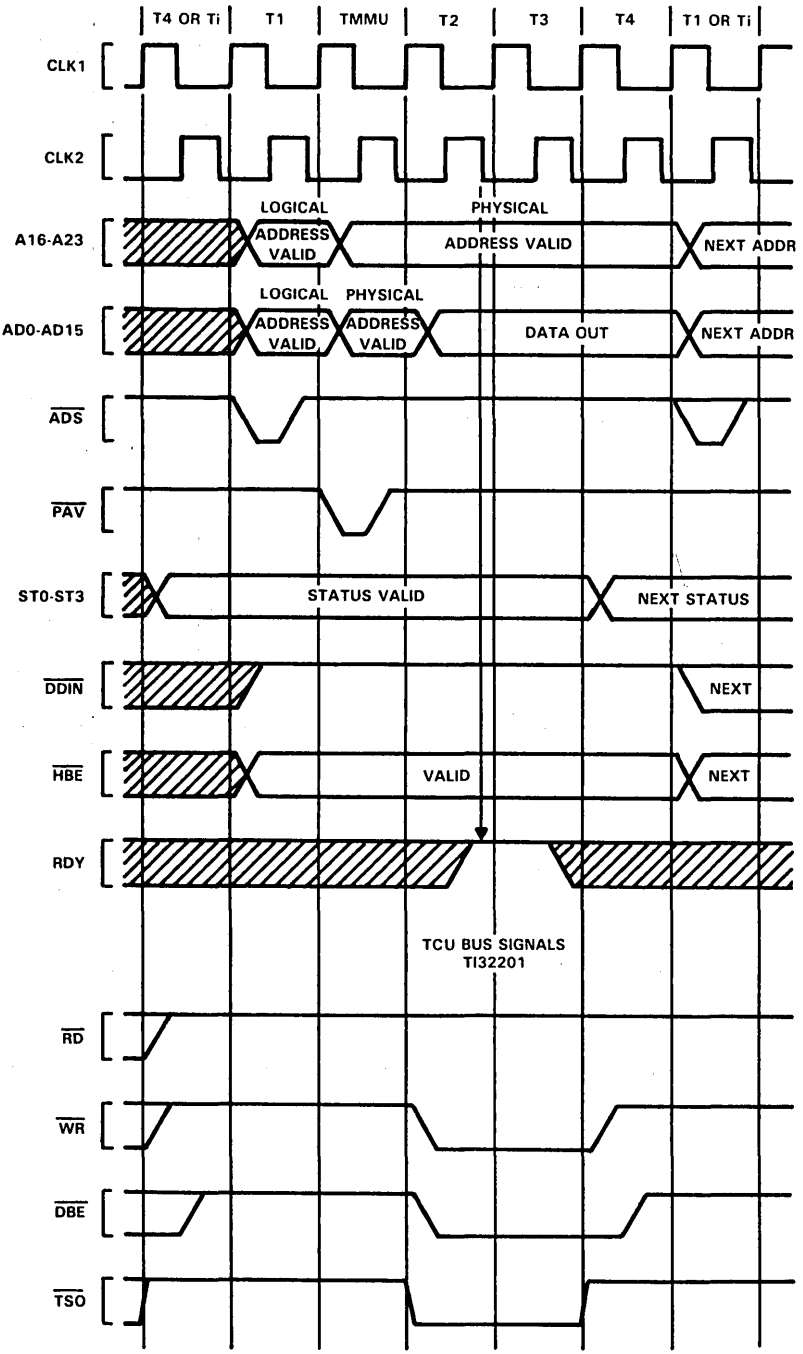


Figure 26. Memory-Managed Read Cycle

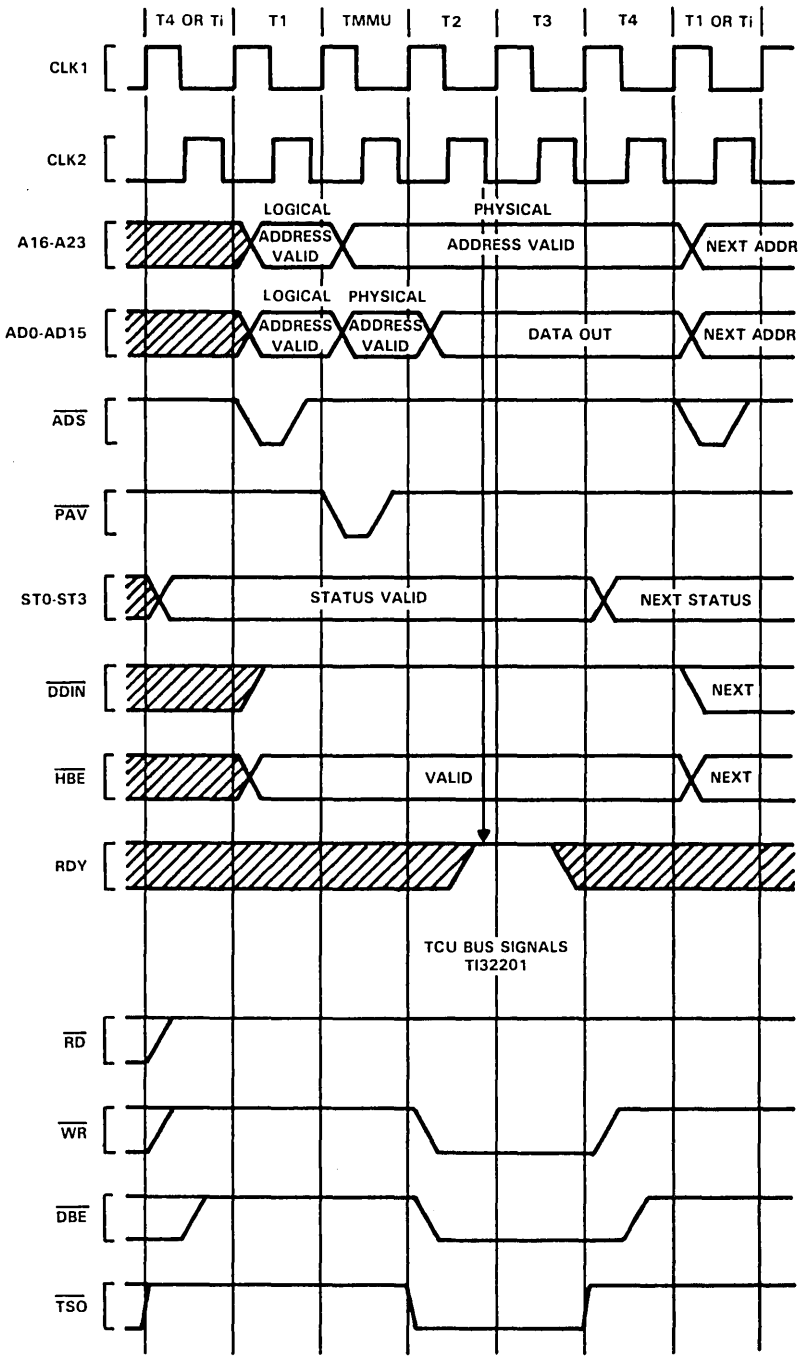
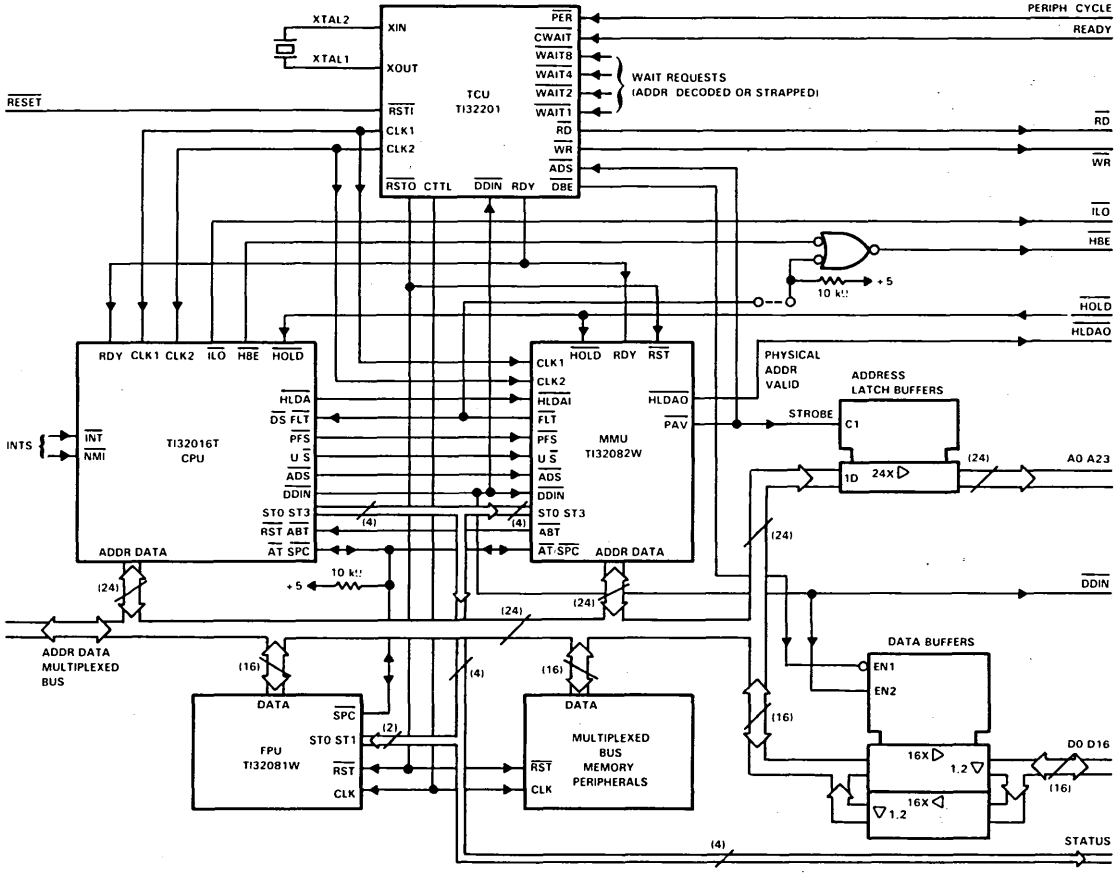


Figure 27. Memory-Managed Write Cycle



NOTE: The AND gate on the $\overline{HBE}$ signal line is not necessary for the current version of the TI32016T, since the CPU forces $\overline{HBE}$ low whenever $\overline{FLT}$ is asserted. However, it is needed for future higher speed versions of the CPU, since $\overline{HBE}$ will not be affected by $\overline{FLT}$ any longer. The jumper is needed in a system that has to work with or without MMU.

Figure 28. System Connection Diagram

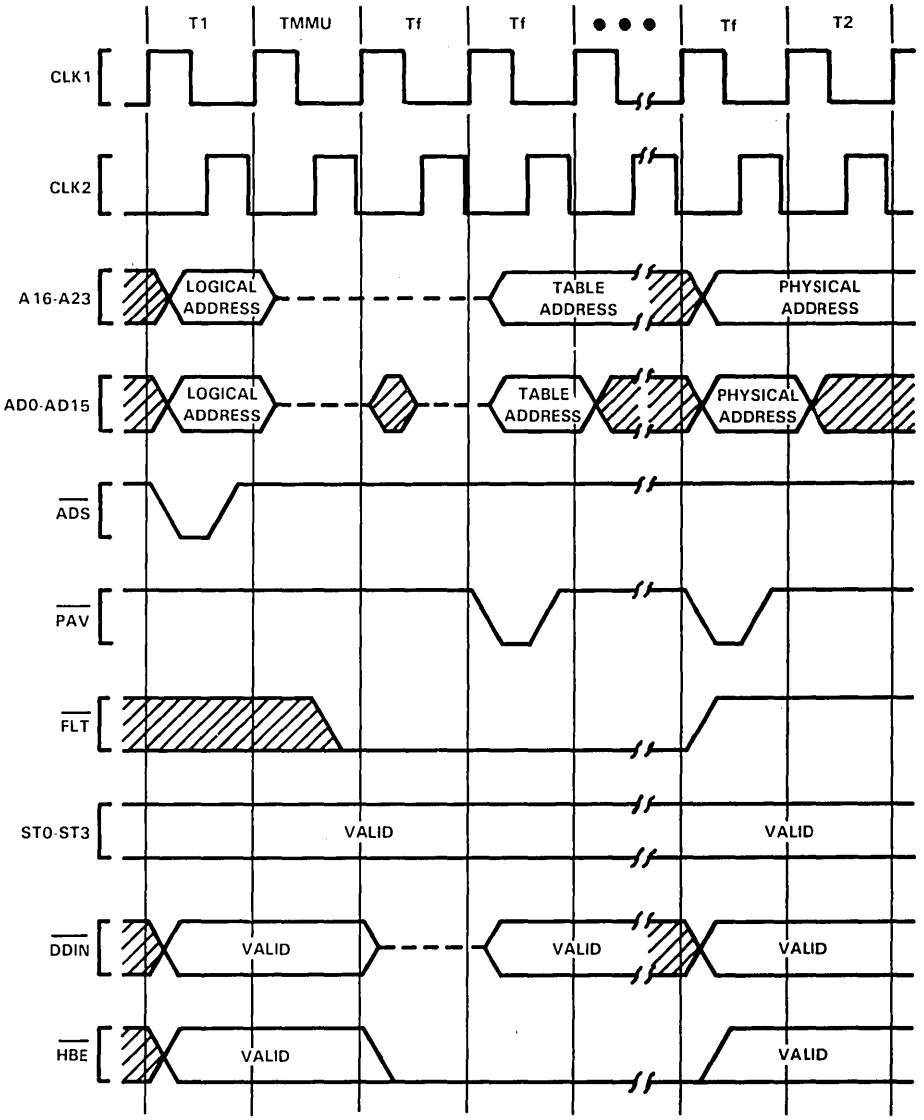


Figure 29. $\overline{\text{FLT}}$ Float Command Timing

address translation and validation. This feature is used occasionally by the TI32082 MMU in order to update its internal translation cache from page tables in memory, or to update certain status bits within them.

Figure 29 shows the effects of $\overline{\text{FLT}}$. Upon sampling $\overline{\text{FLT}}$ low late in Tmmu, the CPU enters idle T-States (Tf) during which it:

1. Sets AD0-AD15, A16-A23, and $\overline{\text{DDIN}}$ to the 3-state (high-impedance) condition (Floating).

2. Sets $\overline{HBE}$ low.
3. Suspends further internal processing of the current instruction. This ensures that the current instruction remains abortable with retry. (See $\overline{RST}/\overline{ABT}$ description, section 3.2.3.5.4)

Note that the AD0—AD15 pins may be briefly asserted during the first idle T-State. The above conditions remain in effect until $\overline{FLT}$ again goes high.

3.2.3.5.4 Aborting Bus Cycles

The $\overline{RST}/\overline{ABT}$ pin, apart from its reset function (section 3.2.3.3), also serves as the means to "abort", or cancel, a bus cycle and the instruction,if any, which initiated it. An Abort request is distinguished from a Reset in that the $\overline{RST}/\overline{ABT}$ pin is held active for only one clock cycle. If $\overline{RST}/\overline{ABT}$ is pulled low during Tmmu or Tf, this signals that the cycle must be aborted. The CPU itself will enter T2 and then Ti, thereby terminating the cycle. Since it is the MMU $\overline{PAV}$ signal which triggers a physical cycle, the rest of the system remains unaware that a cycle was even started.

The TI32082W MMU will abort a bus cycle for either of two reasons:

1. The CPU is attempting to access a logical address which is not currently resident in physical memory. The referenced page must be brought into physical memory from mass storage to make it accessible to the CPU.
2. The CPU is attempting to perform an access which is not allowed due to the protection level assigned to that page.

When a bus cycle is aborted by the MMU, the instruction which caused it to occur is also aborted in such a manner that it is guaranteed to be reexecutable later. The information that is changed irrecoverably by such a partly executed instruction does not affect its reexecution.

The Abort Interrupt. Upon aborting an instruction, the CPU immediately performs an interrupt through the ABT vector in the Interrupt Table. The Return Address pushed on the Interrupt Stack is the address of the aborted instruction, such that a Return from Trap (RETT) instruction will automatically retry it.

The one exception to this sequence occurs if the aborted bus cycle was an instruction prefetch. If so, it is not yet certain that the aborted prefetch code is to be executed. Instead of causing an interrupt, the CPU only aborts the bus cycle, and stops prefetching. If the information in the instruction Queue runs out, meaning that the instruction will actually be executed, the ABT interrupt will occur, in effect aborting the instruction that was being fetched.

Hardware Considerations. In order to guarantee instruction retry, certain rules must be followed in applying an Abort to the CPU. These rules are followed by the TI32082W MMU.

1. If $\overline{FLT}$ has not been applied to the CPU, the Abort pulse must occur during or before Tmmu.
2. If $\overline{FLT}$ has been applied to the CPU, the Abort pulse must be applied before the T-State in which $\overline{FLT}$ goes inactive. The CPU will not actually respond to the Abort command until $\overline{FLT}$ is removed.

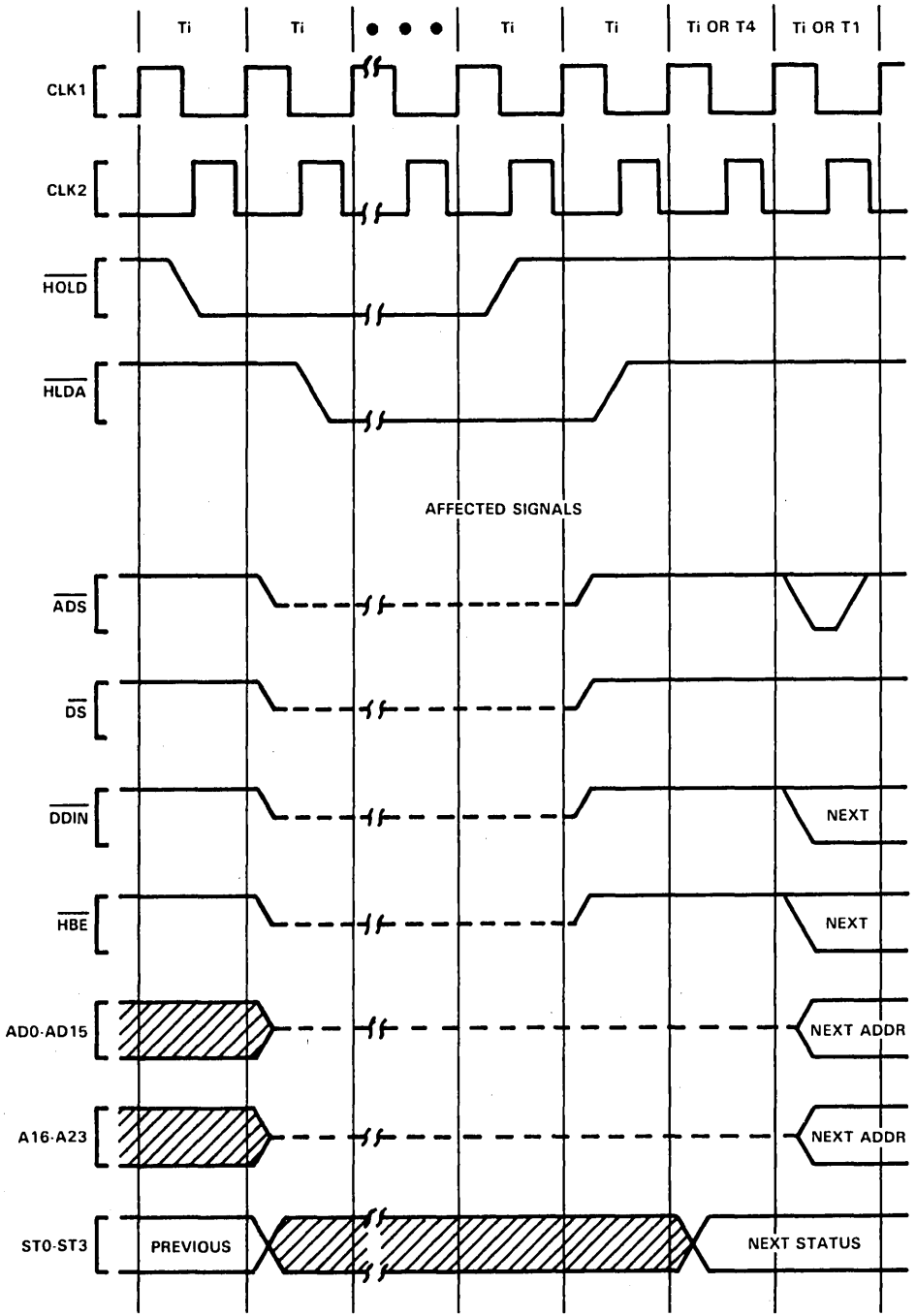


Figure 30. $\overline{\text{HOLD}}$ Timing, Bus Initially Idle

3. The Write half of a Read-Modify-Write operand access may not be aborted. The CPU guarantees that this will never be necessary for Memory Management functions by applying a special RMW status (Status Code 1011) during the Read half of the access. When the CPU presents RMW status, that cycle must be aborted if it would be illegal to write to any of the accessed addresses.

If $\overline{RST}/\overline{ABT}$ is pulsed at any time other than as indicated above, it will abort either the instruction currently under execution or the next instruction and will act as a very high-priority interrupt. However, the program which was running at the time is not guaranteed recoverable.

3.2.3.6 Bus Access Control

The TI32016T CPU has the capability of relinquishing its access to the bus upon request from a DMA device or another CPU. This capability is implemented on the $\overline{HOLD}$ (Hold Request) and $\overline{HLDA}$ (Hold Acknowledge) pins. By asserting $\overline{HOLD}$ low, an external device requests access to the bus. On receipt of $\overline{HLDA}$ from the CPU, the device may perform bus cycles, as the CPU at this point has set AD0-AD15, A16-A23, $\overline{ADS}$, $\overline{DDIN}$,and $\overline{HBE}$ pins to the 3-state condition. To return control of the bus to the CPU, the device sets $\overline{HOLD}$ inactive, and the CPU acknowledges return of the bus by setting $\overline{HLDA}$ inactive.

How quickly the CPU releases the bus depends on whether it is idle on the bus at the time the $\overline{HOLD}$ request is made, as the CPU must always complete the current bus cycle. Figure 30 shows the timing sequence when the CPU is idle. In this case, the CPU grants the bus during the immediately following clock cycle. Figure 31 shows the sequence if the CPU is using the bus at the time that the $\overline{HOLD}$ request is made. If the request is made during or before the clock cycle shown (two clock cycles before T4), the CPU will release the bus during the clock cycle following T4. If the request occurs closer to T4, the CPU may already have decided to initiate another bus cycle. In that case it will not grant the bus until the next T4 state. Note that this situation will also occur if the CPU is idle on the bus but has initiated a bus cycle internally.

In Memory Managed systems, the $\overline{HLDA}$ signal is connected in a daisy chain through the TI32082W, such that the MMU can release the bus if it is using it.

3.2.3.7 Instruction Status

In addition to the four bits of Bus Cycle Status ($\overline{ST0}$-$\overline{ST3}$), the TI32016T CPU also presents Instruction Status information on three separate pins. These pins differ from ST0-ST3 in that they are synchronous to the CPU's internal instruction execution section rather than to its bus interface section.

$\overline{PFS}$ (Program Flow Status) is pulsed low as each instruction begins execution. It is intended for debugging purposes, and is used that way by the TI32082W MMU.

U/$\overline{S}$ originates from the U bit of the Processor Status Register, and indicates whether the CPU is currently running in User or Operating System mode. It is sampled by the MMU for mapping, protection, and debugging purposes. Although it is not synchronous to bus cycles, there are guarantees on its validity during any given bus cycle.

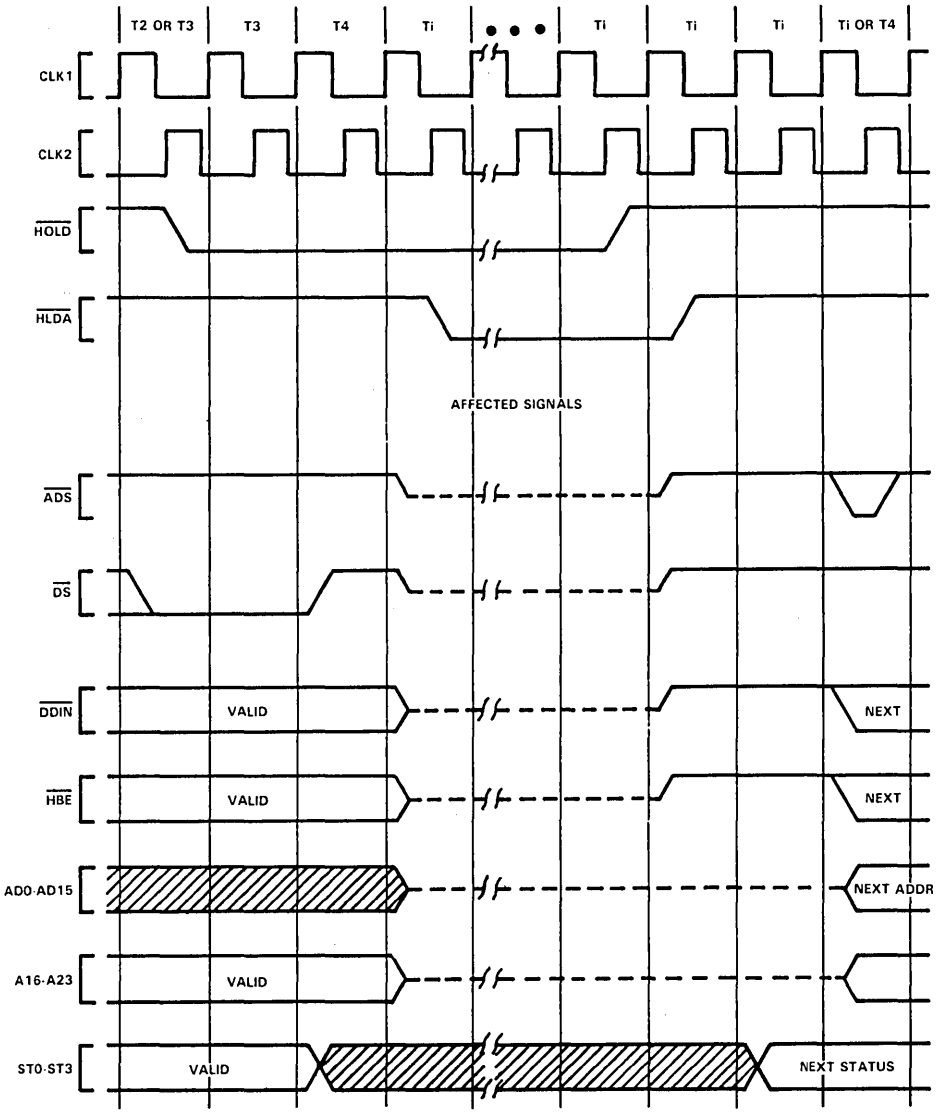


Figure 31. $\overline{\text{HOLD}}$ Timing, Bus Initially Not Idle

$\overline{\text{ILO}}$ (Interlocked Operation) is activated during a Set Bit Interlocked (SBITI) or Clear Bit Interlocked (CBITI) instruction. It is made available to external bus arbitration circuitry in order to allow these instructions to implement the semaphore primitive operations for multiprocessor communication and resource sharing. As with the U/$\overline{\text{S}}$ pin, there are guarantees on it is validity during the operand accesses performed by the instructions.

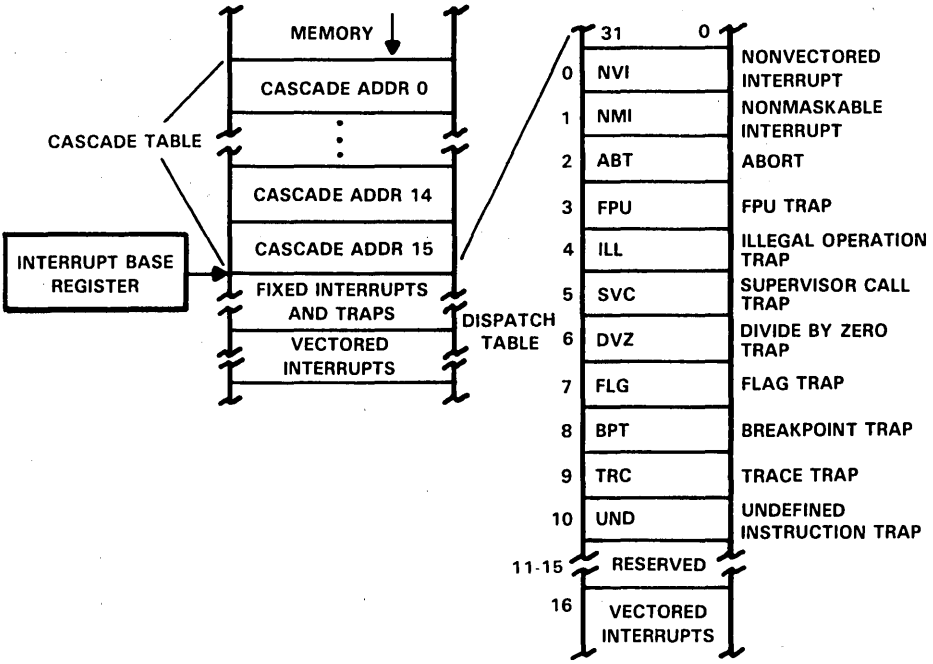


Figure 32. Interrupt Dispatch and Cascade Tables

3.2.3.8 TI32016T Interrupt Structure

1. $\overline{\text{INT}}$, on which maskable interrupts may be requested.
2. $\overline{\text{NMI}}$, on which nonmaskable interrupts may be requested, and
3. $\overline{\text{RST}}/\overline{\text{ABT}}$, which may be used to abort a bus cycle and any associated instruction. It generates an interrupt request if an instruction was aborted (section 3.2.3.5.4).

In addition, there is a set of internally generated "traps" which cause interrupt service to be performed as a result of exceptional conditions (e.g., attempted division by zero) or of specific instructions whose purpose is to cause a trap to occur (e.g., the Supervisor Call instruction).

3.2.3.8.1 General Interrupt/Trap Sequence

Upon receipt of an interrupt or trap request, the CPU goes through four major steps:

1. Adjustment of Registers. Depending on the source of the interrupt or trap, the CPU may restore and/or adjust the contents of the Program Counter (PC), the Processor Status Register (PSR), and the currently selected Stack Pointer (SP). A copy of the PSR is made, and the PSR is then set to reflect Operating System Mode and selection of the Interrupt Stack.
2. Saving Processor Status. The PSR copy is pushed onto the Interrupt Stack as a 16-bit quantity.

3. Vector Acquisition. A vector is either obtained from the Data Bus or is supplied by default.
4. Service Call. The Vector is used as an index into the Interrupt Dispatch Table, whose base address is taken from the CPU Interrupt Base (INTBASE) Register. See Figure 32. A 32-bit External Procedure Call is read from the table entry, and an External Procedure Call is performed using it. The MOD Register (16 bits) and Program Counter (32 bits) are pushed on the Interrupt Stack.

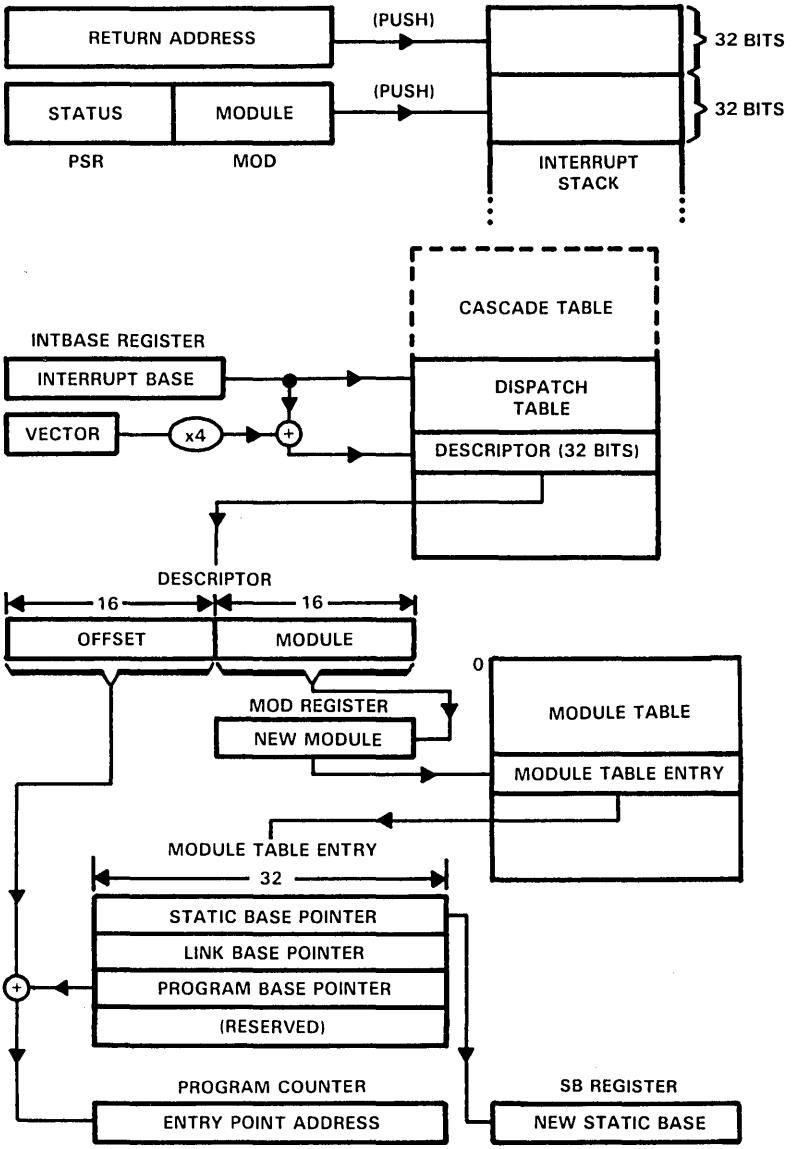


Figure 33. Interrupt/Trap Service Routine Calling Sequence

This process is illustrated in Figure 33, from the viewpoint of the programmer.

Full sequences of events in processing interrupts and traps may be found in section 3.2.3.8.7.

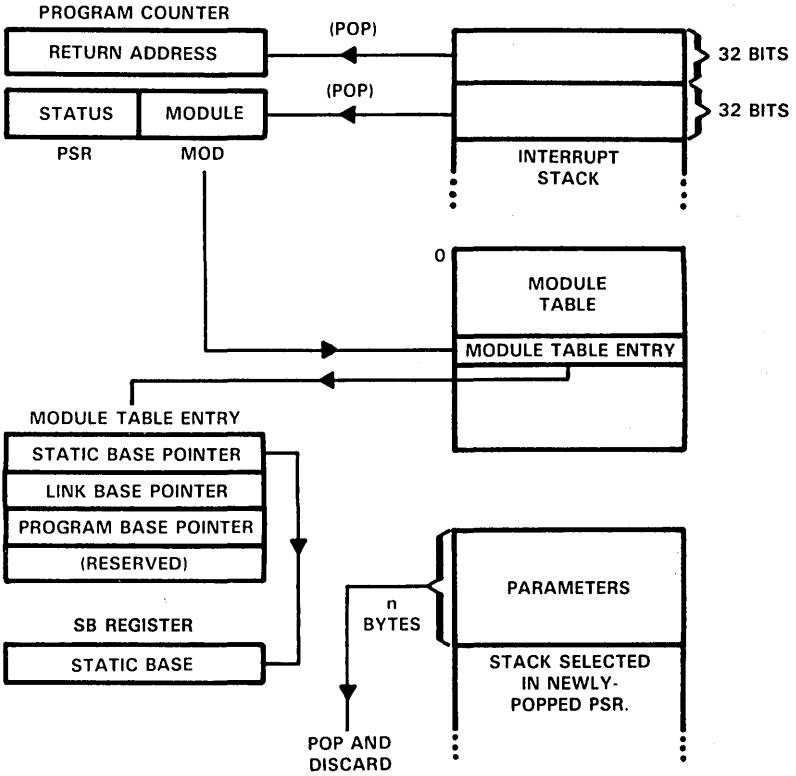


Figure 34. Return from Trap (RETTn) Instruction Flow

3.2.3.8.2 Interrupt/Trap Return

To return to an interrupted program, one of two instructions is used. The RETT (Return from Trap) instruction (Figure 34) restores the PSR, MOD, PC, and SB registers to their previous contents and, since traps are often used deliberately as a call mechanism for Operating System Mode procedures, it also discards a specified number of bytes from the original stack as surplus parameter space. RETT is used to return from any trap or interrupt except the Maskable Interrupt. For this, the RETI (Return from interrupt) instruction is used, which also informs any external Interrupt Control Units that interrupt service has completed. Since interrupts are generally asynchronous external events, RETI does not pop parameters. See Figure 35.

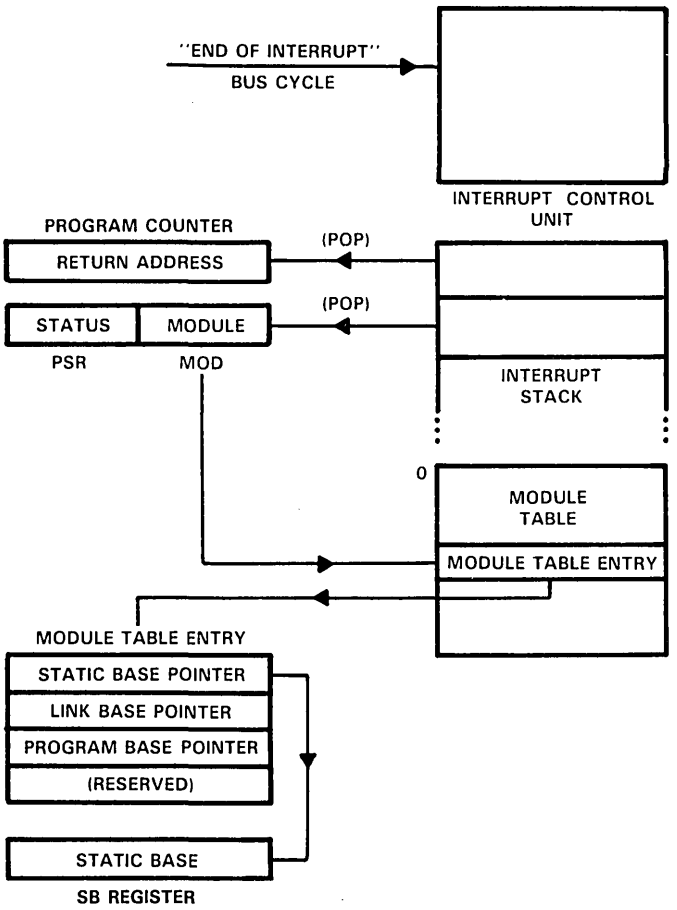


Figure 35. Return from Interrupt (RETI) Instruction Flow

3.2.3.8.3 Maskable Interrupts ($\overline{INT}$ pin)

The $\overline{INT}$ pin is a level-sensitive input. A continuous low level is allowed for generating multiple interrupt requests. The input is maskable, and is therefore enabled to generate interrupt requests only while the Processor Status Register I bit is set. The I bit is automatically cleared during service of an $\overline{INT}$, $\overline{NMI}$, or Abort request, and is restored to its original setting upon return from the interrupt service routine via the RETT or RETI instruction.

The $\overline{INT}$ pin may be configured via the SETCFG instruction as either Nonvectored (CFG Register bit I = 0) or Vectored (CFG Register bit I = 1).

Nonvectored Mode. In the nonvectored mode, an interrupt request on the $\overline{INT}$ pin will cause an Interrupt Acknowledge bus cycle, but the CPU will ignore any value read from the bus and use instead a default vector of zero. This mode is useful for small systems in which hardware interrupt prioritization is unnecessary.

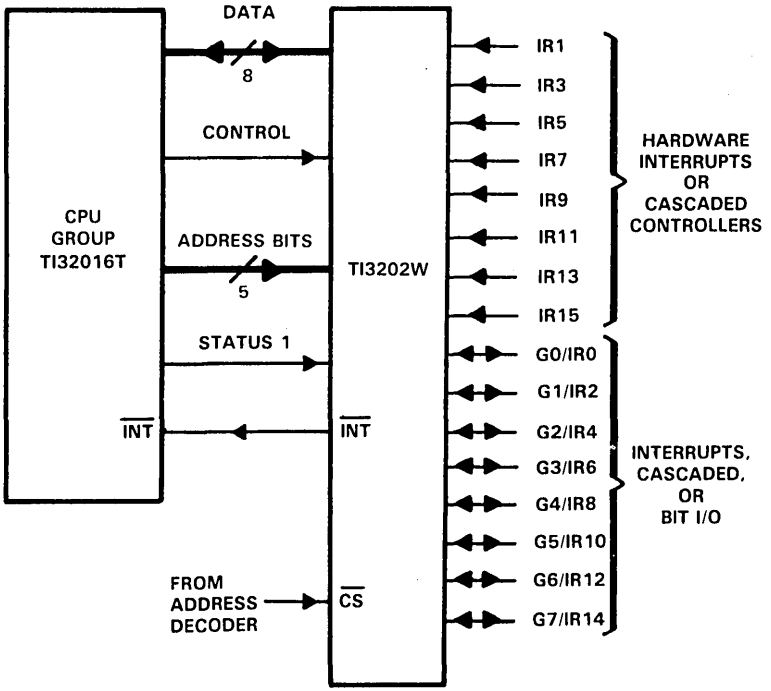


Figure 36. Interrupt Control Unit Connections (16 Levels)

Vectored Mode: Noncascaded Case. In the Vectored mode, the CPU uses a TI32202W Interrupt Control Unit(ICU) to prioritize up to 16 interrupt requests. See Figure 36. Upon receipt of an interrupt request on the $\overline{INT}$ pin, the CPU performs an "Interrupt Acknowledge, Master" bus cycle (section 3.2.3.4.2) reading a vector value from the low-order byte of the Data Bus. This vector is then used as an index into the Dispatch Table in order to find the External Procedure Descriptor for the proper interrupt service procedure. The service procedure eventually returns via the Return from Interrupt (RETI) instruction, which performs an End of Interrupt bus cycle, informing the ICU that it may reprioritize any interrupt requests still pending. The ICU provides the vector number again, which the CPU uses to determine whether it also needs to inform a Cascaded ICU.

In a system with only one ICU (16 levels of interrupt), the vectors provided must be in the range of 0 through 127; that is, they must be positive numbers in eight bits. By providing a negative vector number, an ICU flags the interrupt source as being a Cascaded ICU.

Vectored Mode: Cascaded Case. In order to allow up to 256 levels of interrupt, provision is made both in the CPU and in the TI32202W ICU to transparently support cascading. Figure 37 shows a typical cascaded configuration. Note that the Interrupt output from the Cascaded ICU goes to an Interrupt Request input of the Master ICU, which is the only ICU which drives the CPU $\overline{INT}$ pin.

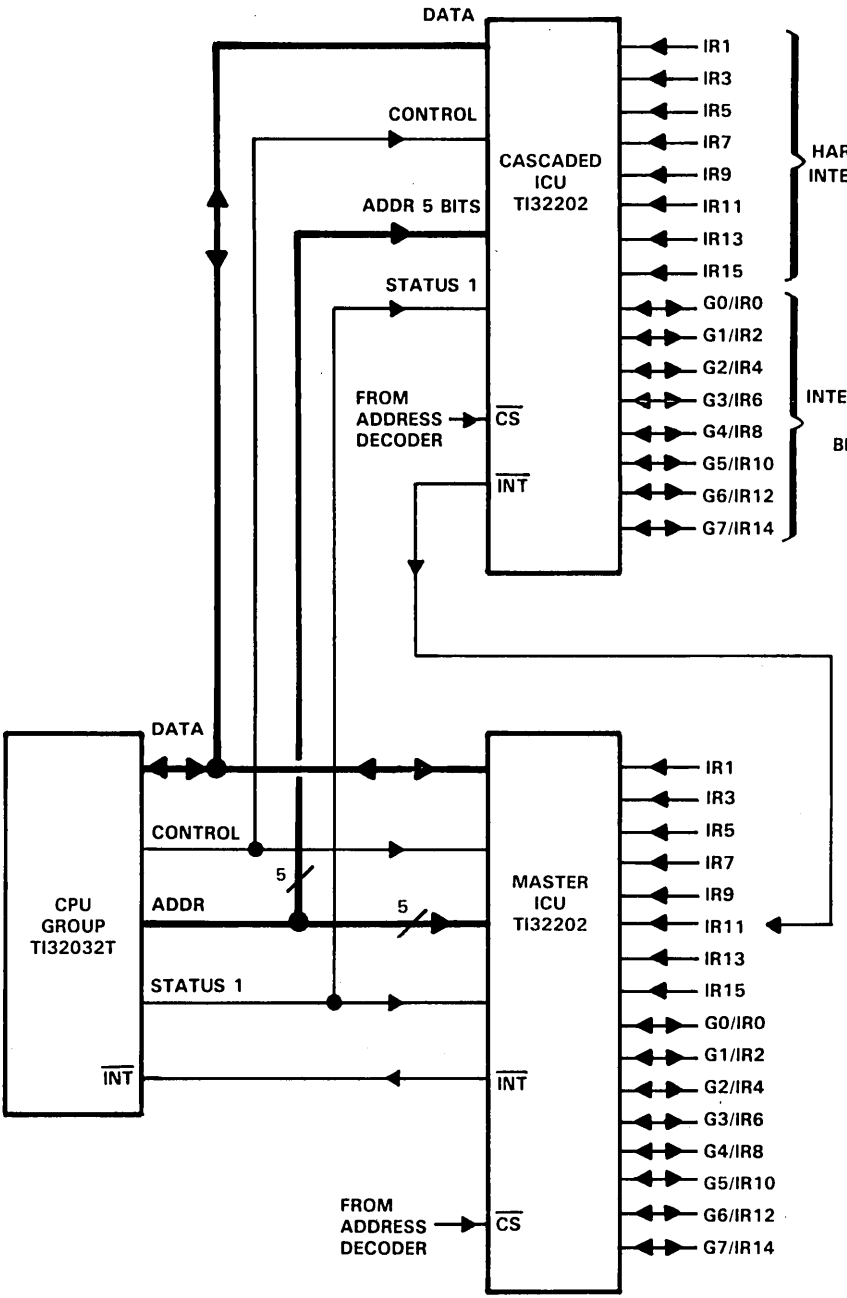


Figure 37. Cascaded Interrupt Control Unit Connections

In a system which uses cascading, two tasks must be performed upon initialization:

1. For each Cascaded ICU in the system, the Master ICU must be informed of the line number (0-15) on which it receives the cascaded requests.
2. A Cascade Table must be established in memory. The Cascade Table is located in a Negative direction from the location indicated by the CPU Interrupt Base (INTBASE) Register. Its entries are 32-bit addresses, pointing to the Vector Registers of each of up to 16 Cascaded ICUs.

Figure 32 illustrates the position of the Cascade Table. To find the Cascade Table entry for a Cascaded ICU, take its Master ICU line number (0-15) and subtract 16 from it, giving an index in the range −16 to −1. Multiply this value by 4 and add the resulting negative number to the contents of the INTBASE Register. The 32-bit entry at this address must be set to the address of the Hardware Vector Register of the Cascaded ICU. This is referred to as the "Cascade Address".

Upon receipt of an interrupt request from a Cascaded ICU, the Master ICU interrupts the CPU and provides the negative Cascade Table index instead of of a (positive) vector number. The CPU, seeing the negative value, uses it as an index into the Cascade Table and reads the Cascade Address from the referenced entry. Applying this address, the CPU performs an "Interrupt Acknowledge, Cascaded" bus cycle (section 3.2.3.4.2), reading the final vector value. This vector is interpreted by the CPU as an unsigned byte, and can therefore be in the range of 0 through 255.

In returning from a Cascaded Interrupt, the service procedure executes the Return from Interrupt (RETI) instruction, as it would for any Maskable Interrupt. The CPU performs an "End of Interrupt, Master" bus cycle,(section 3.2.3.4.2) whereupon the Master ICU again provides the negative Cascade Table index. The CPU, seeing a negative value, uses it to find the corresponding Cascade Address from the Cascade Table. Applying this address, it performs an "End of Interrupt, Cascaded" bus cycle, informing the Cascaded ICU of the completion of the service routine. The byte read from the Cascaded ICU is discarded.

3.2.3.8.4 Nonmaskable Interrupt ($\overline{\text{NMI}}$ pin)

The Nonmaskable Interrupt is triggered whenever a falling edge is detected on the $\overline{\text{NMI}}$ pin. The CPU performs an "Interrupt Acknowledge" bus cycle when processing of this interrupt actually begins. The Interrupt Acknowledge cycle differs from that provided for Maskable Interrupts in that the address presented is $FFFF00_{16}$. The vector value used for the Nonmaskable Interrupt is taken as 1, regardless of the value read from the bus.

The service procedure returns from the Nonmaskable Interrupt using the Return from Trap (RETT) instruction. No special bus cycles occur on return.

For the full sequence of events in processing the Nonmaskable Interrupt, see section 3.2.3.8.7.

3.2.3.8.5 Traps

A trap is an internally generated interrupt request caused as a direct and immediate result of the execution of an instruction. The Return Address pushed by any trap except Trace (TRC) is the address of the first byte of the instruction during which the trap occurred. Traps do not disable interrupts, as they are not associated with external events.Traps recognized by the CPU are:

Trap (FPU): An exceptional condition was detected by the TI32081 Floating Point Unit (FPU) or another coprocessor during the execution of a coprocessor instruction. This trap is requested via the Status Word returned as part of the coprocessor protocol (section 3.2.3.9.1).
Trap (ILL): Illegal operation. A privileged operation was attempted while the CPU was in User Mode (PSR bit U = 1).
Trap (SVC): The Supervisor Call (SVC) instruction was executed.
Trap (DVZ): An attempt was made to divide an integer by zero. (The FPU trap is used for floating-point division by zero.)
Trap (FLG): The FLAG instruction detected a "1" in the CPU PSR F bit.
Trap (BPT): The Breakpoint (BPT) instruction was executed.
Trap (TRC): The instruction just completed is being traced.
Trap (UND): An undefined opcode was encountered by the CPU.

A special case is the Trace Trap (TRC), which is enabled by setting the T bit in the Processor Status Register (PSR). At the beginning of each instruction, the T bit is copied into the PSR P ("Trace Pending") bit. If the P bit is set at the end of an instruction, then the trace Trap is activated. If any other trap or interrupt request is made during a traced instruction, its entire service procedure is allowed to complete before the Trace Trap occurs. Each interrupt and trap sequence handles the P bit for proper tracing, guaranteeing one and only one Trace Trap per instruction, and guaranteeing that the Return Address pushed during a Trace Trap is always the address of the next instruction to be traced.

3.2.3.8.6 Prioritization

The TI32016T CPU internally prioritizes simultaneous interrupt and trap requests as follows:

1. Traps other than Trace (Highest priority)
2. Abort
3. Non-Maskable Interrupt
4. Maskable Interrupts
5. Trace Trap (Lowest priority)

3.2.3.8.7 Interrupt/Trap Sequences: Detailed Flow

For purposes of the following detailed discussion of interrupt and trap service sequences, a single sequence called "service" is defined in Table 6. Upon detecting any interrupt request or trap condition, the CPU first performs a sequence dependent upon the type of interrupt or trap. This sequence will include pushing the Processor Status Register and establishing a Vector and a Return Address. The CPU then performs the Service sequence.

Maskable/Nonmaskable Interrupt Sequence. This sequence is performed by the CPU when the $\overline{NMI}$ pin receives a falling edge, or the $\overline{INT}$ pin becomes active with the PSR I bit set. The interrupt sequence begins either at the next instruction boundary or, in the case of String instructions, at the next interruptable point during its execution

1. If a String instruction was interrupted and not yet completed:
   a. Clear the Processor Status Register P Bit.
   b. Set "Return Address" to the address of the first byte of the interrupted instruction. Otherwise, set "Return Address" to the address of the next instruction.
2. Copy the Processor Status Register (PSR) into a temporary register, then clear PSR bits S, U, T, P, and I.
3. If interrupt is Nonmaskable:
   a. Read a byte from FFFF00, applying Status Code 0100 (Interrupt Acknowledge, Master). Discard the byte read.
   b. Set "Vector" to 1.
   c. Go to Step 8.
4. If the interrupt is Nonvectored:
   a. Read a byte from address $FFFF00_{16}$, applying Status Code 0100 (Interrupt Acknowledge, Master (section 3.2.3.4.2). Discard the byte read.
   b. Set "Vector" to 0.
   c. Go to Step 8.
5. Here the interrupt is Vectored. Read "Byte" from address $FFFE00_{16}$, applying Status Code 0100 (Interrupt Acknowledge, Master (section 3.2.3.4.2).
6. If "Byte" greater than or equal to 0, then set "Vector" to "Byte" and go to Step 8.
7. If "Byte" is in the range -16 through -1, then the interrupt source is Cascaded. (More negative values are reserved for future use). Perform the following:
   a. Read the 32-bit Cascade Address from memory. The address is calculated as INTBASE + 4 × Byte.
   b. Read "Vector" applying the Cascade Address just read and Status Code 0101 (Interrupt Acknowledge) (section 3.2.3.4.2).
8. Push the PSR copy (from Step 2) onto the Interrupt Stack as a 16-bit value.
9. Perform Service (Vector, Return Address), Table 6.

Trap Sequence: Traps other than Trace.

1. Restore the currently selected Stack Pointer and the Processor Status Register to their original values at the start of the trapped instruction.
2. Set "Vector" to the value corresponding to the trap type:
   FPU: Vector = 3
   ILL: Vector = 4
   SVC: Vector = 5
   DVZ: Vector = 6
   FLG: Vector = 7
   BPT: Vector = 8
   UND: Vector = 10

3. Copy the Processor Status Register (PSR) into a temporary register, then clear PSR bits S, U, P, and T.
4. Push the PSR copy onto the Interrupt Stack as a 16-bit value.
5. Set "Return Address" to the address of the first byte of the trapped instruction.
6. Perform Service (Vector, Return Address), Table 6.

Trace Trap Sequence

1. In the Processor Status Register (PSR), clear the P bit.
2. Copy the PSR into a temporary register, then clear PSR bits S, U, and T.
3. Push the PSR copy onto the Interrupt Stack as a 16-bit value.
4. Set "Vector" to 9.
5. Set "Return Address" to the address of the next instruction.
6. Perform Service (Vector, Return Address), Table 6.

Abort Sequence

1. Restore the currently selected Stack Pointer to its original contents at the beginning of the aborted instruction.
2. Clear the PSR P bit.
3. Copy the PSR into a temporary register, then clear PSR bits S, U, T, and I.
4. Push the PSR copy onto the Interrupt Stack as a 16-bit value.
5. Set "Vector" to 2.
6. Set "Return Address" to the address of the first byte of the aborted instruction.
7. Perform Service (Vector, Return Address), Table 6.

Table 6. Service Sequence
Invoked during all interrupt/trap sequences

Service (Vector, Return Address):

1) Read the 32-bit External Procedure Descriptor from the Interrupt Dispatch Table: address is Vector*4 + INTBASE Register contents.
2) Move the Module field of the Descriptor into the MOD Register.
3) Read the new Static Base pointer from the memory address contained in MOD, placing it into the SB Register.
4) Read the Program Base pointer from memory address MOD + 8, and add to it the Offset field from the Descriptor, placing the result in the Program Counter.
5) Flush Queue: Non-sequentially fetch first instructin of Interrupt Routine.
6) Push MOD Register onto the Interrupt Stack as a 16-bit value. (The PSR has already been pushed as a 16-bit value.)
7) Push the Return Address onto the Interrupt Stack as a 32-bit quantity.

3.2.3.9 Coprocessor Instructions

The TI32016T CPU recognizes three groups of instructions as being executable by external coprocessors:

Floating-Point Instruction Set
Memory Management Instruction Set
Application-Specific Coprocessor Instruction Set

Each Coprocessor Instruction Set is validated by a bit in the Configuration Register (section 3.2.1.3). Any coprocessor instruction which does not have its corresponding Configuration Register bit will trap as undefined, without any coprocessor communication attempted by the CPU. This allows software simulation of a nonexisting coprocessor.

3.2.3.9.1 Coprocessor Protocol

Coprocessor instructions have 3-byte Basic Instruction field, consisting of an ID Byte followed by an Operation Word. The ID Byte has three functions:

1. It identifies the instruction as being a coprocessor instruction.
2. It specifies which coprocessor will execute it.
3. It determines the format of the following Operation Word of the instruction.

Upon receiving a coprocessor instruction, the CPU initiates the sequence outlined in Table 7. While applying Status Code 1111 (Broadcast ID) (section 3.2.3.4.2) the CPU transfers the ID Byte on the least significant byte of the Data Bus (AD0-D7). All coprocessors input this byte and decode it. The coprocessor selected by the ID Byte is activated, and from this point the CPU is communicating only with it. If any other coprocessor protocol was in progress (e.g., an aborted coprocessor instruction), this transfer cancels it.

Table 7. Coprocessor Protocol

STEP	STATUS	Status Combinations: Send ID (ID): Code 1111 Xfer Operand (OP): Code 1101 Read Status (ST): Code 1110 ACTION
1	ID	CPU Send ID Byte.
2	OP	CPU Sends Operation Word.
3	OP	CPU Sends Required Operands.
4	–	Coprocessor Starts Execution. CPU Pre-Fetches.
5	–	Coprocessor Pulses $\overline{SPC}$ low.
6	ST	CPU Reads Status Word. (Trap? Alter Flags?)
7	OP	CPU Reads Results (If Any).

The CPU next sends the Operation Word while applying Status Code 1101 (Transfer Coprocessor Operand) (section 3.2.3.4.2). Upon receiving it, the coprocessor decodes it, and at this point the CPU and the coprocessor are aware of the number of operands to be transferred and their sizes. The Operation Word is swapped on the Data Bus; that is, bits 0-7 appear on pins AD8-AD15 and bits 8-15 appear on pins AD0-AD7.

Using the Addressing Mode fields within the Operation Word, the CPU starts fetching operands and issuing them to the coprocessor. To do so, it references any Addressing Mode extensions which may be appended to the coprocessor instruction. Since the CPU is solely responsible for memory accesses, these extensions are not sent to the coprocessor.The Status Code applied is 1101 (Transfer Coprocessor Operand) (section 3.2.3.4.2).

After the CPU has issued the last operand, the coprocessor starts the actual execution of the instruction. Upon completion, it will signal the CPU by pulsing $\overline{SPC}$ low. To allow for this and for the address strap translation function, $\overline{AT}/\overline{SPC}$ is normally held high only by a pull-up device of approximately 5 kΩ inside the CPU.

While the coprocessor is executing the instruction, the CPU is free to prefetch instructions into its queue. If it fills the queue before the coprocessor finishes, the CPU will wait, applying Status Code 0011 (Waiting for Coprocessor) (section 3.2.3.4.2).

Upon receiving the pulse on $\overline{SPC}$, the CPU uses $\overline{SPC}$ to read a Status Word from the coprocessor, applying Status Code 1110 (Read Coprocessor Status) (section 3.2.3.4.2). This word has the format shown in Figure 38. If the Q bit ("Quit", Bit 0) is set, this indicates that an error has been detected by the coprocessor. The CPU will not continue the protocol, but will immediately trap through the FPU vector in the Interrupt Table. If the instruction being performed is CMPf and the Q bit is not set, the CPU loads Processor Status Register (PSR) bits N, Z, and L from the corresponding bits in the Status Word. The TI32081 FPU always sets the L bit to zero.

The last step in the protocol is for the CPU to read a result, if any, and transfer it to the destination. The Read cycles from the coprocessor Processor are performed by the CPU while applying Status Code 1101 (Transfer Coprocessor Operand) (section 3.2.3.4.2).

An exception to the protocol above is the LMR (Load Memory Management Register) instruction, and a corresponding Application-Specific Coprocessor instruction (LCR: Load ACU Register). In executing these instructions, the protocol ends after the CPU has issued the last operand. The CPU does not wait for an acknowledgment from the coprocessor, and it does not read status.

3.2.3.9.2 Floating Point Instructions

Table 8 gives the protocols followed for each Floating Point instruction. The instructions are referenced by their mnemonics. For the bit encodings of each instruction, see the Appendix.

The Operand Class Columns give the Access Class for each general operand, defining how the addressing modes are interpreted (see *TI32000 Programmer's Reference Manual*).

The Operand Issued Columns show the sizes of the operands issued to the Floating Point Unit by the CPU:

- "D" indicates a 32-bit double word.
- "i" indicates that the instruction specifies an integer size for the operand (B = Byte, W = Word, D = Double Word).
- "f" indicates that the instruction specifies a floating-point size for the operand (F = 32-bit Standard Floating, L = 64-bit Long Floating).

The Returned Value Type and Destination column gives the size of any returned value and where the CPU places it. The PSR Bits Affected column indicates which PSR bits, if any, are updated from the Coprocessor Status Word (Figure 38).

Table 8. Floating-Point Instruction Protocols

MNEMONIC	OPERAND 1 CLASS	OPERAND 2 CLASS	OPERAND 1 ISSUED	OPERAND 2 ISSUED	RETURNED VALUE TYPE AND DEST.	PSR BITS AFFECTED
ADDf	read.f	rmw.f	f	f	f to Op. 2	none
SUBf	read.f	rmw.f	f	f	f to Op. 2	none
MULf	read.f	rmw.f	f	f	f to Op. 2	none
DIVf	read.f	rmw.f	f	f	f to Op. 2	none
MOVf	read.f	write.f	f	N/A	f to Op. 2	none
ABSf	read.f	write.f	f	N/A	f to Op. 2	none
NEGf	read.f	write.f	f	N/A	f to Op. 2	none
CMPf	read.f	read.f	f	f	N/A	N,Z,L
FLOORfi	read.f	write.i	f	N/A	i to Op. 2	none
TRUNCfi	read.f	write.i	f	N/A	i to Op. 2	none
ROUNDfi	read.f	write.i	f	N/A	i to Op. 2	none
MOVFL	read.F	write.L	F	N/A	L to Op. 2	none
MOVLF	read.L	write.F	L	N/A	F to Op. 2	none
MOVif	read.i	write.f	i	N/A	f to Op. 2	none
LFSR	read.D	N/A	D	N/A	N/A	none
SFSR	N/A	write.D	N/A	N/A	D to Op. 2	none

Note:
D = Double Word
i = Integer size (B, W, D) specified in mnemonic.
N/A = Not Applicable to this instruction.

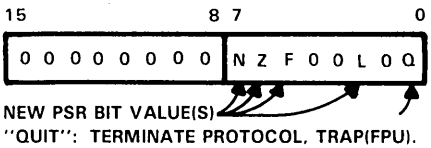


Figure 38. Coprocessor Status Word Format

Any operand indicated as being of type "f" will not cause a transfer if the Register Addressing Mode is specified. This is because the Floating-Point Registers are physically on the Floating Point Unit and are therefore available without CPU assistance.

3.2.3.9.3 Memory Management Instructions

Table 9 gives the protocols for memory management instructions. Encodings for these instructions may be found in the Appendix.

In executing the RDVAL and WRVAL instructions, the CPU calculates and issues the 32-bit Effective Address of the single operand. The CPU then performs a single-byte read cycle from that address, allowing the MMU to safely abort the instruction if the

Table 9. Memory Management Instruction Protocols

MNEMONIC	OPERAND 1 CLASS	OPERAND 2 CLASS	OPERAND 1 ISSUED	OPERAND 2 ISSUED	RETURNED VALUE TYPE AND DEST.	PSR BITS AFFECTED
RDVAL†	addr	N/A	D	N/A	N/A	F
WRVAL†	add	N/A	D	N/A	N/A	F
LMR†	read.D	N/A	D	N/A	N/A	none
SMR†	write.D	N/A	N/A	N/A	D to Op. 1	none

Note:
In the RDVAL and WRVAL instructions, the CPU issues the address as a Double Word, and performs a single-byte Read cycle from that memory address. For details, see the TI32000 Programmer's Reference Manual and the TI32081W Memory Management Unit Data Sheet.
D = Double Word
† = Privileged Instruction: will trap if CPU is in User Mode.
N/A = Not Applicable to this instruction.

necessary information is not currently in physical memory. Upon seeing the memory cycle complete, the MMU continues the protocol, and returns the validation result in the F bit of the Coprocessor Status Word.

The size of the Memory Management operand is always a 32-bit double word. For further details of the Memory Management Instruction set, see the *TI32000 Programmer's Reference Manual* and the Appendix.

3.2.3.9.4 Application Specific Coprocessor Instructions

Provided in the TI32016T is the capability of communicating with a user-defined, "Application-Specific" Coprocessor. The instruction set provided for an Application-Specific Coprocessor defines the instruction formats, the operand classes, and the communication protocol. Left to the user are the interpretations of the op code fields, the programming model of the Application-Specific Coprocessor, and the actual types of data transferred. The protocol specifies only the size of an operand, not its data type.

Table 10 lists the relevant information for the Application-Specific Coprocessor instruction set. The designation "c" is used to represent an operand which can be a 32-bit ("D") or 64'bit ("Q") quantity in any format: the size is determined by the suffix on the mnemonic. Similarly, an "i" indicates an integer size (Byte, Word, Double Word) selected by the corresponding mnemonic suffix.

Any operand indicated as being of type "c" will not cause a transfer if the register addressing mode is specified. It is assumed in this case that the coprocessor is already holding the operand internally.

For the instruction encodings, see the Appendix.

Table 10. Application-Specific Coprocessor Protocols

MNEMONIC	OPERAND 1 CLASS	OPERAND 2 CLASS	OPERAND 1 ISSUED	OPERAND 2 ISSUED	RETURNED VALUE TYPE AND DEST.	PSR BITS AFFECTED
CCAL0c	read.c	rmw.c	c	c	c to Op. 2	none
CCAL1c	read.c	rmw.c	c	c	c to Op. 2	none
CCAL2c	read.c	rmw.c	c	c	c to Op. 2	none
CCAL3c	read.c	rmw.c	c	c	c to Op. 2	none
CMOV0c	read.c	write.c	c	N/A	c to Op. 2	none
CMOV1c	read.c	write.c	c	N/A	c to Op. 2	none
CMOV2c	read.c	write.c	c	N/A	c to Op. 2	none
CCMPc	read.c	read.c	c	c	N/A	N,Z,L
CCV0ci	read.c	write.i	c	N/A	i to Op. 2	none
CCV1ci	read.c	write.i	c	N/A	i to Op. 2	none
CCV2ci	read.c	write.i	c	N/A	i to Op. 2	none
CCV3ci	read.i	write.c	i	N/A	c to Op. 2	none
CCV4DQ	read.D	write.Q	D	N/A	Q to Op. 2	none
CCV5QD	read.Q	write.D	Q	N/A	D to Op. 2	none
LCSR	read.D	N/A	D	N/A	N/A	none
SCSR	N/A	write.D	N/A	N/A	D to Op. 2	none
CATST0†	addr	N/A	D	N/A	N/A	F
CATST1†	addr	N/A	D	N/A	N/A	F
LCR†	read.D	N/A	D	N/A	N/A	none
SCR†	write.D	N/A	N/A	N/A	D to Op. 1	none

NOTE:
D = Double Word
i = Integer size (B, W, D) specified in mnemonic.
c = ACU size (D:32 bits or Q:64 bits) specified in mnemonic.
† = Privileged instruction: will trap if CPU is in User Mode.
N/A = Not Applicable to this instruction.

3.3 TI32081W Floating Point Unit (FPU)

3.3.1 Operand Formats

The TI32081W FPU operates on two floating-point data types: single precision (32 bits) and double precision (64 bits). Floating-point instruction mnemonics use the suffix F (floating) to select the single-precision data type, and the suffix L (long floating) to select the double-precision data type.

A floating-point number is divided into three fields, as shown in Figure 1.

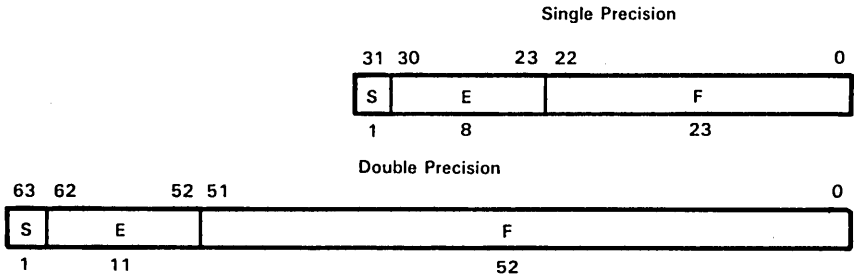


Figure 1. Floating-Point Operand Formats

The F field is the fractional portion of the represented number. In normalized numbers, the binary point is assumed to be immediately to the left of the most significant bit of the F field, with an implied 1 bit to the left of the binary point. Thus, the F field represents values from 1.0 (inclusive) to 2.0 (exclusive) as shown in Table 1.

Table 1. Sample F Fields

F FIELD	BINARY VALUE	DECIMAL VALUE
000...0	1.000...0	1.000...0
010...0	1.010...0	1.250...0
100...0	1.100...0	1.500...0
110...0	1.110...0	1.750...0

↑
Implied Bit

The E field is an unsigned number that gives the binary exponent of the represented number. The value in the E field is biased; that is, a constant bias value must be subtracted from the E field value in order to obtain the true exponent. The bias value is $011...11_2$, which is either the value 127 (single precision) or 1023 (double precision). Thus, the true exponent can be either positive or negative, as shown in Table 2.

Two forms of the E field represent special values, and are not available for use as exponents. The value $11...11_2$ represents a value that is a reserved operand. The value $00...00_2$ represents the number zero if the F field is also all zeros, otherwise the represented value is a reserved operand.

The S bit indicates the sign of the operand: 0 for positive and 1 for negative. Floating-point numbers are in sign-magnitude form, such that only the S bit is complemented in order to change the sign of the represented number.

Table 2. Sample E Fields

E FIELD	F FIELD	REPRESENTED VALUE
011...110	100...0	$1.5 \times 2^{-1} = 0.75$
011...111	100...0	$1.5 \times 2^{0} = 1.50$
100...000	100...0	$1.5 \times 2^{1} = 3.00$

3.3.1.1 Normalized Numbers

Normalized numbers are numbers that can be expressed as floating-point operands, as previously described, where the E field is neither all zeros nor all ones.

The value of a normalized number can be derived by the formula:

$$(-1)^S \times 2^{(E\text{-}Bias)} \times 1.F$$

The range of normalized numbers is given in Table 3.

Table 3. Normalized Number Ranges

	SINGLE PRECISION	DOUBLE PRECISION
Most Positive	$2^{127} \times (2 - 2^{-23})$	$2^{1023} \times (2 - 2^{-52})$
	$= 3.40282346 \times 10^{38}$	$= 1.7976931348623157 \times 10^{308}$
Least Positive	2^{-126}	2^{-1022}
	$= 1.17549436 \times 10^{-38}$	$= 2.2250738585072014 \times 10^{-308}$
Least Negative	$-(2^{-126})$	$-(2^{-1022})$
	$= -1.17549436 \times 10^{-38}$	$= -2.2250738585072014 \times 10^{-308}$
Most Negative	$-2^{127} \times (2 - 2^{-23})$	$-2^{1023} \times (2 - 2^{-52})$
	$= -3.40282346 \times 10^{38}$	$= -1.7976931348623157 \times 10^{308}$

Note: The values given are extended one full digit beyond their represented accuracy to help in generating rounding and conversion algorithms.

3.3.1.2 Zero

There are two representations for zeros: positive and negative. Positive zero has all-zero F and E fields, and the S bit is zero. Negative zero also has all-zero F and E fields, but its S bit is one.

3.3.1.3 Reserved Operands

The proposed IEEE Standard for Binary Floating-Point Arithmetic (Task P754) provides for certain exceptional forms of floating-point operands. The FPU treats these forms as reserved operands. The reserved operands are:

Positive and negative infinity
Not-a-Number (NaN) values
Denormalized numbers

Both infinity and NaN values have all ones in their E fields. Denormalized numbers have all zeros in their E fields and nonzero values in their F fields.

The FPU causes an invalid operation trap if it receives a reserved operand, unless the operation is simply a move (without conversion). The FPU does not generate reserved operands as results.

3.3.1.4 Integers

In addition to performing floating-point arithmetic, the FPU performs conversions between integer and floating-point data types. Integers are accepted and generated by the FPU as two's complement values of byte (8 bits), word (16 bits), or double word (32 bits) length.

3.3.1.5 Memory Representations

The FPU does not directly access memory. However, it is cooperatively involved in the execution of a set of two-address instructions with its TI32000 CPU. The CPU determines the representation of operands in memory.

In the TI32000 microprocesor family, operands are stored in memory with the least significant byte at the lowest byte address. The only exception to this rule is the immediate addressing mode, where the operand is held (within the instruction format) with the most significant byte at the lowest address.

3.3.2 Programming Model

The TI32000 architecture includes nine registers that are implemented on the TI32081W FPU (Figure 2).

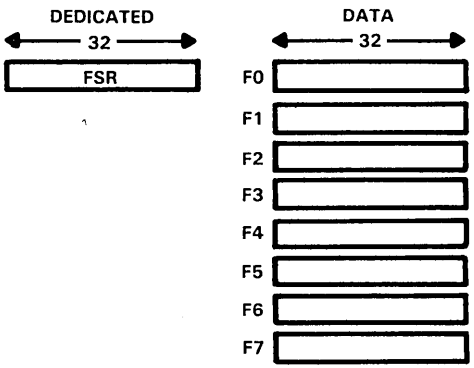


Figure 2. Register Set

3.3.2.1 Floating-Point Registers

There are eight registers (F0-F7) on the FPU for providing high-speed access to floating-point operands. Each is 32 bits long. A floating-point register is referenced whenever a floating-point instruction uses the Register Addressing mode for a floating-point operand. All other register mode usages (i.e., integer operands) refer to the address-data registers (R0-R7) on the CPU. When the Register Addressing mode is specified for a double-precision (64-bit) operand, a pair of registers holds the operand. The programmer must specify the even register of the pair. The even register contains the least significant half of the operand and the next consecutive register contains the most significant half.

3.3.2.2 Floating-Point Status Register (FSR)

The Floating-point Status Register (FSR) selects operating modes and records any exceptinal conditions encountered during execution of a floating-point operation. Figure 3 shows the format of the FSR.

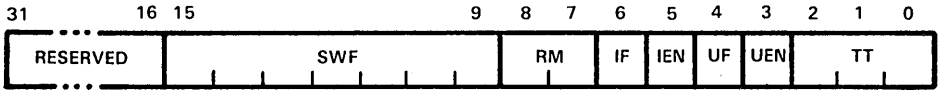


Figure 3. The Floating-Point Status Register

3.3.2.2.1 FSR Mode Control Fields

The FSR mode control fields select FPU operation modes. The definitions of the FSR mode control bits are as follows:

Rounding Mode (RM). Bits 7 and 8. This field selects the rounding method. Floating-point results are rounded whenever they cannot be exactly represented. The rounding modes are:

00 Round to the nearest value. The value that is nearest to the exact result is returned. If the value is exactly halfway between the two nearest values the even value (LSB = 0) is returned.

01 Round towards zero. The nearest value that is closer to zero or equal to the exact result is returned.

10 Round toward positive infinity. The nearest value that is greater than or equal to the exact result is returned.

11 Round toward negative infinity. The nearest value that is less than or equal to the exact result is returned.

Underflow Trap Enable (UEN). Bit 3. If this bit is set, the FPU requests a trap whenever a result is too small in absolute value to be represented as a normalized number. If it is not set, any underflow condition returns a result of exactly zero.

Inexact Result Trap Enable (IEN). Bit 5. If this bit is set, the FPU requests a trap whenever the result of an operation cannot be represented exactly in the operand format of the destination. If it is not set, the result is rounded according to the selected rounding mode.

3.3.2.2.2 FSR Status Fields

The FSR status fields record exceptional conditions encountered during floating-point data processing. The definitions of the FSR status bits are as follows:

Trap Type (TT). Bits 0-2. This 3-bit field records any exceptional condition detected by a floating-point instruction. The TT field is loaded with zero whenever any floating-point instruction except LFSR or SFSR completes without encountering an exceptinal condition. It is also set to zero by a hardware reset or by writing zero into it with the Load FSR (LFSR) instuction. Underflow and Inexact Result are always reported in the TT field, regardless of the settings of the UEN and IEN bits.

000 No exceptional condition occurred.

001 Underflow. A nonzero floating-point result is too small in magnitude to be represented as a normalized floating-point number in the format of the destination operand. This condition is always reported in the TT field and UF bit, but causes a trap only if the UEN bit is set. If the UEN bit is not set, a result of Positive Zero is produced, and no trap occurs.

010 Overflow. A result (either floating-point or integer) of a floating-point instruction is too great in magnitude to be held in the format of the destination operand. Note that rounding, as well as calculations, can cause this condition.

011 Divide by zero. An attempt was made to divide a nonzero floating-point number by zero. Dividing zero by zero is considered an Invalid Operation instead (below).

100 Illegal Instruction. Two undefined floating-point instruction forms are detected by the FPU as being illegal. The binary formats causing this trap are:

xxxxxxxxxx0011xx10111110
xxxxxxxxxx1001xx10111110

101 Invalid Operation. One of the floating-point operands of a floating-point instruction is a Reserved operand, or an attempt has been made to divide zero by zero using the DIVf instruction.

110 Inexact Result. The result (either floating-point or integer) of a floating-point instruction cannot be represented exactly in the format of the destination operand, and a rounding step must alter it to fit. This condition is always reported in the TT field and IF bit unless any other exceptional condition has occurred in the same instruction. In this case, the TT field always contains the code for the other exception and the IF bit is not altered. A trap is caused by this condition only if the IEN bit is set; otherwise the result is rounded and delivered, and no trap occurs.

111 (Reserved for future use.)

Underflow Flag (UF). Bit 4. This bit is set by the FPU whenever a result is too small in absolute value to be represented as a normalized number. Its function is not affected by the state of the UEN bit. The UF bit is cleared only by writing a zero into it with the LFSR instruction or by a hardware reset.

Inexact-Result Flag (IF). Bit 6. This bit is set by the FPU whenever the result of an operation must be rounded to fit within the destination format. This situation applies to both floating-point and integer destinations. The IF bit is set only if no other error has occurred. It is cleared only by writing a zero into it with the LFSR instruction or by a hardware reset.

3.3.2.2.3 FSR Software Field (SWF)

Bits 9-15 of the FSR hold and display any information written to them (using the LFSR and SFSR instruction), but are not otherwise used by the FPU hardware. They are reserved for use with TI32000 floating-point software.

3.3.3 Instruction Set

3.3.3.1 General Instruction Format

Figure 4 shows the general format of a TI32000 instruction. The basic instruction is one to three bytes long and contains the opcode and up to two 5-bit general addressing mode *gen* fields. Following the basic instruction field is a set of optional extensions, which may appear depending on the instruction and the addressing modes selected.

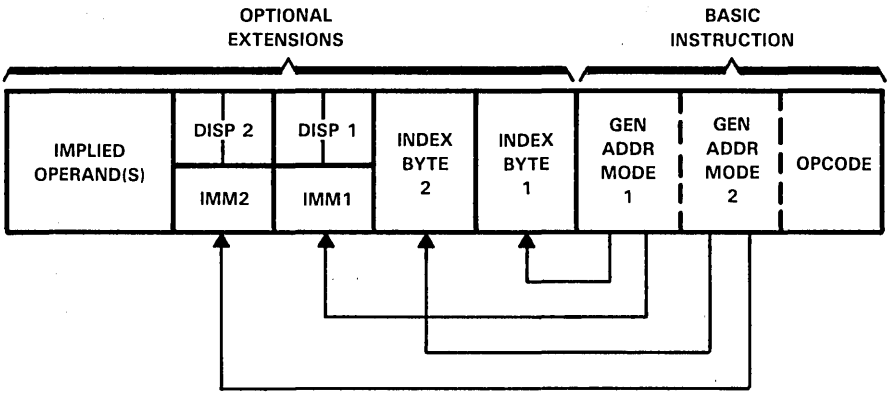


Figure 4. General Instruction Format

The only form of extension issued to the TI32081W FPU is an immediate operand. Other extensions are used only by the CPU to reference memory operands needed by the FPU.

Index bytes appear when either or both *gen* fields specify scaled index. In this case, the *gen* field specifies only the scale factor (1, 2, 4, or 8) and the index byte specifies which address-data register to use as the index, and which addressing mode calculation to perform before indexing. See Figure 5.

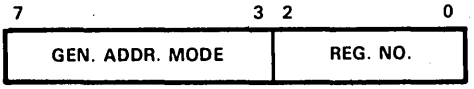


Figure 5. Index Byte Format

Index bytes are followed by any displacements (addressing constants) or immediate values associated with the selected addressing modes. Each Displacement/Immediate (*Disp/Imm*) field may contain one or two displacements, or one immediate value. The size of a *disp* field is encoded within the top bits of that field, as shown in Figure 6, with the remaining bits interpreted as a signed (two's complement) value. The size of an immediate value is determined from the opcode field. Both *Disp* and *Imm* fields are stored most significant byte first.

Some non-FPU instructions require additional, "implied" immediates and/or displacements, apart from those associated with addressing modes. Any such extensions appear at the end of the instruction, in the order that they appear within the list of operands in the instruction definition.

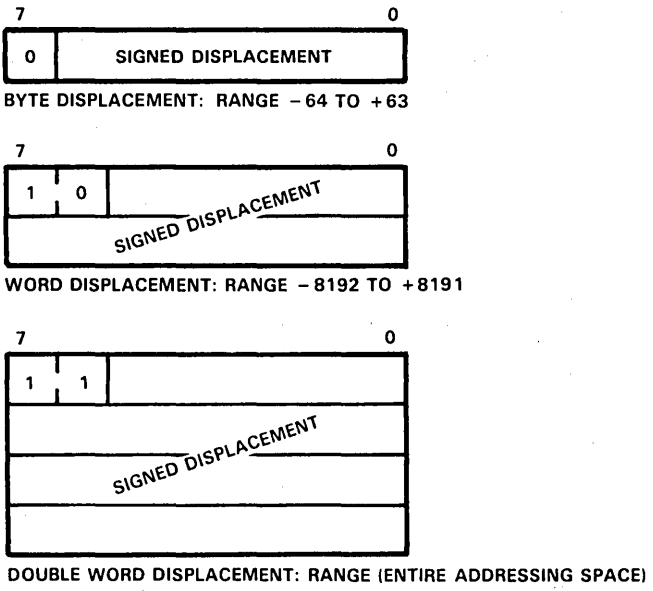


Figure 6. Displacement Encodings

3.3.3.2 Addressing Modes

TI32000 CPUs generally access an operand by calculating its effective address based on information available when the operand is to be accessed. The method to be used in performing this calculation is specified by the programmer as an "addressing mode".

TI32000 addressing modes are designed to optimally support high-level language accesses to variables. In nearly all cases, a variable access requires only one addressing mode within the instruction which acts upon that variable. Extraneous data movement is therefore minimized.

TI32000 Addressing modes fall into nine basic types.

1. Register: In floating-point instructions, these addressing modes refer to a floating-point register (F0-F7) if the operand is of a floating-point type. Otherwise, a CPU address-data register (R0-R7) is referenced.
2. Register Relative: A CPU address-data register contains an address to which is added a displacement value from the instruction, yielding the effective address of the operand in memory.
3. Memory Space: Identical to Register Relative, except that the register used is one of the dedicated CPU registers PC, SP, SB, or FP. These registers point to data areas generally needed by high-level languages.
4. Memory Relative: A pointer variable is found within the memory space pointed to by the CPU's SP, SB, or FP register. A displacement is added to that pointer to generate the effective address of the operand.
5. Immediate: The operand is encoded within the instruction. This addressing mode is not allowed if the operand is to be written. Floating-point operands as well as integer operands may be specified using immediate mode.
6. Absolute: The address of the operand is specified by a *Disp* field in the instruction.
7. External: A pointer value is read from a specified entry of the current link table. To this pointer value is added a displacement, yielding the effective address of the operand.
8. Top-of-Stack: The currently selected CPU stack pointer (SP0 or SP1) specifies the location of the operand. The operand is pushed or popped, depending on whether it is written or read.
9. Scaled Index: Although encoded as an addressing mode, scaled indexing is an option on any addressing mode except immediate or another scaled index. It has the effect of calculating an effective address, then multiplying any address-data register by 1, 2, 4, or 8 and adding it into the total, yielding the final effective address of the operand.

Table 4 provides a brief summary of the addressing modes. For a complete description of their actions refer to the *TI32000 Programmer's Reference Manual*.

3.3.3.3 Floating-Point Instruction Set

The TI32081W FPU instructions occupy formats 9 and 11 of the TI32000 instruction set (Figure 7). A list of all TI32000 instruction formats is found in the Appendix of this data manual.

Certain notations in the following instruction description tables serve to relate the assembly language form of each instruction to its binary format in Figure 7.

The Format column indicates which of the two formats in Figure 7 represents each instruction.

The Op column indicates the binary pattern for the field called "op" in the applicable format.

The Instruction column gives the form of each instruction as it apears in assembly language. The form consists of an instruction mnemonic in upper case, with one or more suffixes (i or f) indicating data types, followed by a list of operands (*gen*1, *gen*2).

Table 4. TI32000 Family Addressing Modes

ENCODING	MODE	ASSEMBLER SYNTAX	EFFECTIVE ADDRESS
Register			
00000	Register 0	R0 or F0	None: Operand is in the specified register.
00001	Register 1	R1 or F1	
00010	Register 2	R2 or F2	
00011	Register 3	R3 or F3	
00100	Register 4	R4 or F4	
00101	Register 5	R5 or F5	
00110	Register 6	R6 or F6	
00111	Register 7	R7 or F7	
Register Relative			
01000	Register 0 relative	disp(R0)	Disp + Register.
01001	Register 1 relative	disp(R1)	
01010	Register 2 relative	disp(R2)	
01011	Register 3 relative	disp(R3)	
01100	Register 4 relative	disp(R4)	
01101	Register 5 relative	disp(R5)	
01110	Register 6 relative	disp(R6)	
01111	Register 7 relative	disp(R7)	
Memory Space			
11000	Frame memory	disp(FP)	Disp + Register; "SP" is either SP0 or SP1, as selected in PSR.
11001	Stack memory	disp(SP)	
11010	Static memory	disp(SB)	
11011	Program memory	* + disp	
Memory Relative			
10000	Frame memory relative	disp2(disp1(FP))	Disp2 + Pointer; Pointer found at address Disp1 + Register. "SP" is either SP0 or SP1, as selected in PSR.
10001	Stack memory relative	disp2(disp1(SP))	
10010	Static memory relative	disp2(disp1(SB))	
Immediate			
10100	Immediate	value	None: Operand is issued from CPU instruction queue.
Absolute			
10101	Absolute	@disp	Disp.

Table 4. TI32000 Family Addressing Modes

ENCODING	MODE	ASSEMBLER SYNTAX	EFFECTIVE ADDRESS
External			
10110	External	EXT (disp1) + disp2	Disp2 + Pointer; Pointer is found at Link Table Entry number Disp1.
Top of Stack			
10111	Top of Stack	TOS	Top of current stack, using either User or Interrupt Stack Pointer, as selected in PSR. Automatic Push/Pop included.
Scaled Index			
11100	Index, bytes	mode[Rn:B]	Mode + Rn.
11101	Index, words	mode[Rn:W]	Mode + 2 × Rn.
11110	Index, double words	mode[Rn:D]	Mode + 4 × Rn.
11111	Index, quad words	mode[Rn:Q]	Mode + 8 × Rn. "Mode" and "n" are contained within the Index Byte.
10011	(Reserved for Future Use)		

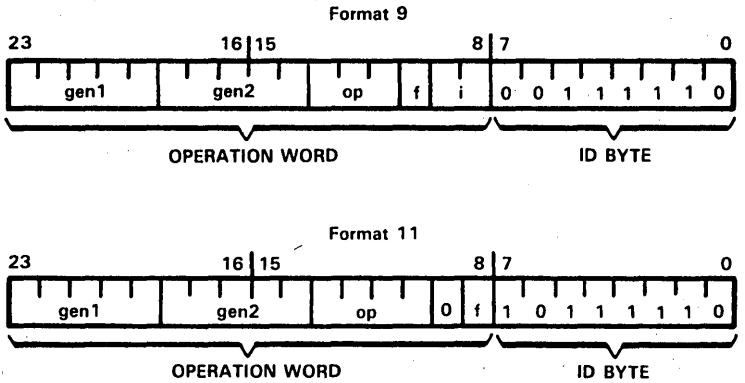


Figure 7. Floating-Point Instruction Formats

An i suffix on an instruction mnemonic indictes a choice of integer data types. This choice affects the binary pattern in the *i* field of the corresponding instruction format as follows:

Suffix i	Data Type	*i* Field
B	Byte	00
W	Word	01
D	Double Word	11

An f suffix on an instruction mnemonic indicates a choice of floating-point data types. This choice affects the setting of the f bit of the corresponding instruction format as follows:

Suffix f	Data Type	f Bit
F	Single Precision	1
L	Double Precision (Long)	0

An operand designation (*gen*1, *gen*2) indicates a choice of addressing mode expressions. This choice affects the binary pattern in the corresponding *gen*1 or *gen*2 field of the instruction format. For the options available and their patterns, refer to Table 4.

Further details of the exact operations performed by each instruction are found in the *TI32000 Programmer's Reference Manual.*

3.3.3.1.1 Movement and Conversion

The following instructions move the *gen*1 operand to the *gen*2 operand, leaving the *gen*1 operand intact:

Format	Op	Instruction		Description
11	0001	MOVf	*gen*1, *gen*2	Move without conversion
9	010	MOVLF†	*gen*1, *gen*2	Move, converting from double precision to single precision.
9	011	MOVFL‡	*gen*1, *gen*2	Move, converting from single precision to double precision.
9	000	MOVif	*gen*1, *gen*2	Move, converting from any integer to any floating-point type.
9	100	ROUNDfi	*gen*1, *gen*2	Move, converting from floating-point to the nearest integer.
9	101	TRUNCfi	*gen*1, *gen*2	Move, converting from floating-point to the nearest integer closer to zero.
9	111	FLOORfi	*gen*1, *gen*2	Move, converting from floating-point to the largest integer less than or equal to its value.

†The MOVLF instruction f bit must be 1 and the *i* field must be 10.
‡The MOVFL instruction f bit must be 0 and the *i* field must be 11.

3.3.3.1.2 Arithmetic Operations

The following instructions perform floating-point arithmetic operations on the *gen*1 and *gen*2 operands, leaving the result in the *gen*2 operand:

Format	Op	Instruction		Description
11	0000	ADDf	*gen*1, *gen*2	Add *gen*1 to *gen*2.
11	0100	SUBf	*gen*1, *gen*2	Subtract *gen*1 from *gen*2.
11	1100	MULf	*gen*1, *gen*2	Multiply *gen*1 by *gen*2.
11	1000	DIVf	*gen*1, *gen*2	Divide *gen*2 by *gen*1.
11	0101	NEGf	*gen*1, *gen*2	Move negative of *gen*1 to *gen*2.
11	1101	ABSf	*gen*1, *gen*2	Move absolute value of *gen*1 to *gen*2.

3.3.3.1.3 Comparison

The compare instruction compares two floating-point values, sending the result to the CPU Processor Status Register (PSR) Z and N bits for use as condition codes. The Z bit is set if the *gen*1 and *gen*2 operands are equal; it is cleared otherwise. The N bit is set if the *gen*1 operand is greater than the *gen*2 operand; it is cleared otherwise. The CPU PSR L bit is unconditionally cleared. Positive and negative zero are considered equal.

Format	Op	Instruction	Description
11	0010	CMPf *gen*1, *gen*2	Compare *gen*1 to *gen*2.

3.3.3.1.4 Floating-Point Status Register Access

The following instructions load and store the FSR as a 32-bit integer.

Format	Op	Instruction	Description
9	001	LFSR *gen*1	Load FSR
9	110	SFSR *gen*2	Store FSR

3.3.4 Traps

Upon detecting an exceptional condition in executing a floating-point instruction, the TI32081W FPU requests a trap by setting the Q bit of the status word transferred during the coprocesor protocol. The CPU responds by performing a trap using a default vector value of 3. Refer to the *TI32000 Programmer's Reference Manual* and the applicable CPU section in this data manual for trap service details.

A trapped floating-point instruction returns no result, and does not affect the CPU PSR. The FPU displays the reason for the trap in the Trap Type (*TT*) field of the FSR.

3.3.5 Functional Operation

3.3.5.1 Power and Grounding

The TI32081W requires a single 5-V power supply on pin 24 (V_{CC}). Ground connections are made on two terminal pins GNDL and GNDB. The GNDL (Logic Ground) terminal pin 12 is the common for on-chip logic. The GNDB (Buffer Ground) terminal pin 13 is the common for the output drivers. For optimum noise immunity, it is recommended that GNDL be attached through a single conductor directly to GNDB, and that all other ground connections also be made only to GNDB, as shown in Figure 8.

3.3.5.2 Clocking

The TI32081W FPU requires a single-phase TTL clock input on its CLK pin (pin 14). When the FPU is connected to a TI32000 CPU, the CLK signal is provided from the CTTL pin of the TI32201 Timing Control Unit.

3.3.5.3 Resetting

The $\overline{RST}$ pin serves as a reset for on-chip logic. The FPU may be reset at any time by pulling the $\overline{RST}$ pin low for at least 64 clock cycles. Upon detecting a reset, the FPU terminates instruction processing, resets its internal logic, and clears the FSR to all zeros.

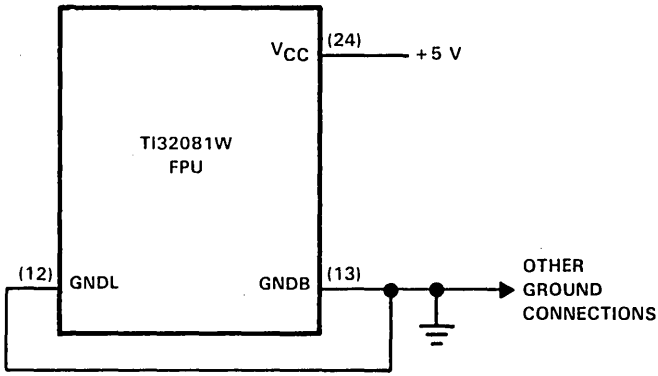


Figure 8. Recommended Supply Connections

On application of power, $\overline{RST}$ must be held low for at least 50 μs after V_{CC} is stable. This insures that all on-chip voltages are completely stable before operation. See Figures 9 and 10.

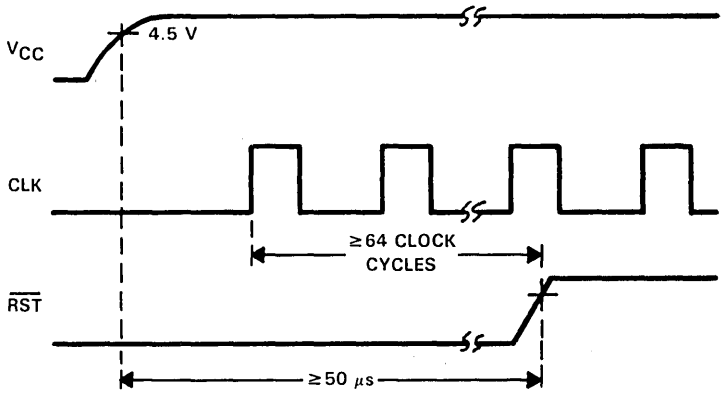


Figure 9. Power-On Reset Requirements

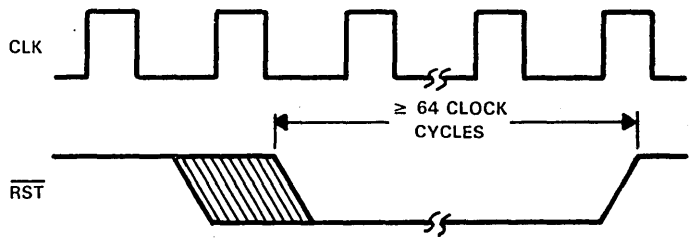


Figure 10. General Reset Timing

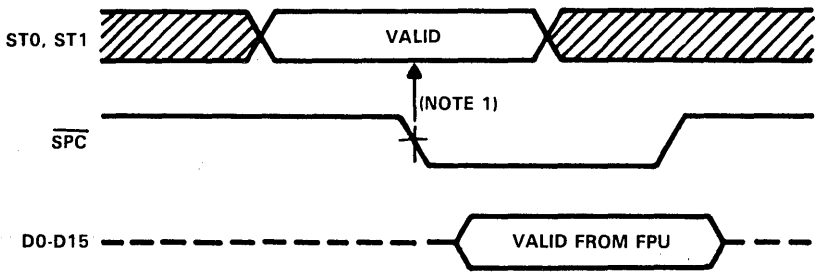
3.3.5.4 Bus Operation

Instructions and operands are passed to the TI32081W FPU with coprocessor bus cycles. Each bus cycle transfers either one byte (8 bits) or one word (16 bits) to or from the FPU. During all bus cycles, the $\overline{SPC}$ line is driven by the CPU as an active-low data strobe, and the FPU monitors pins ST0 and ST1 to keep track of the sequence (protocol) established for the instruction being executed. This is especially necessary in a virtual memory environment, allowing the FPU to retry an aborted instruction.

3.3.5.4.1 Bus Cycles

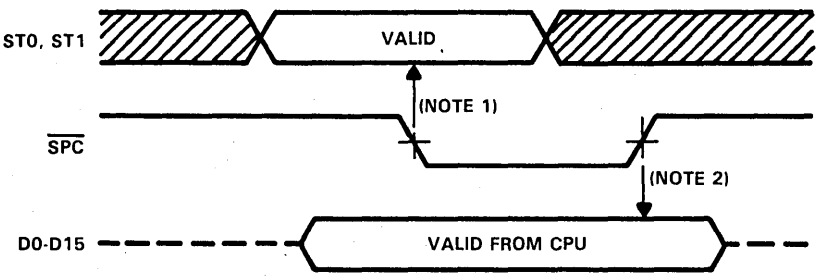
A bus cycle is initiated by the CPU, which asserts the proper status on ST0 and ST1 and pulses $\overline{SPC}$ low. ST0 and ST1 are sampled by the FPU on the leading (falling) edge of the $\overline{SPC}$ pulse. If the transfer is from the FPU (a coprocessor read cycle), the FPU asserts data on the data bus for the duration of the $\overline{SPC}$ pulse. If the transfer is to the FPU (a coprocessor write cycle), the FPU latches data from the data bus on the trailing (rising) edge of the $\overline{SPC}$ pulse. Figures 11 and 12 illustrate these sequences.

The direction of the transfer and role of the bidirectional $\overline{SPC}$ line are determined by the instruction protocol being performed. $\overline{SPC}$ is always driven by the CPU during coprocessor bus cycles. Protocol sequences for each instruction are given in section 3.3.5.5.



NOTE 1: FPU samples CPU status here.

Figure 11. Coprocessor Read Cycle



NOTE 1: FPU samples CPU status here.
NOTE 2: FPU samples data bus here.

Figure 12. Coprocessor Write Cycle

3.3.5.4.2 Operand Transfer Sequences

An operand is transferred in one or more bus cycles. A 1-byte operand is transferred on the least significant byte of the data bus (D0-D7). A 2-byte operand is transferred on the entire bus. A 4-byte or 8-byte operand is transferred in consecutive bus cycles, least significant word first.

3.3.5.5 Instruction Protocols

3.3.5.5.1 General Protocol Sequence

Coprocessor instructions have a three-byte basic instruction field, consisting of an ID byte followed by an operation word. See Figure 7 for FPU instruction encodings. The ID byte has three functions:

1. It identifies the instruction to the CPU as being a coprocessor instruction.
2. It specifies which coprocessor will execute it.
3. It determines the format of the following operation word of the instruction.

Upon receiving a coprocessor instruction, the CPU initiates the sequence outlined in Table 6. While applying status code 11 (broadcast ID, Table 5), the CPU transfers the ID byte on the least significant half of the data bus (D0-D7). All coprocessors input this byte and decode it. The coprocessor selected by the ID byte is activated, and from this point the CPU is communicating only with it. If any other coprocessor protocol is in progress (e.g., an aborted coprocessor instruction), this transfer cancels it.

Table 5. Bus Status Combinations

ST1	ST0	CPU FUNCTION
0	0	(Reserved)
0	1	Transferring Operation Word or Operand
1	0	Reading Status Word
1	1	Broadcasting ID Byte

Table 6. General Instruction Protocol

STEP	STATUS	ACTION
1	11	CPU sends ID Byte.
2	01	CPU sends Operation Word.
3	01	CPU sends required operands.
4	XX	FPU starts execution.
5	XX	FPU pulses $\overline{\text{SPC}}$ low.
6	10	CPU reads Status Word.
7	01	CPU reads result (if any).

The CPU next sends the operation word while applying code 01 (transfer coprocessor operand, Table 5). Upon receiving it, the coprocessor decodes it, and at this point both the CPU and the coprocessor are aware of the number of operands to be transferred and their sizes. The operation word is swapped on the data bus; that is,

bits 0-7 appear on pins D8-D15 respectively, and bits 8-15 appear on pins D0-D7, respectively.

Using the addressing mode fields within the operation word, the CPU starts fetching operands and issuing them to the coprocessor. To do so, it references any addressing mode extensions which may be appended to the coprocessor instruction. Since the CPU is solely responsible for memory accesses, these extensions are not sent to the coprocessor. The status code applied is 01 (transfer coprocessor operand, Table 5).

After the CPU has issued the last operand, the coprocessor starts the actual execution of the instruction. Upon completion, it will signal the CPU by pulsing $\overline{SPC}$ low. To allow for this, the CPU releases the $\overline{SPC}$ signal, causing it to float. $\overline{SPC}$ must be held high by an external pull-up resistor.

Upon receiving the pulse on $\overline{SPC}$, the CPU uses $\overline{SPC}$ to read a status word from the coprocessor, applying status code 10 (read coprocessor status, Table 5). This word has the format shown in Figure 13. If the Q bit (''quit'', bit 0) is set, this indicates that an error has been detected by the coprocessor. The CPU will not continue the protocol, but will immediately trap through the CPU vector in the Interrupt Dispatch Table. If the instruction being performed is CMPf and the Q bit is not set, the CPU loads the PSR bits N, Z, and L from the corresponding bits in the status word. The TI32081W FPU always sets the L bit to zero.

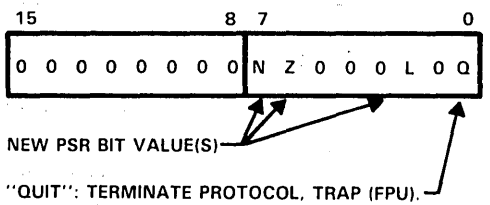


Figure 13. FPU Protocol Status Word Format

The last step in the protocol is for the CPU to read a result, if any, and transfer it to the destination. The read cycles from the coprocessor are performed by the CPU while applying status code 01 (transfer coprocessor operand, Table 5).

3.3.5.5.2 Floating-Point Protocols

Table 7 gives the protocols followed by each floating-point instruction. The instructions are referenced by their mnemonics. For the bit encodings of each instruction, refer to section 3.3.3.3.

The operand class columns give the access classes for each general operand, defining how the addressing modes are interpreted by the CPU. The operand issued columns show the sizes of the operands issued to the FPU by the CPU. A ''D'' indicates a 32-bit double word. An ''i'' indicates that the instruction specifies an integer size for the operand (B = Byte, W = Word, D = Double Word). An ''f'' indicates that the instruction specifies a floating-point size for the operand (F = 32-bit standard floating, L = 64-bit long floating).

The returned value type and destination column gives the size of any value and where the CPU places it. The PSR bits affected column indicates which PSR bits, if any, are updated from the coprocessor status word (Figure 13).

Any operand indicated as being of type "f" will not cause a transfer if the register addressing mode is specified. This is because the floating-point registers are physically on the FPU and are therefore available without CPU assistance.

Table 7. Floating-Point Instruction Protocols

MNEMONIC	OPERAND 1 CLASS	OPERAND 2 CLASS	OPERAND 1 ISSUED	OPERAND 2 ISSUED	RETURNED VALUE TYPE AND DEST.	PSR BITS AFFECTED
ADDf	read.f	rmw.f	f	f	f to Op. 2	none
SUBf	read.f	rmw.f	f	f	f to Op. 2	none
MULf	read.f	rmw.f	f	f	f to Op. 2	none
DIVf	read.f	rmw.f	f	f	f to Op. 2	none
MOVf	read.f	write.f	f	N/A	f to Op. 2	none
ABSf	read.f	write.f	f	N/A	f to Op. 2	none
NEGf	read.f	write.f	f	N/A	f to Op. 2	none
CMPf	read.f	read.f	f	f	N/A	N,Z,L
FLOORfi	read.f	write.i	f	N/A	i to Op. 2	none
TRUNCfi	read.f	write.i	f	N/A	i to Op. 2	none
ROUNDfi	read.f	write.i	f	N/A	i to Op. 2	none
MOVFL	read.F	write.L	F	N/A	L to Op. 2	none
MOVLF	read.L	write.F	L	N/A	F to Op. 2	none
MOVif	read.i	write.f	i	N/A	f to Op. 2	none
LFSR	read.D	N/A	D	N/A	N/A	none
SFSR	N/A	write.D	N/A	N/A	D to Op. 2	none

D = Double Word
i = Integer size (B, W, D) specified in mnemonic.
f = Floating-Point type (F,L) specified in mnemonic.
N/A = Not Applicable to this instruction.

3.4 TI32082W Memory Management Unit (MMU)

For purposes of address translation, memory is divided into 512-byte pages. A logical address for the MMU is composed of two fields: a logical page frame number and a 9-bit offset. The offset is unchanged by the translation algorithm. The MMU translates the logical page number to a physical page number via tables stored in memory. In the established hierarchy, a level 1 table is referred to as a "Page" table, and all level 2 tables are referred to as "Pointer" tables.

The operating system and MMU use the translation tables to exchange information on the status of the memory pages. The table entries track both the presence of a page in the physical memory and the protection level of that page.

By manipulating the page and pointer tables, an operating system dynamically controls the mapping of logical-to-physical addresses. In particular, the operating system may specify that references to certain pages should generate translation error aborts. This mechanism implements virtual memory management and protection.

The logical address output from the TI32032T CPU is 24 bits wide, while the physical address output from the MMU is 25 bits wide. This extra bit (bit 25) can be used to partition memory, but does not increase a task's logical address space.

The MMU has an internal cache memory which contains direct logical-to-physical address mappings of the 32 most recently used pages. Thus, most address translations take only one additional clock cycle. The "hit rate" of the cache memory is usually better than 98%, so that the overhead time involved in dynamic translation is minimal.

The MMU is also capable of breakpoint debugging support. Up to two breakpoint addresses, logical or physical, may be activated in the MMU. A counter may be attached to one of these, enabling "break-on-n occurrences" capability.

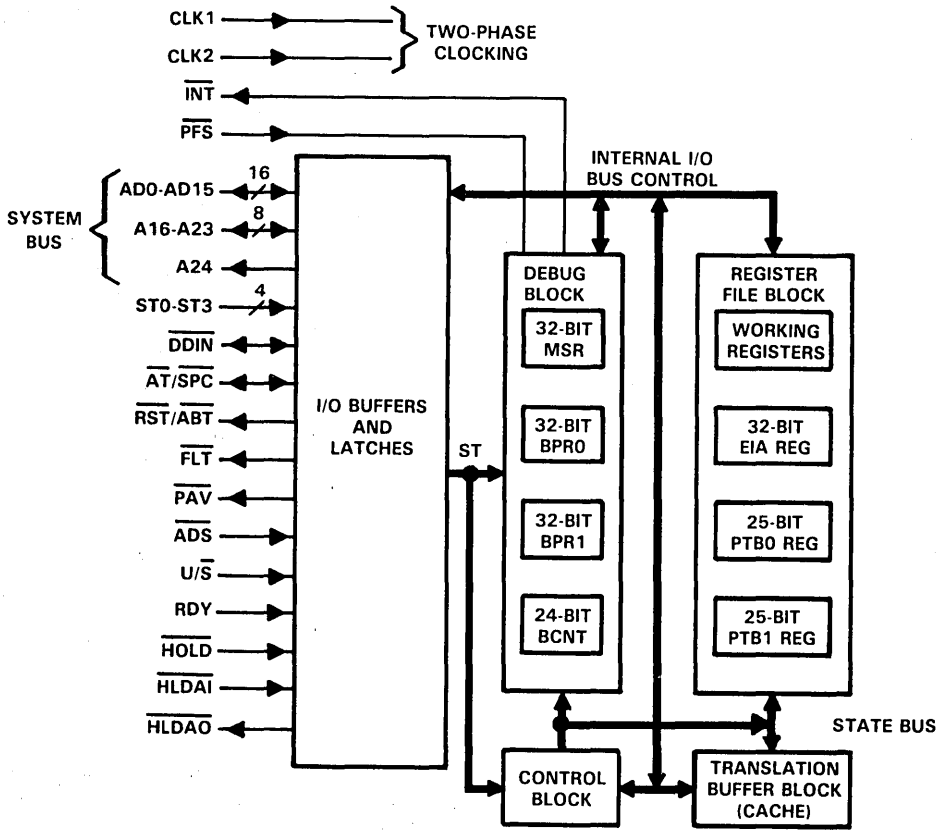
3.4.1 Internal Organization

Internal organization of the TI32082W MMU consists of five functional blocks and their respective addressable registers. These are shown in Figure 1. Both internal and external MMU connections are shown in the block diagram. Detailed block and register operation is described in the following paragraphs.

3.4.1.1 Hardware Debug Block

The debug block contains the registers, counters, and logic which allow the execution of program breakpoints. The debug block includes the following registers:

MMU Status Register (MSR)
Breakpoint Registers (BPR0 and BPR1)
Breakpoint Counter Register (BCNT)



REGISTER DESCRIPTION

MSR = Memory Management Status Register
BPR0 = Breakpoint Register 0
BRP1 = Breakpoint Register 1
BCNT = Breakpoint Count Register
EIA = Error/Invalidate Address Register
PTB0 = Page-Table Base Register 0
PTB1 = Page-Table Base Register 1

Figure 1. MMU Block Diagram

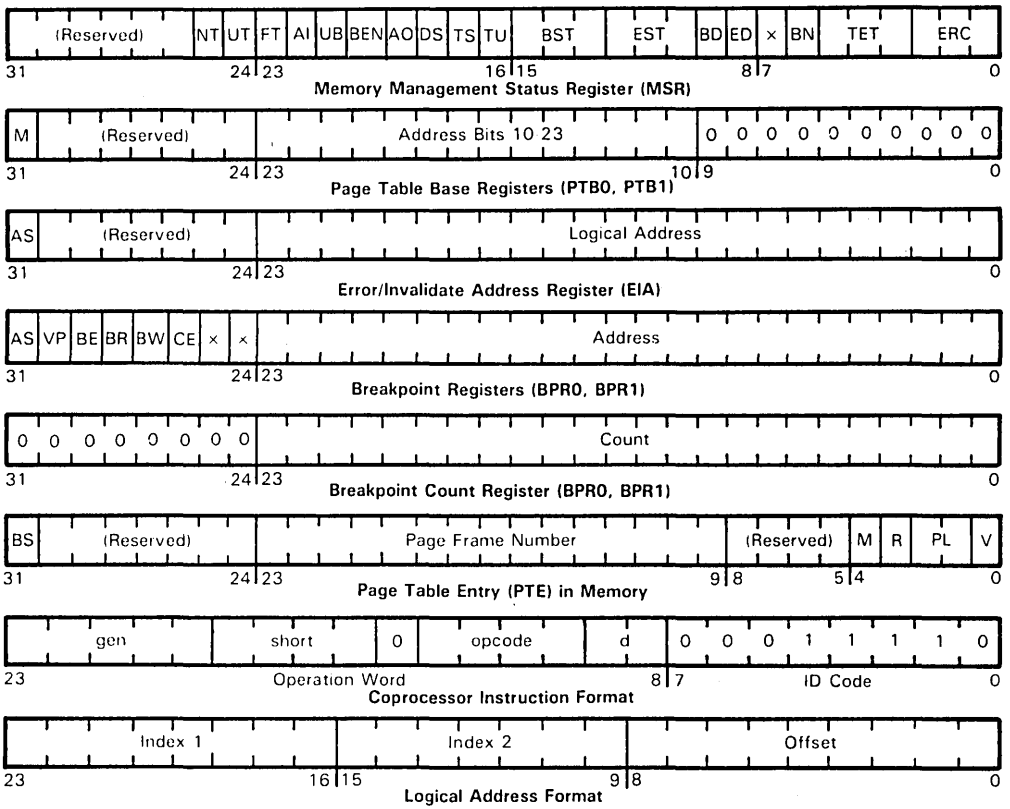
3.4.1.1.1 Memory Management Status Register

The Memory Management Status Register (MSR) specifies the operational mode and current processing status of the MMU. The register permits user control of address translation and breakpoints. The MSR is 32 bits in length. The MSR format is shown in Table 1.

Bits 0 to 25 are the various control bits and flags of the MMU. Bits 26 to 31 are not used. The following describes the control bits and flags:

ERC Error Class flag. This 3-bit flag specifies the cause of the current MMU exception.

Table 1. MMU Bit Maps



Bit 0 is set to 1 on an address translation error.
Bit 1 is not used.
Bit 2 is set to 1 on a break.

TET Translation Error Trace flag. The 3-bit flag specifies the cause of the current address translation error.

Bit 3 is set to 1 on a protection level error.
Bit 4 is set to 1 on an invalid Page Table entry.
Bit 5 is set to 1 on an invalid Pointer Table entry.

BN Breakpoint Number bit. BN is set to indicate the breakpoint address of the current break. If BN is 1, the breakpoint address is contained in BPR1. If BN is 0, the breakpoint address is in BPR0.

ED Error Data Direction bit. If ED is 1, a read operation or the first part of a read-modify-write operation caused an address translation error. If ED is 0, a write or the last part of a read-modify-write operation caused the error.

BD Breakpoint Direction bit. If BD is 1, a read operation or the first part of a read-modify-write operation caused the current break. If BD is 0, a write operation on the last part of a read-modify-write operation caused the break.

EST Error Status flag. On an address translation error, the 3-bit flag is set to the low order three bits of the system status bus.

BST Breakpoint Status flag. On a break, the 3-bit flag is set to the low order three-bits of the system status bus.

TU Translate User bit. If TU is 1, the MMU translates all logical addresses specified in the User mode. If TU is 0, the MMU interprets logical addresses specified in the User mode as physical addresses.

TS Translate Operating System bit. If TS is 1, the MMU translates all addresses specified in the Operating System mode. If TS is 0, the MMU interprets addresses specified in the Operating System mode as physical addresses.

DS Dual Space bit. If DS is 1, the PTB1 register contains the Level 1 Page Table Base address of all addresses specified in the User mode. If DS is 0, the PTB0 register contains the Level 1 Page Table Base address of all addresses specified in both User and Operating System modes.

AO Access Override bit. If AO is 1, the MMU overrides the protection level of all addresses. This permits a program to access memory which is normally accessible only to the operating system while the system is in the User mode. If AO is 0, the MMU does not override the protection level.

BEN Breakpoint Enable bit. If BEN is 1, the MMU enables the BPR0 and BPR1 registers and breaks program execution whenever a breakpoint is encountered. If BEN is 0, the MMU disables the BPR0 and BPR1 registers.

UB User Break bit. If UB is 1, the MMU enables the BPR0 and BPR1 registers for User mode operation only. If UB is 0, the MMU enables the registers for both User and Operating System mode. The UB bit is ignored if breakpoints are disabled (Breakpoint Enable = 0).

AI Abort or Interrupt bit.

FT Flow Trace bit. Not presently used. (Reserved).

UT User Trace bit. Not presently used. (Reserved)

NT Nonsequential Trace bit. Not presently used. (Reserved)

The MSR control bits and flags may be read or modified by executing the SMR and LMR instructions. The NT, FT, TS, TU bits and the ERC flag are set to 0 whenever the system is reset. The BEN bit is set to 0 whenever the MMU generates a breakpoint or an instruction abort on an address translation error.

After writing to the MSR, the MMU automatically suppresses the generation of breakpoints until a branch, jump, call, or return instruction has been executed. This permits a routine to set the MSR and then pass execution to the program being debugged without generating a premature break. The Error Memory Cycle Type (EMCT) is the combination of the BST, EST, BD, and ED fields.

3.4.1.1.2 Breakpoint Registers

The Breakpoint Registers BPR0 and BPR1 provide the breakpoint addresses and breakpoint conditions for system breaks. The registers are each 32 bits in length and have the format shown in Table 1.

Bits 0 to 23 specify the breakpoint address. The MMU compares the breakpoint address with addresses referred to by the program. If a match is found and breakpoint conditions are met, the MMU sends a Nonmaskable Interrupt (NMI) to the system CPU and breaks program execution.

Bits 26 to 31 specify the breakpoint conditions (bits 24 and 25 are not used). Breakpoint conditions define how the MMU compares the breakpoint address and which conditions permit the MMU to generate breaks.

AS Address Space bit. If AS is 0, the MMU compares the breakpoint address with logical addresses whose Level 1 Page Table is specified by the PTB0 register. If AS is 1, the MMU compares the breakpoint address with logical addresses whose Level 1 Page Table is specified by the PTB1 register. If the VP bit is 1, the MMU takes the AS bit as bit 24 of the physical address.

VP Logical/Physical bit. If VP is 0, the MMU compares the breakpoint address with logical addresses only. If VP is 1, the MMU compares the breakpoint address with translated logical addresses (i.e., final physical addresses) or physical addresses only.

BE Breakpoint Execution bit. If BE is 1, the MMU breaks program execution when the instruction at the breakpoint address is executed. The instruction must start at the breakpoint address for the break to occur. If BE is 0, no break occurs.

BR Breakpoint Read bit. If BR is 1, the MMU breaks execution when data is read from the breakpoint address. If BR is 0, no break occurs.

BW Breakpoint Write bit. If BW is 1, the MMU breaks execution when data is written to the breakpoint address or when data is read from the breakpoint address in the first part of a read-modify-write operation. If BW is 0, no break occurs.

CE Counter Enable bit (BPR0 only). If CE is 1, the Breakpoint Count register is enabled. If CE is 0, the register is disabled. The Breakpoint Count register is described in the next section.

3.4.1.1.3 Breakpoint Count Register

The Breakpoint Count (BCNT) register controls the generation of the MMU interrupt signal to the CPU. It permits the user to specify the number of breakpoints the MMU should ignore before generating a break. The BCNT register is 24 bits in length.

The BCNT register affects system breaks only when it is enabled. The CE bit in the BPR0 register enables/disables the register. When the MMU encounters a breakpoint, it checks the CE bit in the register containing the breakpoint address. If CE is 1, the MMU decrements the contents of BCNT by 1, and compares the new contents to zero. If the new contents are not equal to zero, the MMU ignores the breakpoint, i.e., it

permits program execution to continue. If the contents are zero, the MMU breaks execution. If CE is 0, the MMU ignores the BCNT register and breaks program execution.

The user may set the register to any value within the range 0 to $2^{24}-1$ by executing an LMR instruction. If the register is not given a new value after a break, the next breakpoint decrements the register contents by 1.

3.4.1.2 Register File Block

This block contains a number of working registers, with no external access, used to execute the address translation algorithm. In addition, it has three addressable registers (PTB0, PTB1, and EIA) used in performing dynamic address translations.

3.4.1.2.1 Page Table Base Registers

The Page Table Base registers PTB0 and PTB1 specify the base addresses of the Level 1 Page Tables used in address translation. The PTB0 and PTB1 registers are each 32 bits in length and have the format shown in Table 1.

Bits 0 to 23 specify the Page Table Base address. When a logical address is translated, the MMU reads the base address from the register and accesses the specified Page Table. Bits 0 to 9 must be zeros. Bits 24 to 30 are not used. Bit 31 is the Memory Space bit, which may be used to partition physical memory.

The MMU accesses only one Page Table Base register for any given address translation. The current mode of system operation (User or Operating System) and the Dual Space bit (DS) in the MSR specify which register is read. If the DS bit is 0, the MMU reads the base address from the PTB0 register when in either the User or the Operating System mode. If the DS bit is 1, the MMU reads the base address from PTB1 when in User mode and PTB0 when in Operating System mode.

The contents of the registers may be read or modified at any time by executing an SMR and LMR instruction.

3.4.1.2.2 Error/Invalidate Address Register

The Error/Invalidate Address (EIA) register is a dual-purpose register that (1) holds a logical address that has generated an MMU exception; and, (2) when written to, removes Pointer Table entries from the MMU's Translation Buffer. The EIA is 32 bits in length.

The EIA permits examination of the logical address that caused the current MMU exception. On an exception (such as a protection-level error), the MMU copies the logical address that generated the error to the EIA. The MMU sets bit 31 in the EIA to 1 if the address's Level 1 Page Table is specified by PTB1, and to 0 if the Level 1 Page Table is specified by PTB0. The error address may be read by executing an SMR instruction. The cause of the error is specified by the ERC and TET flags in the MSR.

The EIA also permits removal of invalid Pointer Table entries from the MMU's Translation Buffer. The Translation Buffer contains a copy of the Pointer Table entries of recently accessed logical addresses. A logical address written to the EIA causes the MMU to remove the table entry of that logical address from the Translation Buffer. Bit 31 of the EIA must be set to 1 if the Page Table is specified by PTB1 and to 0

if it is specified by PTB0. Entries in the Translation Buffer must be removed whenever the user modifies the corresponding entries in the tables themselves. The user may write to the EIA register using an LMR instruction.

3.4.1.3 Translation Buffer Block

The Translation Buffer is the cache memory of the chip. It provides direct logical-to-physical address mapping for the most recently used pages in memory. Entries in the Translation Buffer are allocated and replaced by the MMU; the programmer is not involved in the process.

The Translation Buffer is a content-addressable memory. The logical page frame number (the 15 high order bits of the logical address) and the address space bit are compared to the entries in the buffer. If the logical page frame number is present in the buffer, the mapped physical address is output immediately. If not, a control line is set, indicating to the Control Block that the memory translation tables should be referenced. When this occurs, the MMU gets the corresponding mapping from memory and replaces the least recently used entry in the Translation Buffer with the new mapping.

Each entry in the Translation Buffer has, besides the logical and physical page frame numbers and the address space bit, a copy of the protection level field (PL) and the modified bit (M) of the corresponding Pointer table entry. These bits are used by the MMU to implement the translation and error handling algorithms described in the Functional Operation section. The protection level field contains the most restrictive combination of the Page and Pointer table entries.

3.4.1.4 Control Block

The Control Block is made up of state machines and combinatorial logic. Each machine controls the sequence of operations taking place during the different MMU operations. A state bus carries the operation code; the different blocks decode appropriate signals from the state bus.

3.4.1.5 Input/Output Block

The Input/Output block consists of I/O buffers and internal buffers.

The I/O buffers provide the communication between the MMU and the outside system bus. The internal buffers between the I/O buses which transfer the address offset and the complete address in no-translation mode are also part of this block.

3.4.2 Memory Management Instructions

Format	Instruction		Description
14	LMR	*mreg,gen*	Load Memory Management Register. (Privileged)
14	SMR	*mreg,gen*	Store Memory Management Register. (Privileged)
14	RDVAL	*gen*	Validate address for reading. (Privileged)
14	WRVAL	*gen*	Validate address for writing. (Privileged)
8	MOVSUi	*gen,gen*	Move a value from operating system Space to User Space. (Privileged)
8	MOVUSi	*gen,gen*	Move a value from User Space to operating system Space. (Privileged)

The MOVSUi and MOVUSi instructions are intended for memory management. Instruction format detail can be found in the Appendix to this data manual.

3.4.3 Functional Operation

3.4.3.1 Power and Grounding

The TI32082W requires a single 5-V power supply applied to pin 48 (V_{CC}). See recommended operating conditions of the TI32082W data sheet.

Grounding connections are made on pins 24 and 25, Logic Ground (GNDL) pin and Buffer Ground (GNDB) pin, respectively. GNDL is the common pin for on-chip logic, and GNDB is the common pin for the output drivers. As shown in Figure 2, GNDL is directly connected to GNDB with a single conductor.

All other grounding connections should be made only to GNDB (pin 25) to ensure optimum noise immunity.

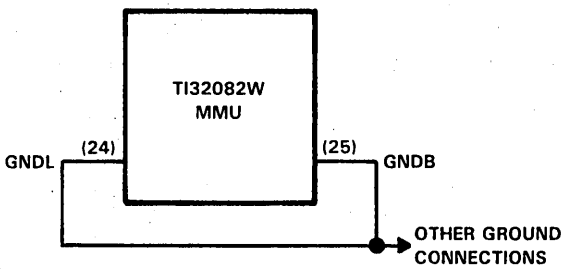


Figure 2. Grounding Connections

3.4.3.2 MMU Operation

The MMU operation incorporates the following:

1. Bus Operation as related to address translation, Direct Memory Access (DMA) transfers, Breakpoints-on-Physical Address, and Coprocessor Operation
2. Coprocessor Instruction Execution
3. Address Translation
4. Hardware Debugging
5. Error Handling

3.4.3.2.1 Bus Operation

Address Translation (see Figures 3 through 6). The MMU time-shares the address/data bus with the CPU. During a memory access cycle, the MMU reads the logical address, performs the logical-to-physical translation, and places the physical address on the bus. A typical memory cycle has five clock periods: T1, TMMU (time of physical address on the bus), T2, T3, and T4. The 16 A/D bus drivers of the MMU are in high impedance state at all times except during TMMU or when the $\overline{FLT}$ signal is active. The bus drivers of lines A16 to A24 drive the bus from TMMU through T4.

During period T1, the CPU places on the bus the logical address to be translated; this address is strobed into the MMU with the $\overline{ADS}$ pulse. During period TMMU, the CPU places the bus in high impedance and the MMU does one of two things. If the address to be translated is in the translation buffer, the MMU sends the physical address on

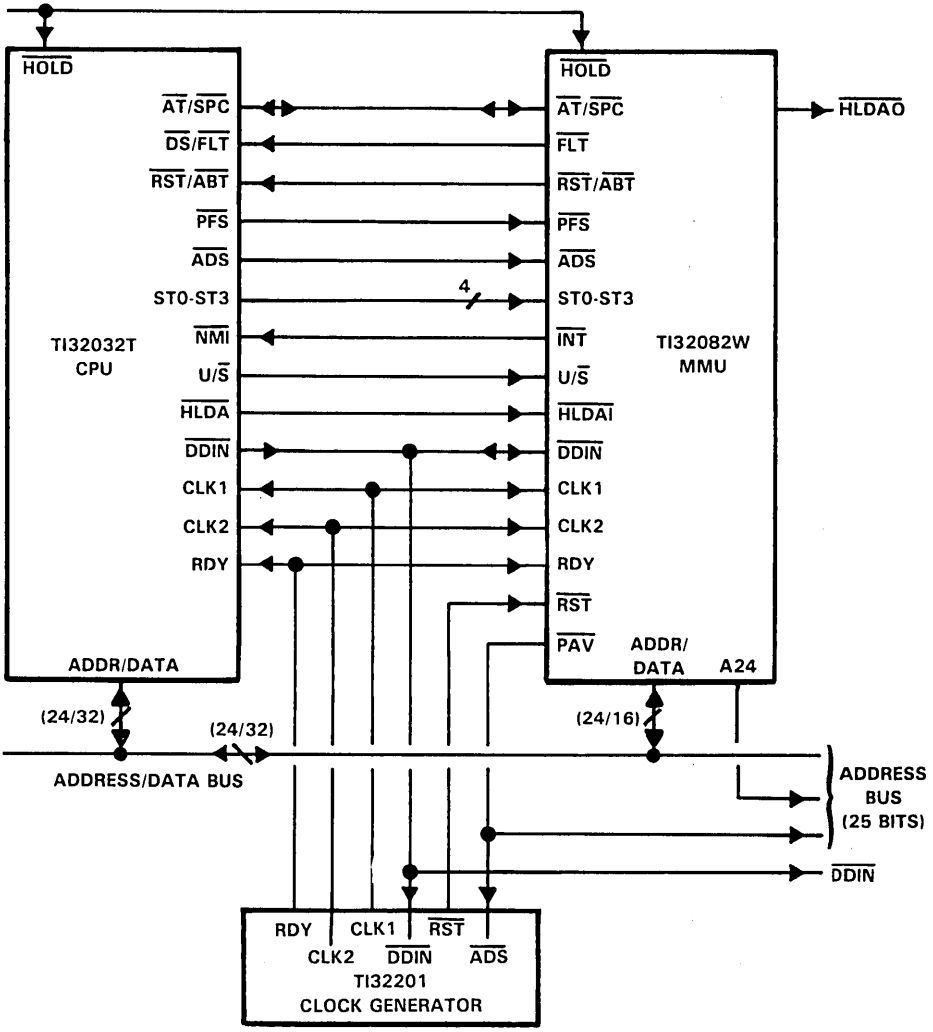


Figure 3. CPU, MMU Interconnections

the bus with a $\overline{PAV}$ timing pulse; if not, it takes the bus from the CPU with the $\overline{FLT}$ signal and executes four memory read cycles, to get the two double words needed to perform the translation algorithm. When necessary, the MMU executes two memory write cycles to update the referenced and modified bits in the Page and Pointer table entries. It then releases control of the bus and sends the physical address on the bus. The memory cycle initiated by the CPU is resumed from the point it was stopped.

Between periods T2 and T4, there is data on the AD0-AD15 bus lines, output either by the CPU or memory. Bus lines A16 to A24 continue to hold the physical address.

DMA Transfers. The Hold ($\overline{HOLD}$) and Hold Acknowledge ($\overline{HLDA}$) lines are connected as shown in Figure 7.

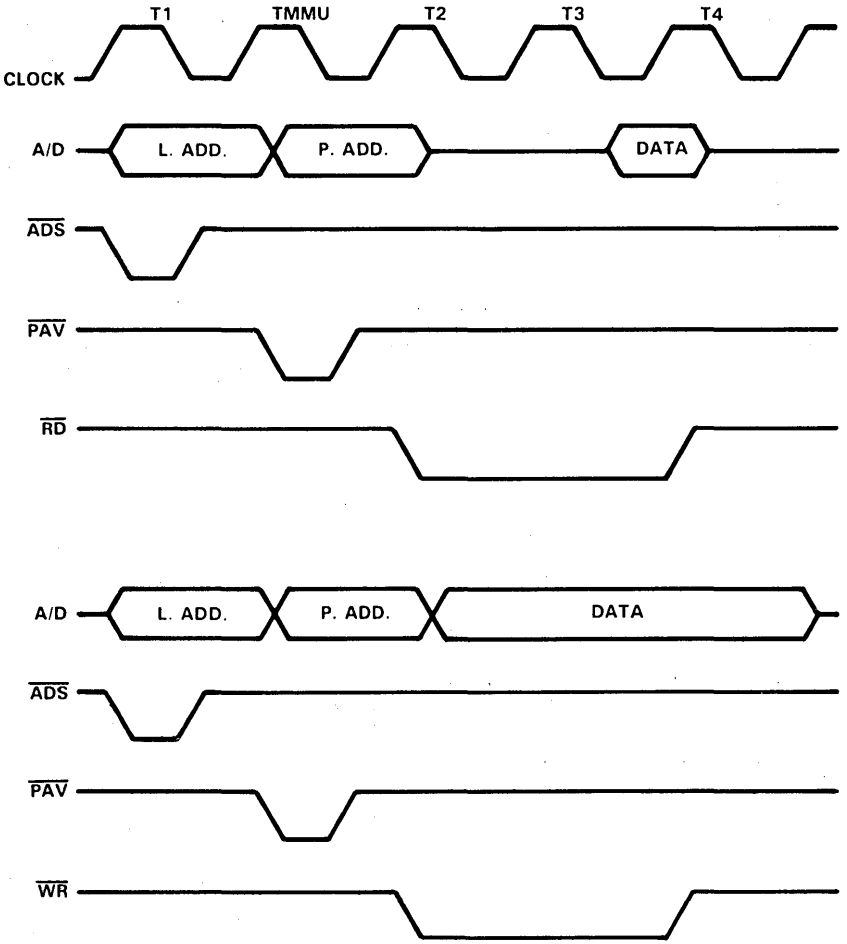


Figure 4. Bus Operation Timing: Logical Address in Translation Buffer

The DMA device pulls the $\overline{\text{HOLD}}$ line to request the bus; this line is input to both the CPU and the MMU. If the MMU is not floating the CPU (through the $\overline{\text{FLT}}$ line), the MMU transfers the CPU $\overline{\text{HLDA}}$ output directly to the MMU $\overline{\text{HLDAO}}$ output. If the MMU (when accessing the address translation tables) is floating the CPU, the CPU cannot respond to a Hold Request, $\overline{\text{HLDAI}}$ remains high, and the MMU grants the bus by pulling low $\overline{\text{HLDAO}}$ at the end of the present memory cycle. When the DMA device releases $\overline{\text{HOLD}}$, the MMU releases $\overline{\text{HLDAO}}$ and regains control of the bus.

Breakpoints-on-Physical Address. During debug, if a breakpoint is specified to occur on a physical address (VP is set in any BPR), an additional clock period is needed in the bus cycle. The additional clock period is required to make the address comparison after getting the physical address from the cache or Pointer Table. In this case, the MMU floats the CPU for one clock period. This gives the memory cycles six periods: T1, TMMU, Tf, T2, T3 and T4. The corresponding waveforms are illustrated in Figure 8.

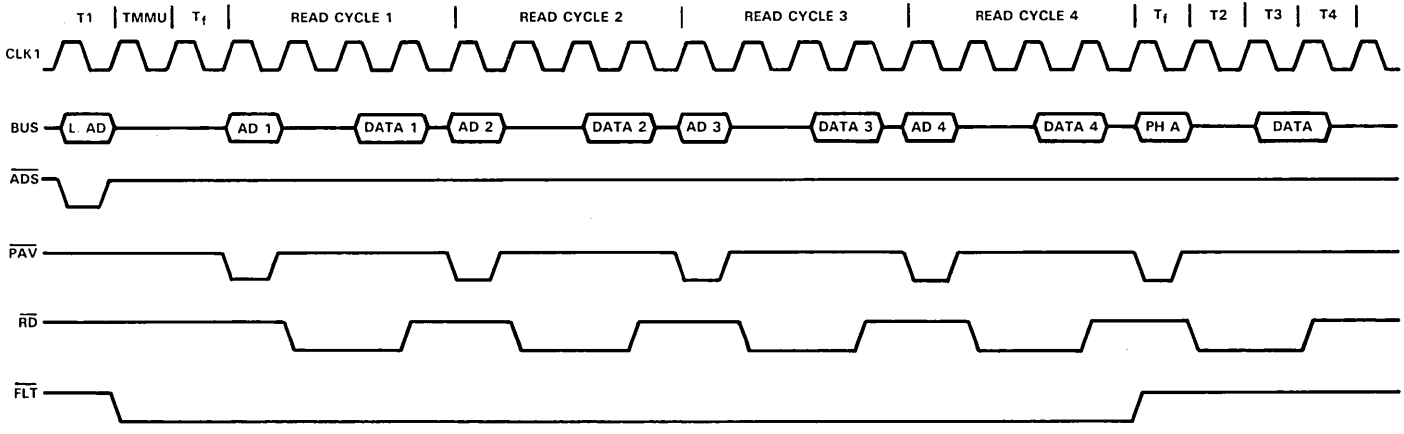


Figure 5. Bus Operation: Read Cycle When Logical Address is not in Translation Buffer

Figure 6. Bus Translation Write Cycle When Logical Address is not in Translation Buffer

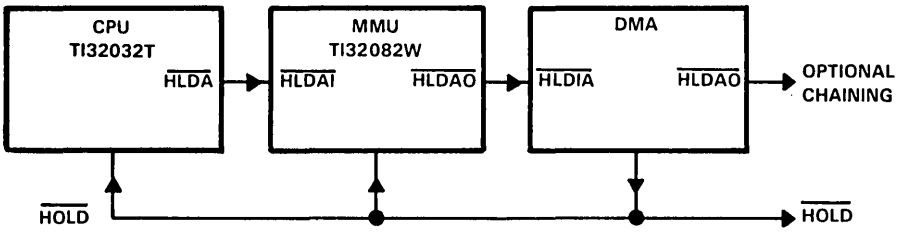


Figure 7. Hold Connections

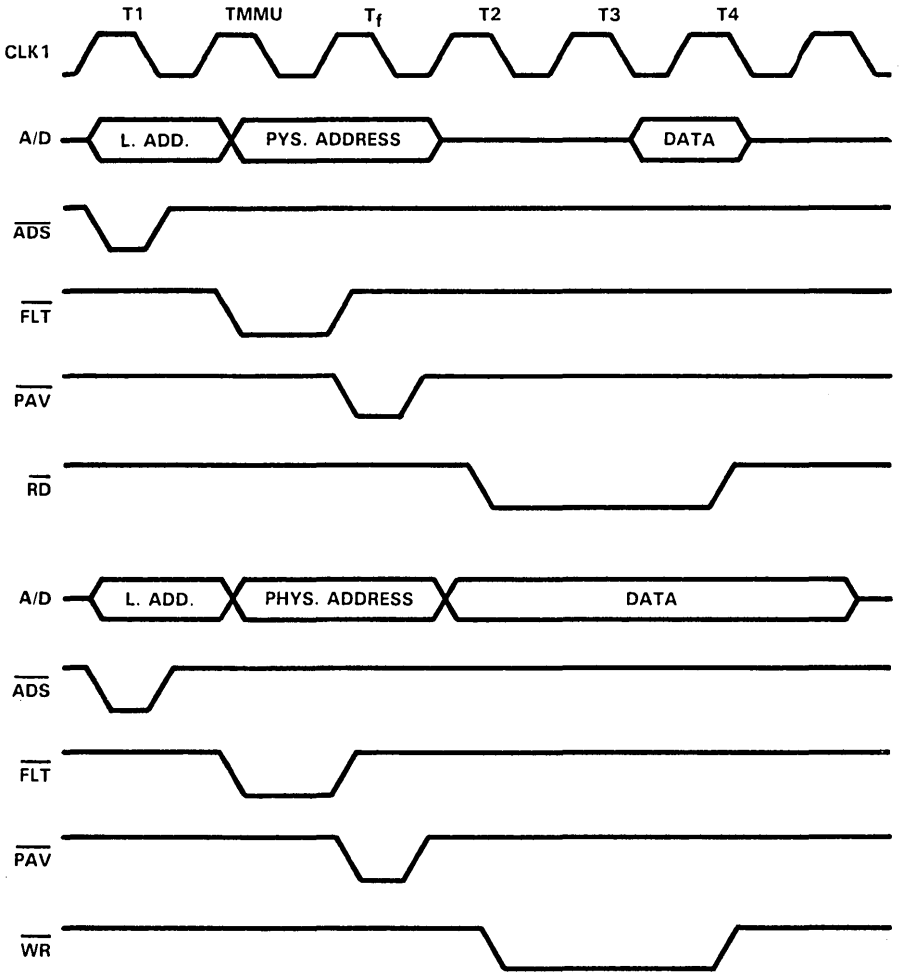


Figure 8. Bus Operation in Breakpoints on Physical Address

Coprocessor Instruction Bus Operation. For coprocessor instructions, the bus operation follows a different protocol. The bus cycle has only two periods (T1 and T4) and the timing is done by a one-clock-wide pulse on the Coprocessor Control ($\overline{SPC}$) bidirectional line. All bus transfers are illustrated in Figures 9 and 10.

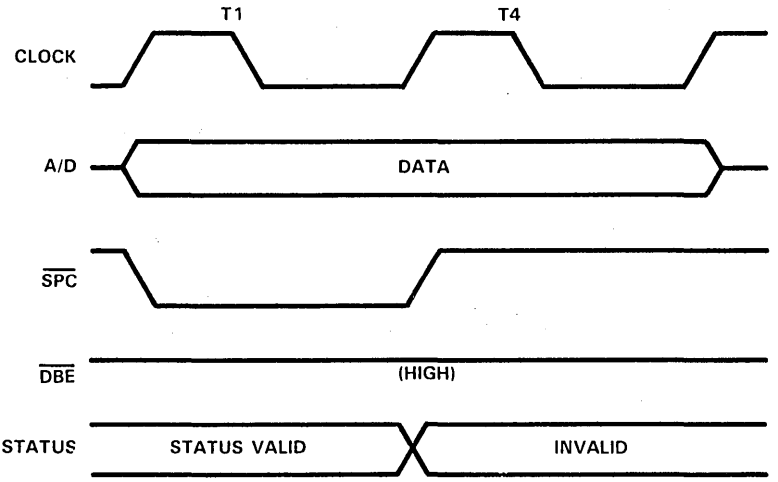


Figure 9. Coprocessor Instruction Timing: Get ID/Opcode/Data from CPU

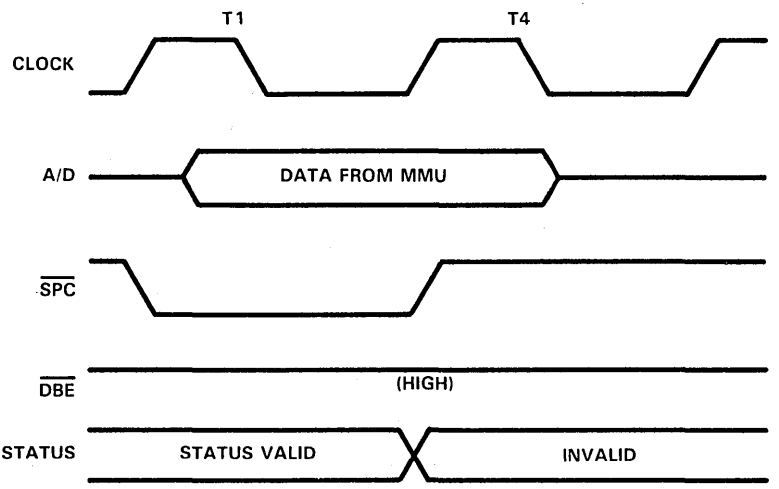


Figure 10. Coprocessor Instruction Timing: MMU Sends Status/Data to CPU

3.4.3.2.2 Coprocessor Instructions

Introduction to Coprocessor Instructions. The MMU coprocessor instructions serve two purposes. (1) Coprocessor instructions set up the different registers and check their contents (LMR and SMR instructions) in order to control the MMU mode of

operation. (2) A coprocessor instruction can request the MMU to return a flag indicating whether a specified access to a given address would generate a protection fault in User mode.

The general format for coprocessor instructions appears in the Appendix to this data manual. The formats for the MMU coprocessor instructions are described in the following paragraphs.

Note:
All MMU instructions are privileged. While in the User Mode, the CPU will trap on any MMU instruction.

MMU Coprocessor Instruction Format. The 3-byte format of the MMU coprocessor instruction is shown in Table 1.

The format corresponds to the instructions as they are stored in memory; the CPU sends the operation word to the MMU with its bytes swapped, i.e., high byte in the low bus byte and vice versa.

The short code assignments for the registers are shown below:

Code Value	Register
0000	BPR0
0001	BPR1
1010	MSR
1011	BCNT
1100	PTB0
1101	PTB1
1111	EIA

Note:
All other short codes are illegal.

Address Translation Validation Instructions. The two instructions used to validate an address are: The RDVAL address and the WRVAL address. Both instructions consists of mnemonics and address type operands. (Table 2)

Upon receipt of a RDVAL or WRVAL instruction, the MMU checks to see if the address operand can be translated without protection violations in User mode (user space). If the address can be translated without violations, the MMU sends status word zero. If not, the MMU sends status word 32.

If the first translation table entry is invalid, a trap is generated with error class 1 and error translation type 2. No trap is generated if the second PTE is invalid or if protection violation errors occur.

A validate instruction generates a status word which sets or resets the flag (F) bit in the CPU PSR register. The remaining bits are all zero.

Table 2. RDVAL/WRVAL Instruction (Validate Read/Write Address)

CPU		MMU	
EXECUTION UNIT	BUS INTERFACE UNIT	STATUS PINS	ACTION
Sends ID Code in low byte	Sends ID Code with $\overline{SPC}$ timing pulse	1111	Recognizes ID Code
Sends Opcode in two bytes	Sends Opcode with $\overline{SPC}$ timing pulse	1101	Latches Opcode
Sends Address to be validated in two words (bits 24-31 set to zero)	Sends Address in two Coprocessor Write cycles with $\overline{SPC}$ timing pulse	1101	
Generates Dummy Read with address to be validated	Starts a Read cycle with address to be validated	1010	Performs validation
	Detects MMU completion	0011	Signals completion $\overline{SPC}$ pulse
Reads MMU status	Reads MMU status word with $\overline{SPC}$ strobe	1110	Sends status word

LMR INSTRUCTION (LOAD MMU REGISTER)

LMR short, read.d (See *TI32000 Programmer's Reference Manual*)

The MMU register specified by first operand is loaded with the contents of the second operand. The instruction executes as follows:

CPU		MMU	
EXECUTION UNIT	BUS INTERFACE UNIT	STATUS PINS	ACTION
Sends ID Code in low byte	Sends ID Code with $\overline{SPC}$ timing pulse	1111	Recognizes ID Code
Sends Opcode in two bytes	Sends Opcode with $\overline{SPC}$ timing pulse	1101	Latches Opcode
Sends low word of operand	Sends low word of operand with $\overline{SPC}$ timing pulse	1101	Stores operand in low word of addressed register
Sends high word of operand	Sends high word of operand with $\overline{SPC}$ timing pulse	1101	Stores operand in high word of addressed register

Table 2. RDVAL/WRVAL Instruction (Validate Read/Write Address) (Concluded)

SMR INSTRUCTION (STORE MMU REGISTER)

SMR short, write.d

The MMU register specified by first operand is stored in the second operand. The instruction executes as follows:

CPU		MMU	
EXECUTION UNIT	**BUS INTERFACE UNIT**	**STATUS PINS**	**ACTION**
Sends ID Code in low byte	Sends ID Code with $\overline{SPC}$ timing pulse	1111	Recognizes ID Code
Sends Opcode in two bytes	Sends Opcode with $\overline{SPC}$ timing pulse	1101 (See Note 1)	Latches Opcode
	Detects MMU completion	0011	Signals completion with $\overline{SPC}$ pulse
	Reads status with $\overline{SPC}$ strobe	1110	Sends zero status
	Strobes operand with the $\overline{SPC}$ pulse	1101	Sends low word of addressed register
	Strobes operand with $\overline{SPC}$ pulse	1101	Sends high word of addressed register

Notes: 1. The CPU may prefetch more code before this step.
2. After CPU reads the operand, the contents are stored in second operand according to the second operand addressing mode.
3. If addressed register is less than 32 bits, then the high order bits are reset to zero.

Page and Pointer Table Entry (PTE) Format. Address translation is controlled by Page and Pointer tables contained in memory. A table is a linear array of 32-bit entries. Each PTE defines the access characteristics of one page (512 bytes) of virtual storage. The PTE bit format is shown in Table 1.

BS Bank select: Most significant bit of PFN field.

PFN Page Frame Number: When the V bit is set, the PFN low field, together with the BS bit, contains the high order 16 bits of a physical page address which is used by the address translation algorithm.

M Modified: Used only in Pointer table entries (bits 9 to 15 of logical address) and set when mapped page is modified.

R Referenced: Set when page mapped by PTE is referenced.

PL Protection Level: Page and Pointer table entries control access to pages mapped by the PTE. The following shows the relationship between user, operating system and protection level bits:

	PSR	Protection Level Bits			
Mode	**Bit 8**	**00**	**01**	**10**	**11**
User	1	no access	no access	read only	full access
Operating System	0	read only	full access	full access	full access

V Valid bit: When set, indicates that the corresponding page is resident in physical memory. When cleared, any attempted reference to the page will cause the MMU to abort the reference. If the V-bit is cleared, the PTE may be used by the operating system for any desired function.

Note:
Bits 7 and 8 are reserved for the user and are not affected by the MMU.

Address Translation Algorithm. The MMU translates the 24-bit logical address generated by the CPU to either a 25-bit physical address or a translation error. This process is described in the following paragraphs. See Figure 11.

The logical address is divided into three components as shown in Table 1. The access level of a reference is a 2-bit number whose logical expressions are:

bit 1 = U AND AO

where

AO = Access Override bit in MSR
bit 0 = 1 for write, Read/Modify/Write (RMW)
bit 0 = 0 for read

The detailed description of the translation algorithm follows. (Also refer to the *TI32000 Programmer's Reference Manual.*)

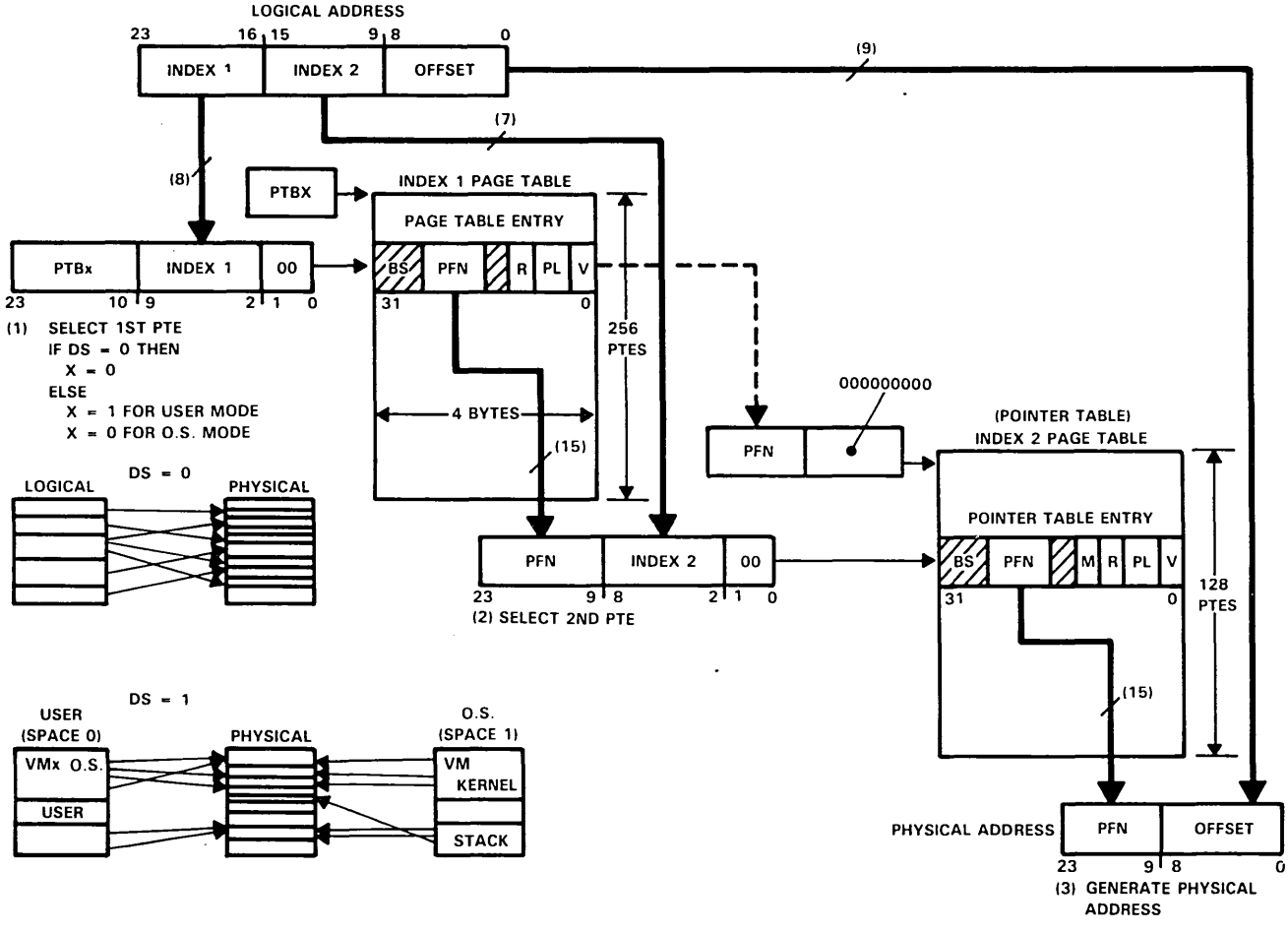


Figure 11. Logical to Physical Address Translation

If TU = 0 and U = 1 or TS = 0 and U = 0, then PA = logical address, else

1. Select first PTE:

   If DS (in MSR) = 1 and U (in PSR) = 1, then PTEP = PTB1 or Index 1 * 4.

   else

   PTEP = PTB0 or Index 1*4

   end.

   Validate PTE:

   If access level is greater than (PTEP).PL or if (PTEP).V = 0, then abort CPU

   else

   Set (PTEP).R = 1

2. Select second PTE:

   PTEP = (PTEP).PFN * 512 or Index 2 * 4

   Validate PTE:

   If access level is greater than (PTEP).PL or if (PTEP).V = 0, then abort CPU

   else

   Set (PTEP).R = 1

   If writing, then set (PTEP).M = 1

3. Generate physical address:

   PA = (PTEP).PFN * 512 or Offset

   Legend:

PA	Physical Address
TU, DS, TS	MSR bits
U	PSR bit (sent to MMU via the U/S pin)
PTEP	PTE pointer
(PTEP).PL	represents protection level in Page or Pointer Table Entry
(PTEP).V	represents valid bit in Page or Pointer Table Entry
(PTEP).M	represents modified bit in Page or Pointer table entry
PFN	Page Frame Number

The MMU marks bits R and M of the PTE for subsequent use by the operating system. If a physical page is written upon, it is assumed that the user intends for this modification to be permanent in his storage system. The M bit indicates whether a page needs to be written to mass storage when it is deallocated from physical memory. The R bit is tested and cleared periodically by the operating system in order to compile statistics of the frequency of references to each page currently in memory. It will use this information to deallocate the least frequently used pages when new pages must be called in.

Pointer tables that refer to physical pages are referenced by page tables, 1K bytes in length. Selection of the PTB0 or PTB1 register depends on the Dual Space (DS) and User/Operating System (U/$\overline{S}$) modes as shown in the following:

		U/$\overline{S}$	
		0	1
DS	0	PTB0	PTB0
	1	PTB0	PTB1

Page Table Base (PTB) Registers. PTB0 and PTB1 registers are specified as double words. The BS bit in the PTE is used by the MMU to produce the 25th bit of the physical addresses pointing to the entries in the Page table. Their format is given in Table 1.

3.4.3.2.4 Hardware Debugging

The TI32082W MMU incorporates a special debugging facility: address breakpointing. A breakpoint generates an abort or interrupt pulse when a software specified address is referenced under software controlled conditions. It also updates the ERC and BN fields in MSR. Breakpoints are controlled by the BEN and UB bits (in MSR) and the BPR registers which have the format shown earlier in Table 1.

Breakpoint-on-Execution Fetch Mechanism. When a sequential instruction is fetched by the CPU, the instruction is placed in the queue. Unless the queue is empty, aborts on queue fetches are not received and a breakpoint could be missed. The proper operation of breakpoint execution requires flushing the queue, as described in the following paragraphs.

When the BE bit is set and the location specified in the BPR is accessed in a nonsequential fetch, an Abort ($\overline{RST}$/$\overline{ABT}$) or $\overline{INT}$ pulse is generated.

When the BE bit is set and the location specified in the breakpoint register is accessed in a sequential fetch (or in a nonsequential fetch from an even-numbered address (2n) and the location specified in BPR is (2n + 1), the MMU returns a DIA instruction instead of the memory byte at the breakpoint location. This is preceded by a read cycle in order to return the other original byte from memory. This causes the CPU to flush the queue and to fetch the instruction a second time, this time with a nonsequential fetch status.

The BPR bit functions are tabulated below:

AS Address Space: Logical address when VP = 0, bank select bit of physical address when VP = 1.

VP Logical or Physical address: If VP is set, the address field is matched against physical address. If VP is reset, the address field is matched against the logical address.

BE Breakpoint-on-Execution: If BE is set, a breakpoint occurs when the location specified in the address field is referenced in an instruction fetch cycle (instruction execution detailed below).

BR Breakpoint-on-Read operand: If BR is set, a breakpoint occurs when the location specified in the address field is referenced in a read operand cycle.

BW Breakpoint-on-Write operands: If BW is set, a breakpoint occurs when the location specified in the address field is referenced in a write or RMW operand cycle.

CE Counter Enable (BPR0 only): The 24-bit BCNT counter decrements when Counter Enable bit (CE) is set and the conditions for a breakpoint in register BPR0 are obtained. When this counter reaches zero, an "Abort" or $\overline{INT}$ pulse is generated by the MMU.

Note:
An erroneous count will result if both the CE and BW bits are set.

3.4.3.2.5 Error Handling

Traps are serviced according to class and type (c, t). In the MSR register, the appropriate bit in the ERC field is set due to the fact that RMW accesses are counted twice.

For Address Translation Error, the following bits are set in the TET field:

If access level is greater than (PTEP).PL	bit 0 set
If (PTEP).V = 0 in Page Table PTE	bit 1 set
If (PTEP).V = 0 in Pointer Table PTE	bit 2 set

In the EMCT field, set the CPU status and $\overline{DDIN}$ bits.

In the EAI register, set AS bit to designate the address space PTB0/PTB1 of logical address being translated and set the address field to the value of the logical address being translated, as shown in the register format in Table 1.

For Breakpoint Error, the following bits are set in the MSR register:

BN field — the number of the appropriate breakpoint register
EMCT field — CPU status and DDIN bits

3.5 TI32202W Interrupt Control Unit (ICU)

3.5.1 General Description

The TI32202W ICU functions as an overall manager in an interrupt-oriented system environment. Its many features and options permit the design of sophisticated interrupt systems.

Figure 1 shows the internal organization of the TI32202W which is divided into five functional blocks.

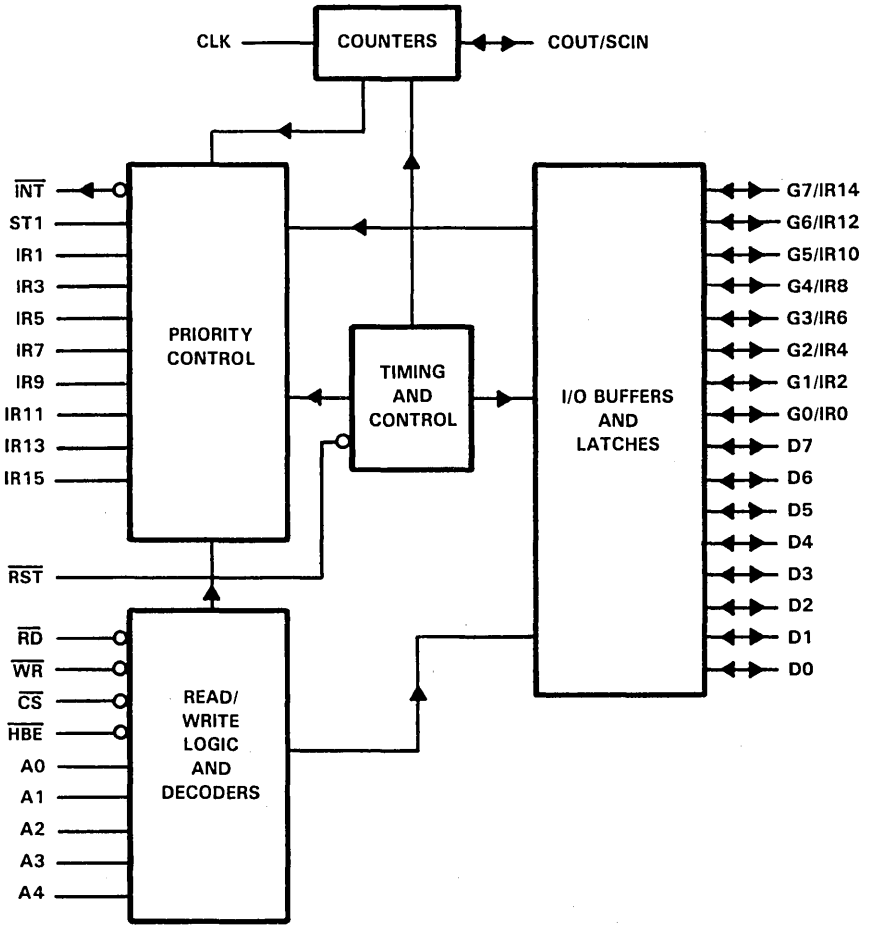


Figure 1. TI32202W ICU Block Diagram

3.5.1.1 I/O Buffers and Latches

The I/O buffers and latches block is the interface with the system data bus. It contains bidirectional buffers for the data I/O pins. It also contains registers and logic circuits that control the operation of pins G0/IR0,...,G7/IR14 when the ICU is in the 8-bit bus mode.

3.5.1.2 Read/Write Logic and Decoders

The read/write logic and decoders manage all internal and external data transfers for the ICU. These include data, control, and status transfers. This circuit accepts inputs from the CPU address and control buses. In turn, it issues commands to access the internal registers of the ICU.

3.5.1.3 Timing and Control

The timing and control block contains status elements that select the ICU operating mode. It also contains state machines that generate all the necessary sequencing and control signals.

3.5.1.4 Priority Control

The priority control block contains 16 units, one for each interrupt position. These units provide the following functions:

1. Sensing the various forms of hardware interrupt signals, e.g., level (high/low) or edge (rising/falling)
2. Resolving priorities and generating an interrupt request to the CPU
3. Handling cascaded arrangements
4. Enabling software interrupts
5. Providing for an automatic return from interrupt
6. Enabling the assignment of any interrupt position to the internal counters
7. Providing for rearrangement of priorities by assigning the first priority to any interrupt position
8. Enabling automatic rotation of priorities

3.5.1.5 Counters

This block contains two 16-bit counters, called the H-counter and the L-counter. These are down counters that count from an initial value to zero. Both counters have a 16-bit register (designated HCSV and LCSV) for loading their restarting values. They also have registers containing the current count values (HCCV and LCCV). Both sets of registers are fully described in section 3.5.2.6.

The counters are under program control and can be used to generate interrupts. When the count reaches zero, either counter can generate an interrupt request to any of the 16 interrupt positions. The counter then reloads the start value from the appropriate registers and resumes counting. Figure 2 shows typical counter output signals available from the TI32202W.

The maximum input clock frequency is 2.5 MHz.

A divide-by-four prescaler is also provided. When the prescaler is used, the input clock frequency can be up to 10 MHz.

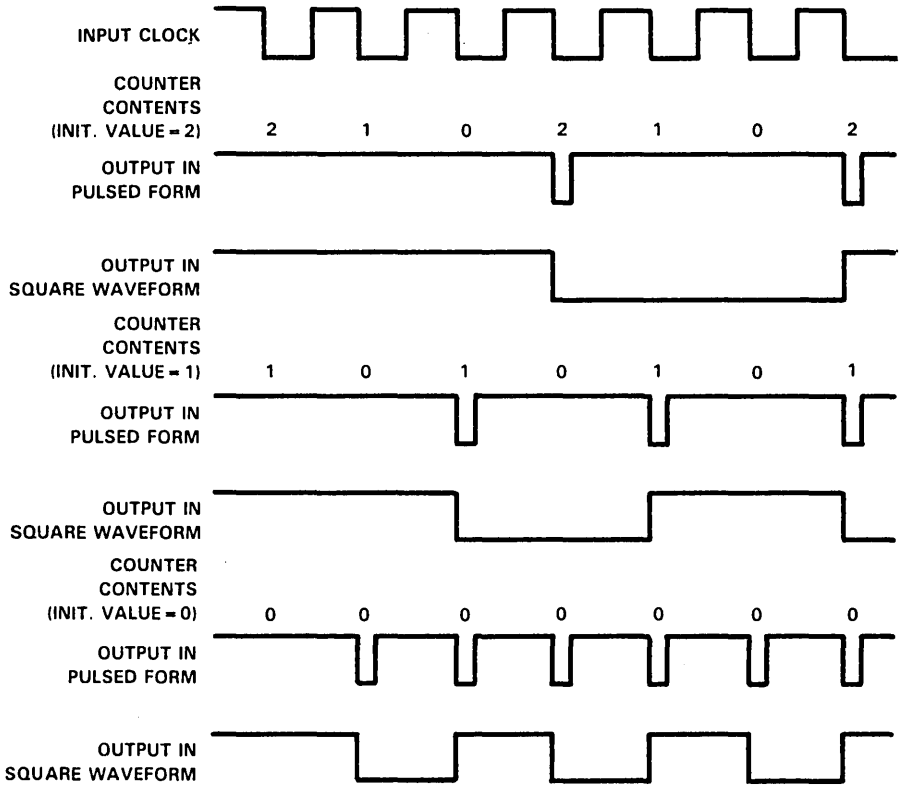


Figure 2. Counter Output Signals in Pulsed Form and Square Waveform for Three Different Initial Values

When intervals longer than those provided by a 16-bit counter are needed, the L- and H-counters can be concatenated to form a 32-bit counter. In this case, both counters are controlled by the H-counter control bits. Figure 3 illustrates counter read/write operations. For additional information, refer to the discussion of the Counter Control register in section 3.5.2.6.16.

3.5.2 Functional Description

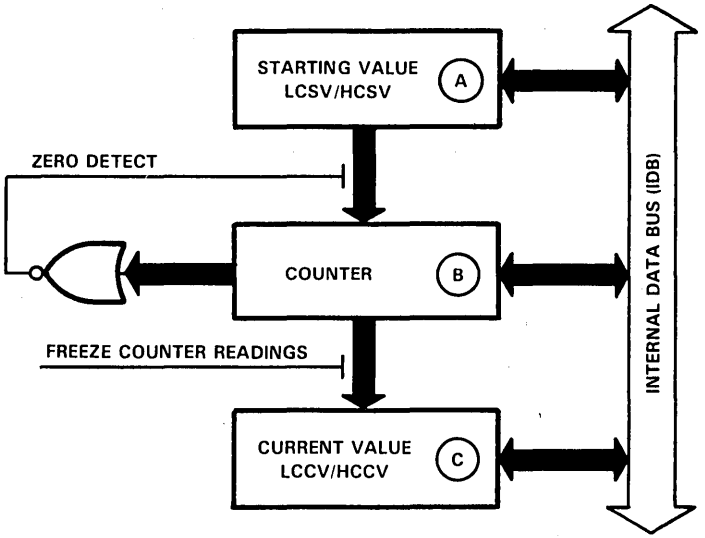
3.5.2.1 Reset

The ICU is reset when a logic low signal is present on the $\overline{RST}$ pin. At reset, most internal ICU registers are affected, and the ICU becomes inactive.

3.5.2.2 Initialization

After reset, the CPU must initialize the TI32202W to establish its configuration. Proper initialization requires knowledge of the ICU register's formats (refer to Figure 14).

The operation sequence shown in Figure 14 ensures that all counter output pins remain inactive until the counters are completely initialized.



BASIC OPERATIONS:

WRITING TO LCSV/HCSV	A ← IDB
READING LCSV/HCSV	A → IDB
WRITING TO LCCV/HCCV	B ← IDB
(only possible when counters are halted)	C ← IDB
READING LCCV/HCCV	C → IDB
(only possible when counter readings are frozen)	
COUNTER COUNTS AND READINGS ARE NOT FROZEN	C ← B
COUNTER RELOADS STARTING VALUE	B ← A
(occurs on the clock cycle following the one where it reaches zero)	

Figure 3. Counter Configuration and Basic Operations

3.5.2.3 Vectored Interrupt Handling

For details on the operation of the vectored interrupt mode for a particular TI32000 microprocessor family CPU, refer to the applicable section for that CPU. In this discussion, it is assumed that the TI32202W is working with a CPU in the vectored interrupt mode. Several ICU applications are discussed, including noncascaded and cascaded operation. Figures 4, 5, and 6 show typical configurations of the ICU used with the TI32016T CPU.

A peripheral device issues an interrupt request by sending the proper signal to one of the TI32202W interrupt inputs. If the interrupt input is not masked, the ICU activates its Interrupt ($\overline{INT}$) output pin and generates an interrupt vector byte. The interrupt vector byte identifies the interrupt source in its four least significant bits. When the CPU detects a low level on its Interrupt input pin, it performs one or two interrupt-

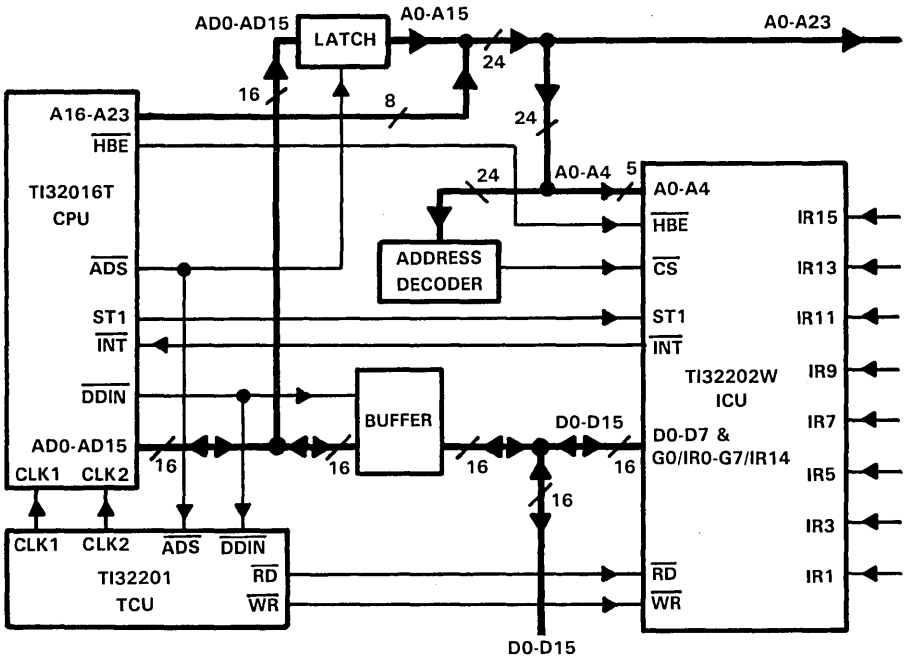


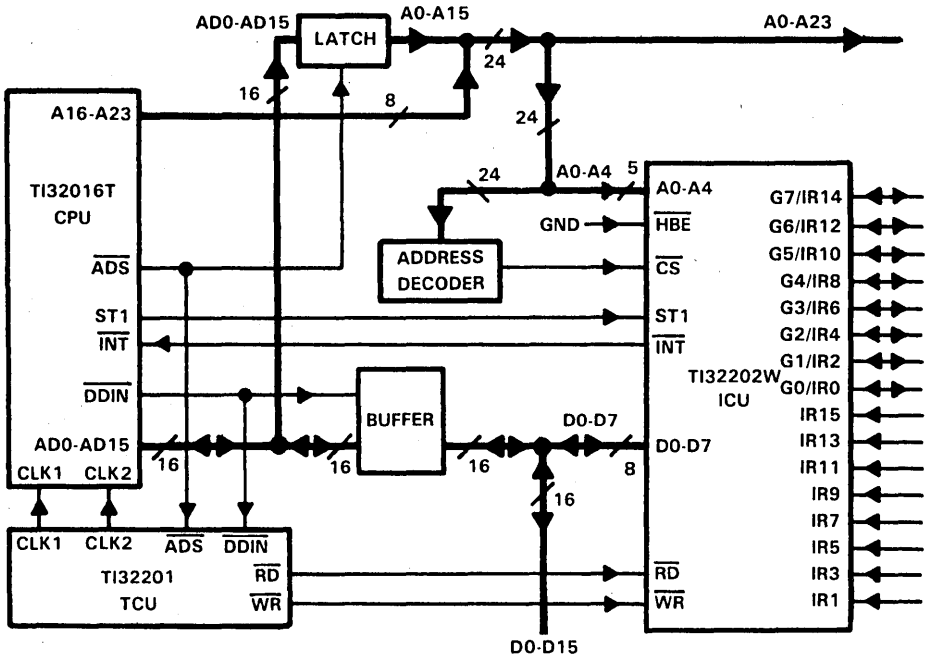
Figure 4. Interrupt Control Unit Connections in 16-Bit Bus Mode

acknowledge cycles depending on whether the interrupt request is from the master ICU or a cascaded ICU. Figure 7 shows a flowchart of a typical CPU Interrupt-Acknowledge sequence.

In general, vectored interrupts are serviced by interrupt routines stored in system memory. The Dispatch Table stores up to 256 external procedure descriptors for the various service procedures. The CPU INTBASE register points to the top of the Dispatch Table. Figure 8 shows the layout of the Dispatch Table. This figure also shows the layout of the Cascade Table, which is discussed with ICU cascaded operation.

3.5.2.3.1 Noncascaded Operation

Whenever an interrupt request from a peripheral device is issued directly to the master ICU, a noncascaded interrupt request to the CPU results. In a system using a single TI32202W, up to 16 interrupt requests can be prioritized. Upon receipt of an interrupt request on the $\overline{INT}$ pin, the CPU performs a Master Interrupt-Acknowledge bus cycle, reading a vector byte from address $FFFE00_{16}$. This vector is then used as an index into the dispatch table in order to find the External Procedure Descriptor for the proper interrupt service procedure. The service procedure eventually returns via the Return-from-Interrupt (RET) instruction, which performs a Return-from-Interrupt bus cycle, informing the ICU that it may re-prioritize any interrupt requests still pending. Figure 9 shows a typical CPU RETI sequence. In a system with only one ICU, the vectors provided must be in the range of 0 through 127; this can be ensured by writing 0XXXXXXX into the SVCT register. By providing a negative vector value, the master ICU flags the interrupt source as a cascaded ICU.



NOTE: In the 8-Bit Bus Mode the Master ICU Registers appear at even addresses (A0 = 0) since the ICU communicates with the least significant byte of the CPU data bus.

Figure 5. Interrupt Control Unit Connections in 8-Bit Bus Mode

3.5.2.3.2 Cascaded Operation

In cascaded operation, one or more of the interrupt inputs of the master ICU are connected to the Interrupt output ($\overline{INT}$) pin of one or more cascaded ICUs. Up to 16 cascaded ICUs can be used, giving a system total of 256 interrupts.

Note:
The number of cascaded ICUs is practically limited to 15 because the Dispatch Table for the TI32016T CPU is constructed with entries 1 through 15 either used for NMI and Trap descriptors, or reserved for future use. Interrupt position 0 of the master ICU should not be cascaded, so it can be vectored through Dispatch Table entry 0, reserved for nonvectored interrupts. In this case, the nonvectored interrupt entry (entry 0) is also available for vectored interrupt operation, since the CPU is operating in the vectored interrupt mode.

The address of the master ICU should be $FFFE00_{16}$. Cascaded ICUs can be located at any system address. A list of cascaded ICU addresses is maintained in the Cascade Table as a series of sixteen 32-bit entries.

Note:
The CPU status corresponding to both master interrupt acknowledge and return from interrupt bus cycles, as well as address bit A8, could be used to generate the chip select ($\overline{CS}$) signal for accessing the master ICU during one of the above cycles. In this case, the master ICU can reside at any system address. The only limitation is that

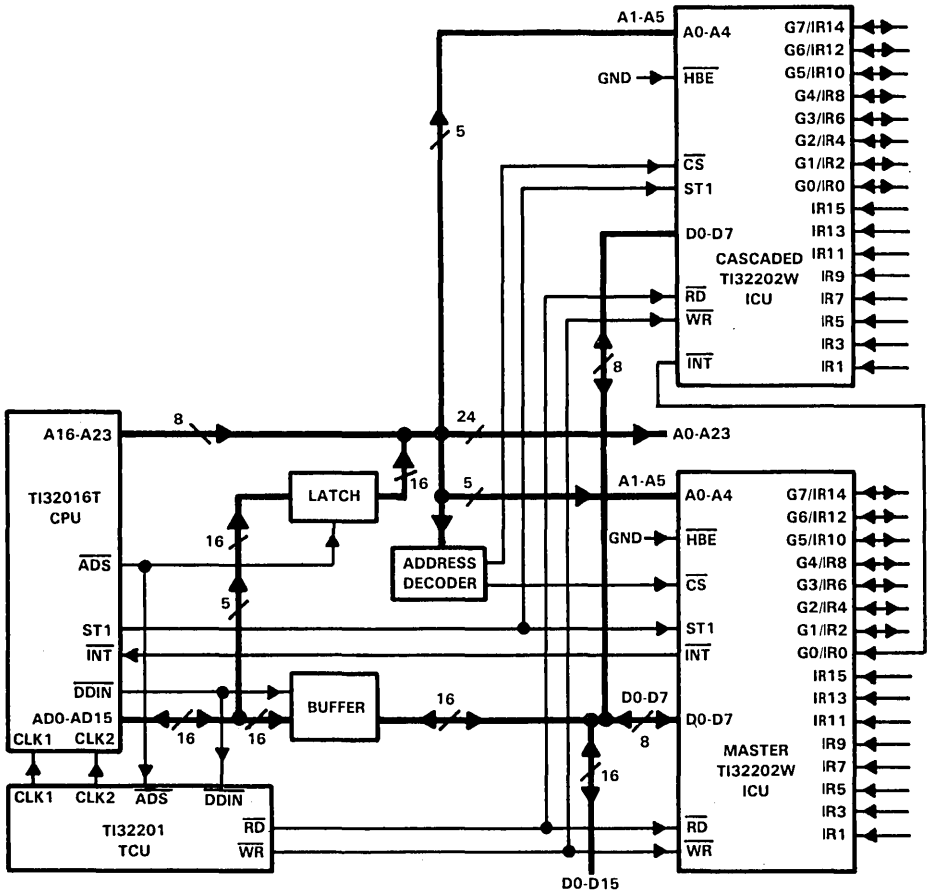
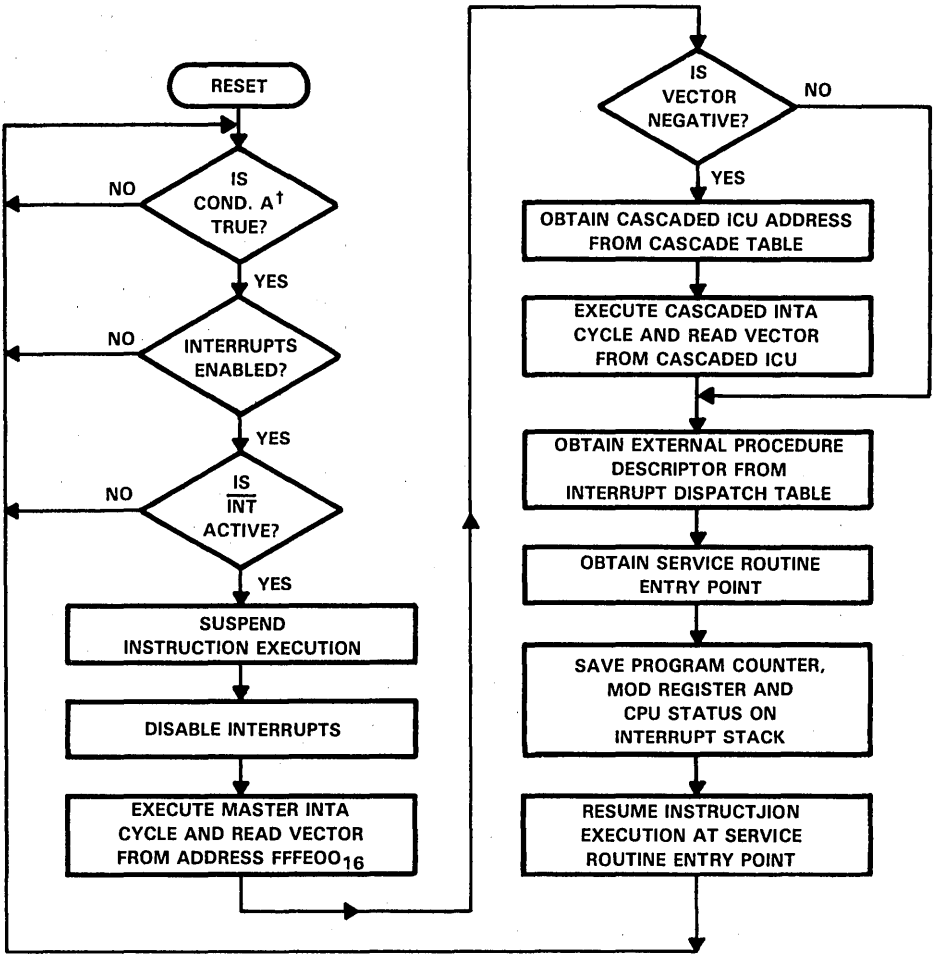


Figure 6. Cascaded Interrupt Control Unit Connections in 8-Bit Bus Mode

the least significant 5 or 6 address bits (6 in the 8-bit bus mode) must be zero. Address bit A8 must be decoded to prevent an NMI bus cycle from reading the hardware vector register of the ICU. This could happen, since the TI32016T CPU performs a dummy read cycle from address FFFF00$_{16}$, with the same status as a master INTA cycle, when a nonmaskable-interrupt is acknowledged.

The master ICU maintains a list (in the CSRC register pair) of its interrupt positions that are cascaded. When a cascaded interrupt input is active, the master ICU activates its interrupt output and the CPU responds with a Master Interrupt-Acknowledge cycle. However, instead of generating a positive interrupt vector, the master ICU generates a negative Cascade Table index.

The CPU interprets the negative number returned from the master ICU as an index into the Cascade Table. The Cascade Table is located in a negative direction from the Dispatch Table, and it contains the logical addresses of the hardware vector registers for any cascaded TI32202Ws in the system. Thus, the Cascade Table index supplied by the master ICU identifies the cascaded ICU that requested the interrupt.



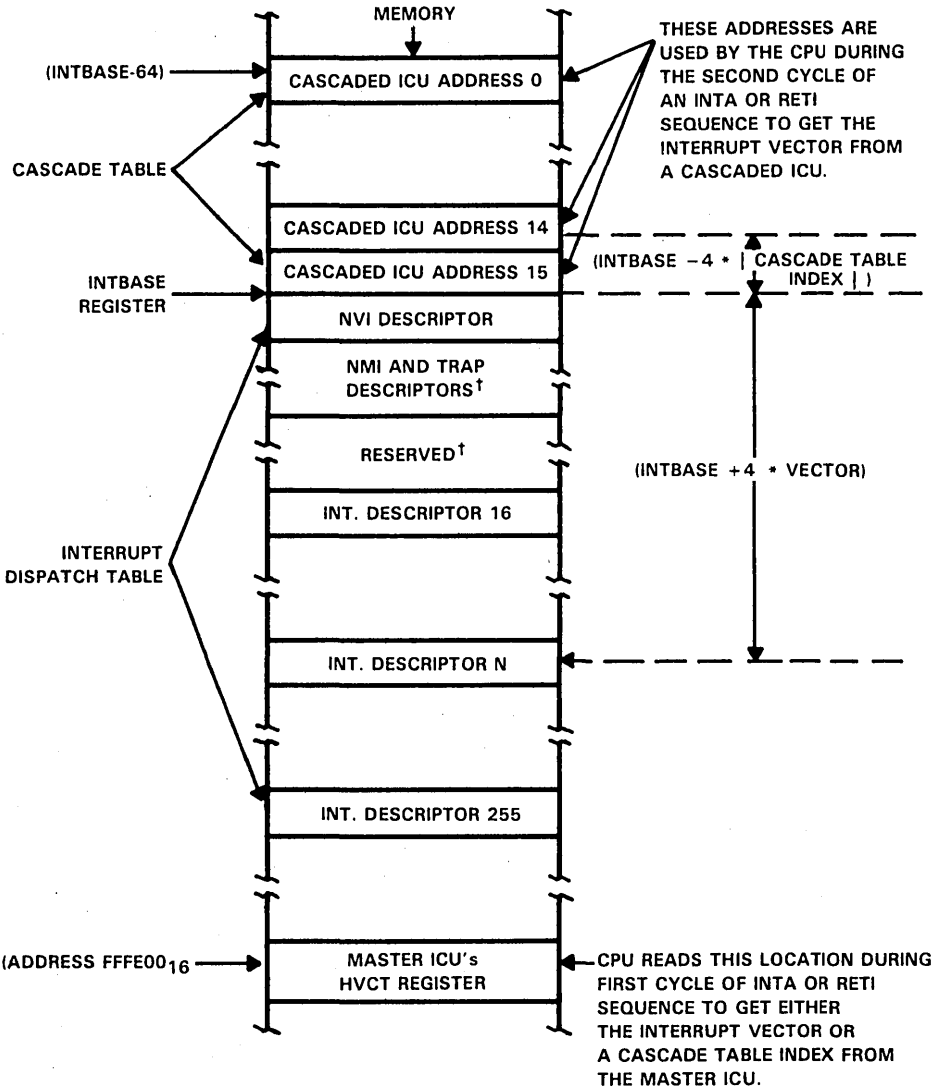
†Cond. A is true if current instruction is terminated or an interruptible point in a string instruction is reached.

Figure 7. CPU Interrupt-Acknowledge Sequence

Once the cascaded ICU is identified, the CPU performs a Cascaded Interrupt-Acknowledge cycle. During this cycle, the CPU reads the final vector value directly from the cascaded ICU, and uses it to access the Dispatch Table. Each cascaded ICU has its own set of 16 unique interrupt vectors, one vector for each of its 16 interrupt positions.

The CPU interprets the vector value read during a Cascaded Interrupt-Acknowledge cycle as an unsigned number. Thus, this vector can be in the range 0 through 255.

When a cascaded interrupt service routine completes its task, it must return control to the main program with the same RETI instruction used in noncascaded interrupt service routines. However, when the CPU performs a Master Return-from-Interrupt



†Table entries 1 to 15 should not be used by the ICU since they contain NMI and Trap Descriptors or are reserved for future use.

Figure 8. Interrupt Dispatch and Cascade Tables

cycle, the CPU accesses the master ICU and reads the negative Cascade Table index identifying the cascaded ICU that originally received the interrupt request. Using the cascaded ICU address, the CPU now performs a Cascaded Return-from-Interrupt cycle, informing the cascaded ICU that the service routine is over. The byte provided by the cascaded ICU during this cycle is ignored.

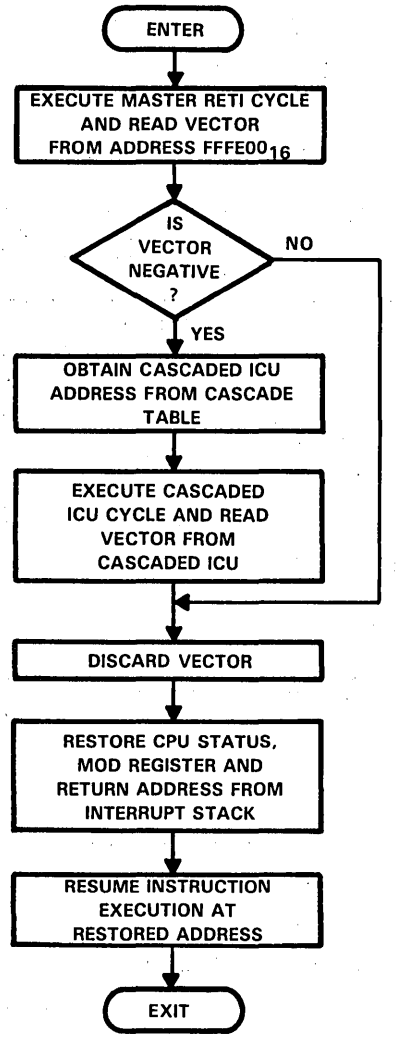


Figure 9. CPU Return from Interrupt Sequence

3.5.2.4 Internal ICU Operating Sequence

The TI32202W ICU accepts two interrupt types, software and hardware.

Software interrupts are initiated when the CPU sets the proper bit in the Interrupt Pending (IPND) registers (R6, R7), located in the ICU. Bits are set and reset by writing the proper byte to either R6 and R7. Software interrupts can be masked, by setting the proper bit in the mask registers (R10, R11).

Hardware interrupts can be either internal or external to the ICU. Internal ICU hardware interrupts are initiated by the on-chip counter outputs. External hardware interrupts are initiated by devices external to the ICUs that are connected to any of the ICU interrupt input pins.

Hardware interrupts can be masked by setting the proper bit in the mask registers (R10, R11). If the Freeze bit (FRZ), located in the Mode Control (MCTL) Register, is set, all incoming hardware interrupts are inhibited from setting their corresponding bits in the IPND registers. This prevents the ICU from recognizing any hardware interrupts.

When the ICU is initialized, it is enabled to accept interrupts. If an active interrupt is not masked and has a higher priority than any interrupt currently being serviced, the ICU activates its Interrupt ($\overline{INT}$) output. Figure 10 is a flowchart showing the ICU interrupt-acknowledge sequence.

The CPU responds to the active $\overline{INT}$ line by performing an Interrupt-Acknowledge bus cycle. During this cycle, the ICU clears the IPND bit corresponding to the active interrupt position and sets the corresponding bit in the Interrupt In-Service (ISRV) registers. The ISRV bit remains set until the CPU performs a RETI bus cycle triggered by the completion of the interrupt service routine for the active interrupt position. Figure 11 is a flowchart showing ICU operation during a RETI bus cycle.

When the ISRV bit is set, the $\overline{INT}$ output is disabled. This output remains inactive until a higher priority interrupt position becomes active, or the ISRV bit is cleared.

3.5.2.5 Interrupt Priority Modes

The TI32202W ICU can operate in one of four interrupt priority modes: fixed-priority, auto-rotate, special mask, or polling. Each mode is described in the following paragraphs.

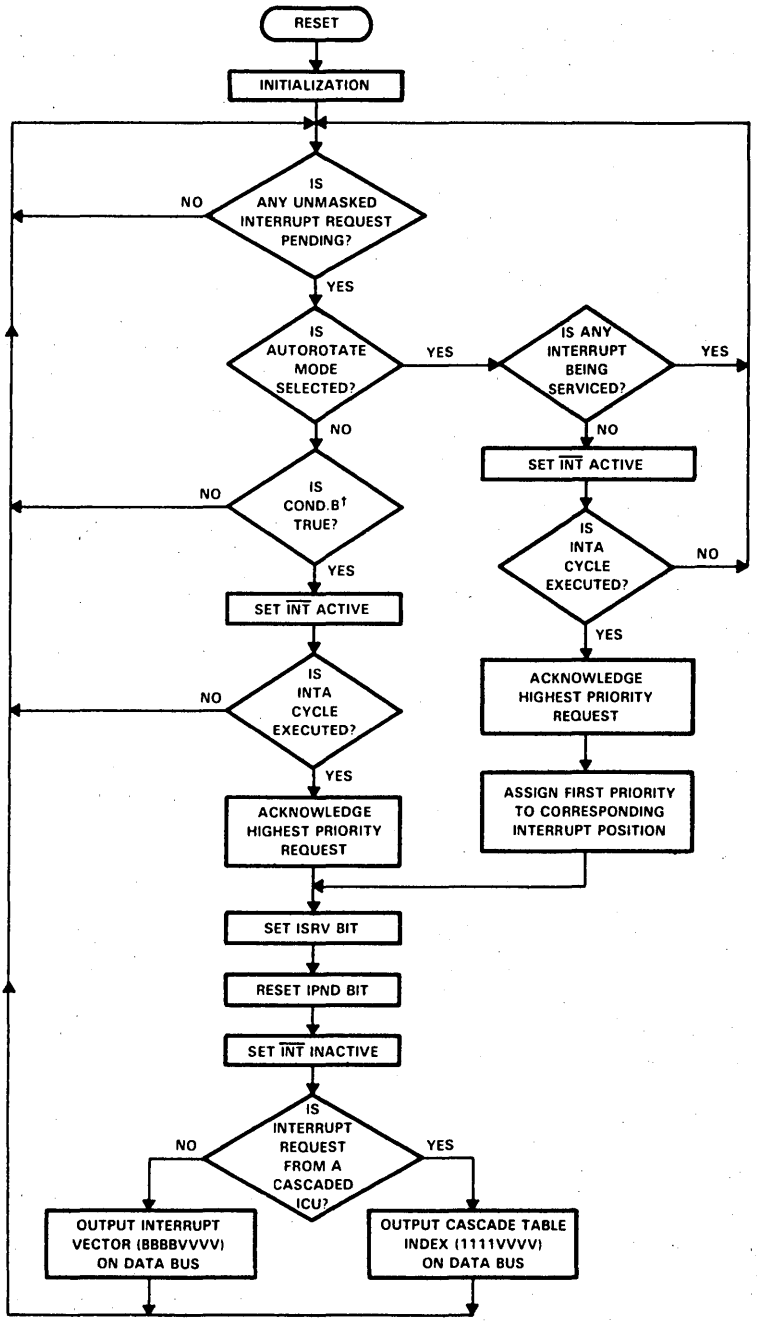
3.5.2.5.1 Fixed-Priority Mode

In the fixed-priority mode (also called fully-nested mode), each interrupt position is ranked in priority from 0 to 15, with 0 being the highest priority. In this mode, the processing of lower priority interrupts is nested with higher priority interrupts. Thus, while an interrupt is being serviced, any other interrupts of the same or lower priority are inhibited. However, the ICU does recognize higher priority interrupt requests.

When the interrupt service routine executes its RETI instruction, the corresponding ISRV bit is cleared. This allows any lower priority interrupt request to be serviced by the CPU.

At reset, the default priority assignment gives interrupt IR0 priority 0 (highest priority), interrupt IR1 priority 1, and so forth. Interrupt IR15 is, of course, assigned priority 15, the lowest priority. The default priority assignment can be altered by writing an appropriate value into register FPRT (L) as explained in section 3.5.2.6.9.

Note:
When the ICU generates an interrupt request to the CPU for a higher priority interrupt while a lower priority interrupt is still being serviced by the CPU, the CPU responds to the interrupt request only if its internal interrupt enable flag is set. Normally, this flag is reset at the beginning of an Interrupt-Acknowledge cycle and set during the RETI cycle. If the CPU is to respond to higher priority interrupts during any interrupt service routine, the service routine must set the internal CPU interrupt enable flag, as soon as desired during the service routine.



†Cond. B is true if no interrupt is being serviced, or if there is a pending unmasked interrupt whose priority is higher than that of the interrupt being serviced.

Figure 10. ICU Interrupt-Acknowledge Sequence

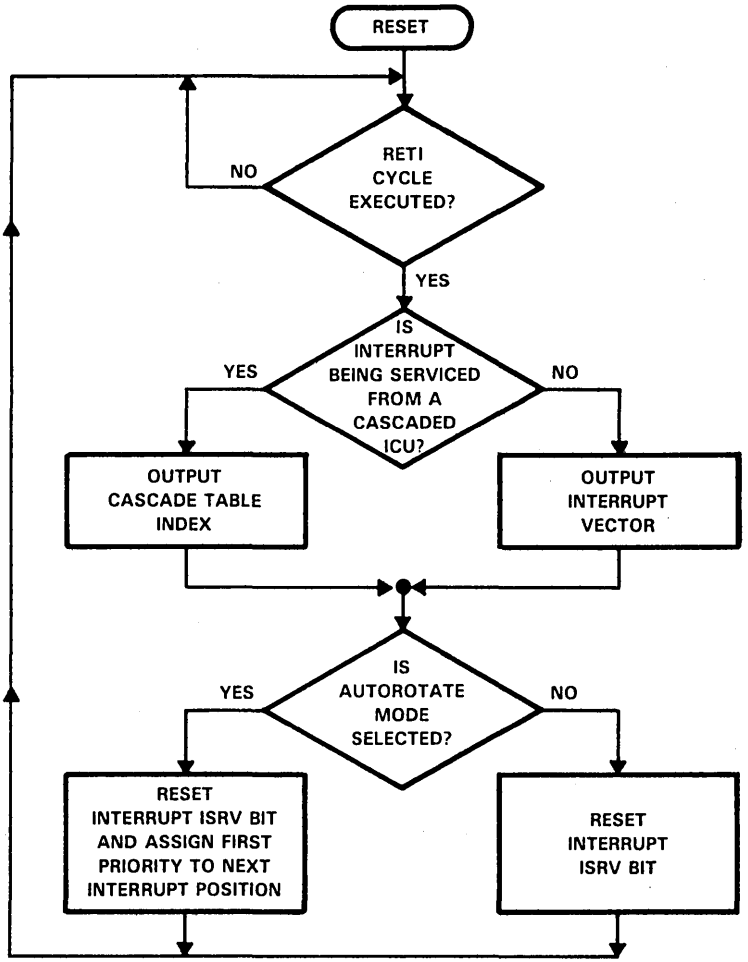


Figure 11. ICU Return from Interrupt Sequence

3.5.2.5.2 Auto-Rotate Mode

The auto-rotate mode is selected when the NTAR bit is set to 0, and is automatically entered after reset. In this mode, an interrupt source position is automatically assigned lowest priority after a request at that position has been serviced. Highest priority then passes to the next lower priority position. For example, when servicing of the interrupt request at position 3 is completed (ISRV bit 3 is cleared), interrupt position 3 is assigned the lowest priority and position 4 assumes the highest priority. The nesting of interrupts is inhibited, since the interrupt being serviced always has the highest priority.

This mode is used when the interrupting devices have to be assigned equal priority. A device requesting an interrupt will have to wait, in the worst case, until each of the 15 other devices has been serviced at most once.

3.5.2.5.3 Special Mask Mode

The special mask mode is used when it is necessary to dynamically alter the ICU priority structure while an interrupt is being serviced. For example, it may be desired in a particular interrupt service routine to enable lower priority interrupts during a part of the routine. To do so, the ICU must be programmed in fixed-priority mode and the interrupt service routine must control its own in-service bit in the ISRV registers.

The bits of the ISRV registers are changed with either the Set Bit Interlocked (SBITIW) or Clear Bit Interlocked (CBITIW) instructions. The in-service bit is cleared to enable lower priority interrupts and set to disable them.

Note:
For proper operation of the ICU, an interrupt service routine must set its ISRV bit before executing the RETI instruction. This prevents the RETI cycle from clearing the wrong ISRV bit.

3.5.2.5.4 Polling Mode

The polling mode gives complete control of interrupt priority to the system software. Either some or all of the interrupt positions can be assigned to the polling mode. To assign all interrupt positions to the polling mode, the CPU interrupt enable flag is reset. To assign only some of the interrupt positions to the polling mode, the desired interrupt positions are masked in the Interrupt Mask (IMSK) registers. In either case, the polling operation consists of reading the Interrupt Pending (IPND) registers.

If necessary, the IPND read can be synchronized by setting the Freeze (FRZ) bit in the Mode Control (MCTL) register. This prevents any change in the IPND registers during the read. The FRZ bit must be reset after the polling operation so the IPND contents can be updated. If an edge-triggered interrupt occurs while the IPND registers are frozen, the interrupt request is latched, and transferred to the IPND registers as soon as FRZ is reset.

The polling mode is useful when a single routine is used to service several interrupt levels.

3.5.2.6 Register Functions

The TI32202W has thirty-two 8-bit registers that can be accessed either individually or in pairs. In 16-bit data bus mode, register pairs can be accessed with the CPU word or double-word reference instructions. Figure 12 shows the ICU internal registers, and lists the name, function, and offset address for each register.

Because some registers hold similar data, they are grouped into functional pairs and assigned a single name. However, if a single register in a pair is referenced, either an L or an H is appended to the register name. The letters are placed in parentheses and stand for the low order 8 bits (L) and the high order 8 bits (H). For example, register R6, part of the Interrupt Pending (IPND) register pair, is referred to individually as IPND(L).

The following paragraphs give detailed descriptions of the registers shown in Figure 12.

REG. NUMBER AND ADDRESS IN HEX.		REG. NAME	REG. FUNCTION
	R0 (00_{16})	HVCT –	HARDWARE VECTOR
	R1 (01_{16})	SVCT –	SOFTWARE VECTOR
R3 (03_{16})	R2 (02_{16})	ELTG –	EDGE/LEVEL TRIGGERING
R5 (05_{16})	R4 (04_{16})	TPL –	TRIGGERING POLARITY
R7 (07_{16})	R6 (06_{16})	IPND –	INTERRUPTS PENDING
R9 (09_{16})	R8 (08_{16})	ISRV –	INTERRUPTS IN-SERVICE
R11 ($0B_{16}$)	R10 ($0A_{16}$)	IMSK –	INTERRUPT MASK
R13 ($0D_{16}$)	R12 ($0C_{16}$)	CSRC –	CASCADED SOURCE
R15 ($0F_{16}$)	R14 ($0E_{16}$)	FPRT –	FIRST PRIORITY
	R16 (10_{16})	MCTL –	MODE CONTROL
	R17 (11_{16})	OCASN –	OUTPUT CLOCK ASSIGNMENT
	R18 (12_{16})	CIPTR –	COUNTER INTERRUPT POINTER
	R19 (13_{16})	PDAT –	PORT DATA
	R20 (14_{16})	IPS –	INTERRUPT/PORT SELECT
	R21 (15_{16})	PDIR –	PORT DIRECTION
	R22 (16_{16})	CCTL –	COUNTER CONTROL
	R23 (17_{16})	CICTL –	COUNTER INTERRUPT CONTROL
R25 (19_{16})	R24 (18_{16})	LCSV –	L-COUNTER STARTING VALUE
R27 ($1B_{16}$)	R26 ($1A_{16}$)	HCSV –	H-COUNTER STARTING VALUE
R29 ($1D_{16}$)	R28 ($1C_{16}$)	LCCV –	L-COUNTER CURRENT VALUE
R31 ($1F_{16}$)	R30 ($1E_{16}$)	HCCV –	H-COUNTER CURRENT VALUE

Figure 12. ICU Internal Registers

3.5.2.6.1 HVCT — Hardware Vector Register (R0)

The HVCT register is a single register that contains the interrupt vector byte supplied to the CPU during an Interrupt-Acknowledge (INTA) or Return-from-Interrupt (RETI) cycle. The HVCT bit map is shown below:

7	6	5	4	3	2	1	0
B	B	B	B	V	V	V	V

The BBBB field is the bias which is programmed by writing $BBBB0000_2$ to the SVCT register (R1). The VVVV field identifies one of the 16 interrupt positions. The HVCT register provides various information to the CPU, as shown in Table 1.

Note:
The ICU always interprets a read of the HVCT register as either an INTA or RETI cycle. Since these cycles cause internal changes to the ICU, normal programs must never read the ICU HVCT register.

Table 1. HVCT Register Data Coding

	INTA CYCLE (ST1 = 0)		RETI CYCLE (ST1 = 1)	
	Highest priority pending interrupt is from:		Highest priority in-service interrupt was from:	
BBBB	cascaded ICU	any other source	cascaded ICU	any other source
	1111	programmed bias*	1111	programmed bias*
VVVV	encoded value of the highest priority pending interrupt		encoded value of the highest priority in-service interrupt	

*The Programmed bias for the master ICU must range from 0000 to 0111_2 because the CPU interprets a one in the most significant bit position as a Cascade Table Index indicator for a cascaded ICU.

3.5.2.6.2 SVCT — Software Vector Register (R1)

The SVCT register contains a copy of the value stored in the HVCT register. This allows the programmer to read the contents of the HVCT register without initiating an INTA or RETI cycle in the ICU. It also allows a programmer to change the BBBB field of the HVCT register. The bit map of the SVCT register is the same as for the HVCT register.

During a write to the SVCT register, the four least significant bits are unaffected and the four most significant bits are written into both SVCT and HVCT (R1 and R0).

The SVCT register is continuously updated by the ICU. The four least significant bits always contain the vector value that would be returned to the CPU if an INTA or RETI cycle were executed. Therefore, when reading the SVCT register, the state of the CPU ST1 pin is used to select either pending interrupt data or in-service interrupt data. For example, if the SVCT register is read with ST1 = 0 (as for an INTA cycle), the VVVV field contains the encoded value of the highest priority pending interrupt. On the other hand, if the SVCT register is read with ST1 = 1, the VVVV field contains the encoded value of the highest priority in-service interrupt.

Note:
If the CPU ST1 output is connected directly to the ICU ST1 input, the vector read from SVCT is always the RETI vector. If both the INTA and RETI vectors are desired, additional logic must be added to drive the ICU ST1 input. A typical circuit is shown in Figure 13. In this circuit, the state of the ICU ST1 input is controlled by both the CPU ST1 output and the selected address bit.

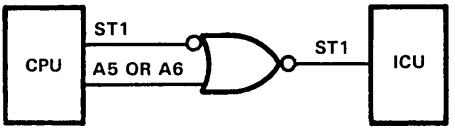


Figure 13. Typical Circuit to Show RETI and INTA Vector Capability

3.5.2.6.3 ELTG — Edge/Level Triggering Registers (R2, R3)

The ELTG registers determine the input trigger mode for each of the 16 interrupt inputs. Each input is assigned a bit in this register pair. An interrupt input is level-triggered if its bit in ELTG is set to 1. The input is edge-triggered if its bit is cleared. At reset, all bits in ELTG are set to 1.

Software interrupt positions are not affected by the state of their ELTG bits.

3.5.2.6.4 TPL — Triggering Polarity Registers (R4, R5)

The TPL registers determine the polarity of either the active level or the active edge for each of the 16 interrupt inputs. As with the ELTG registers, each input is assigned a bit. Possible triggering modes for the various combinations of ELTG and TPL bits are shown below.

ELTG BIT	TPL BIT	TRIGGERING MODE
0	0	Falling Edge
0	1	Rising Edge
1	0	Low Level
1	1	High Level

Software interrupt positions are not affected by their TPL bits. At reset, all TPL bits are set to 0.

Note:
Hardware interrupt inputs connected to cascaded ICUs must have their TPL bits set to 0.

3.5.2.6.5 IPND — Interrupt Pending Registers (R6, R7)

The IPND registers track interrupt requests that are pending but not yet serviced. Each interrupt position is assigned a bit in IPND. When an interrupt is pending, the corresponding bit in IPND is set. The IPND data are used by the ICU to generate interrupts to the CPU. These data are also used in polling operations.

The IPND registers are also used for requesting software interrupts. This is done by writing specially formatted data bytes to either IPND(L) or IPND(H). The formats differ for registers R6 and R7. These formats are shown below:

IPND(L) (R6)—S0000PPP
IPND(H) (R7)—S0001PPP

Where:

S = Set (S = 1) or Clear (S = 0)
PPP = is a binary number identifying one of eight bits

Note:
The data read from either R6 or R7 are different from that written to the register because the ICU returns the register contents rather than the formatted byte used to set the register bits.

The ICU automatically clears a set IPND bit when the pending interrupt request is serviced. All pending interrupts in a register can be cleared by writing the pattern 'X1XXXXXX' to it (X = don't care). To avoid conflicts with asynchronous hardware interrupt requests, the IPND registers should be frozen before pending interrupts are cleared. Refer to the Mode Control Register description for details on freezing the IPND registers.

At reset, all IPND bits are set to 0.

Note:
The edge sensing mechanism used for hardware interrupts in the TI32202W ICU is a latching device that can be cleared only by acknowledging the interrupt or by changing the trigger mode to level sensing. Therefore, before clearing pending interrupts in the IPND registers, any edge-triggered interrupt inputs must first be switched to the level-triggered mode. This clears the edge-triggered interrupts; the remaining interrupts can then be cleared in the manner described above. This applies to clearing the interrupts only. Edge-triggered interrupts can be set without changing the trigger mode.

13.5.2.6.6 ISRV — Interrupt In-Service Registers (R8, R9)

The ISRV registers track interrupt requests that are currently being serviced. Each interrupt position is assigned a bit in ISRV. When an interrupt request is serviced by the ICU, its corresponding bit is set in the ISRV registers. Before generating an interrupt to the CPU, the ICU checks the ISRV registers to ensure that no higher priority interrupt is currently being serviced.

Each time the CPU executes an RETI instruction, the ICU clears the ISRV bit corresponding to the highest priority interrupt in service. The ISRV registers can also be written into by the CPU. This is done to implement the special mask priority mode.

At reset, the ISRV registers are set to 0.

3.5.2.6.7 IMSK — Interrupt Mask Registers (R10, R11)

Each TI32202W interrupt position can be individually masked. A masked interrupt source is not acknowledged by the ICU. The IMSK registers store a mask bit for each of the ICU interrupt positions. If an interrupt position's IMSK bit is set to 1, the position is masked.

The IMSK registers are controlled by the system software. At reset, all IMSK bits are set to 1, disabling all interrupts.

3.5.2.6.8 CSRC — Cascaded Source Registers (R12, R13)

The CSRC registers track any cascaded interrupt positions. Each interrupt position is assigned a bit in the CSRC registers. If an interrupt position's CSRC bit is set, that position is connected to the $\overline{INT}$ output of another TI32202W ICU, i.e., it is a cascaded interrupt.

At reset, the CSRC registers are set to 0.

Note:
Only the Master ICU should have any CSRC bits set. If CSRC bits are set in a cascaded ICU, incorrect operation results.

3.5.2.6.9 FPRT — First Priority Registers (R14, R15)

The FPRT registers track the ICU interrupt position that currently holds first priority. Only one bit of the FPRT registers is set at one time. The set bit indicates the interrupt position with first (highest) priority.

The FPRT registers are automatically updated when the ICU is in the auto-rotate mode. The first priority interrupt can be determined by reading the FPRT registers. This operation returns a 16-bit word with only one bit set. An interrupt position can be assigned first priority by writing a formatted data byte to the FPRT(L) register. The format is shown below:

7	6	5	4	3	2	1	0
X	X	X	X	F	F	F	F

Where:
XXXX = Don't Care
FFFF = A binary number from 0 to 15 indicating the interrupt position assigned first priority.

Note:
The byte above is written only to the FPRT(L) register. Any data written to FPRT(H) is ignored.

At reset the FFFF field is set to 0, thus giving interrupt position 0 first priority.

3.5.2.6.10 MCTL — Mode Control Register (R16)

The contents of the MCTL set the operating mode of the TI32202W ICU. The MCTL bit map is shown below.

7	6	5	4	3	2	1	0
CFRZ	COUTD	COUTM	CLKM	FRZ	unused	NTAR	T16N8

CFRZ Determines whether or not the TI32202W counter readings are frozen. When frozen, the counters continue counting but the LCCV and HCCV registers are not updated. Reading of the true value of LCCV and HCCV is possible only while they are frozen.

CFRZ = 0, then LCCV and HCCV are not frozen
CFRZ = 1, then LCCV and HCCV are frozen

COUTD Determines whether the COUT/SCIN pin is an input or an output. COUT/SCIN should be used as an input only for testing purposes. In this case an external sampling clock must be provided otherwise hardware interrupts will not be recognized.

COUTD = 0, then COUT/SCIN is Output
COUTD = 1, then COUT/SCIN is Input

COUTM When the COUT/SCIN pin is programmed as an output (COUTD = 0), this bit determines whether the output signal is in pulsed form or in square wave form.

COUTM = 0, then Square Waveform
COUTM = 1, then Pulsed Form

CLKM Used only in the 8-bit bus mode. This bit controls the clock wave form on any of the pins G0/IR0,...,G3/IR6 programmed as counter output.

CLKM = 0, then Square Waveform
CLKM = 1, then Pulsed Form

FRZ Freeze Bit. In order to allow a synchronous reading of the interrupt pending registers (IPND), their status may be frozen, causing the ICU to ignore incoming requests. This is of special importance if a polling method is used.

FRZ = 0, then IPND not frozen
FRZ = 1, then IPND frozen

NTAR Determines whether the ICU is in the auto-rotate or fixed-priority mode. In auto-rotate mode, the interrupt source at the highest priority position, after being serviced, is automatically assigned lowest priority. In this mode, the interrupt in service always has highest priority and nesting of interrupts is therefore inhibited.

NTAR = 0, then Auto-Rotate Mode
NTAR = 1, then Fixed Mode

T16N8 Controls the data bus mode of operation.

T16N8 = 0, then 8-Bit Bus Mode
T16N8 = 0, then 16-Bit Bus Mode

At reset, all MCTL bits except COUTD, are reset to 0. COUTD is set to 1.

3.5.2.6.11 OCASN — Output Clock Assignment Register (R17)

Used only in the 8-bit Bus Mode. The four least significant bits of this register control the output clock assignments on pins G0/IR0,...,G3/IR6. If any of these bits is set

to 1, the clock generated by either the H-Counter or the H + L-Counter will be output to the corresponding pin. The four most significant bits of OCASN are not used. At Reset, the four least significant bits are set to 0.

Note:
The interrupt sensing mechanism on pins G0/IR0,...,G3/IR6 is not disabled when any of these pins is programmed as clock output. Thus, to avoid spurious interrupts, the corresponding bits in register IPS should also be set to zero.

3.5.2.6.12 CIPTR – Counter Interrupt Pointer Register (R18)

The CIPTR register tracks the assignment of counter outputs to interrupt positions. A bit map of this register is shown below.

7	6	5	4	3	2	1	0
H	H	H	H	L	L	L	L

Where:

HHHH = A 4-bit binary number identifying the interrupt position assigned to the H-Counter (or the H + L-Counter if the counters are concatenated).

LLLL = A 4-bit binary number identifying the interrupt position assigned to the L-Counter.

Note:
Assignment of a counter output to an interrupt position also requires control bits to be set in the CICTL register. If a counter output is assigned to an interrupt position, external hardware interrupts at that position are ignored.

At reset, all bits in the CIPTR are set to 1. (This means both counters are assigned to interrupt position 15.)

3.5.2.6.13 PDAT – Port Data Register (R19)

Used only in the 8-bit Bus Mode. This register is used to input or output data through any of the pins G0/IR0,...,G7/IR14 programmed as I/O ports by the IPS register. Any pin programmed as an output delivers the data written into PDAT. The input pins ignore it. Reading PDAT provides the logical value of all I/O pins, INPUT and OUTPUT.

3.5.2.6.14 IPS – Interrupt/Port Select Register (R20)

Used only in the 8-bit Bus Mode. This register controls the function of the pins G0/IR0,...,G7/IR14. Each of these pins is individually programmed as an I/O port, if the corresponding bit of IPS is 0; as an interrupt source, if the corresponding bit is 1. The assignment of the H-Counter output to G0/IR0,...,G3/IR6 by means of reg. OCASN overrides the assignment to these pins as I/O ports or interrupt inputs.

At Reset, all the IPS bits are set to 1.

Note:
Whenever a bit in the IPS register is set to zero, to program the corresponding pin as an I/O port, any pending interrupt on the corresponding interrupt position will be cleared.

3.5.2.6.15 PDIR — Port Direction Register (R21)

Used only in the 8-bit bus mode. This register determines the direction of any of the pins G0/IR0,...,G7/IR14 programmed as I/O ports by the IPS register. A logic 1 indicates an input, while a logic 0 indicates an output.

At Reset, all the PDIR bits are set to 1.

3.5.2.6.16 CCTL — Counter Control Register (R22)

The CCTL register controls the operating modes of the counters. A bit map of CCTL is shown below.

7	6	5	4	3	2	1	0
CCON	CFNPS	COUT1	COUT0	CRUNH	CRUNL	CDCRH	CDCRL

CCON — Determines whether the counters are independent or concatenated to form a single 32-bit counter (H + L-Counter). If a 32-bit counter is selected, the bits corresponding to the H-Counter will control the H + L-Counter, while the bits corresponding to the L-Counter are not used.

CCON = 0, then Two 16-bit Counters
CCON = 1, then One 32-bit Counter

CFNPS — Determines whether the external clock is prescaled or not.

CFNPS = 0, then Clock Prescaled (divided by 4)
CFNPS = 1, then Clock Not Prescaled.

COUT1 & COUT0 — These bits are effective only when the COUT/SCIN pin is programmed as an output (COUTD bit in reg. MCTL is 0). Their logic levels are decoded to provide different outputs for COUT/SCIN, as detailed in the table below:

COUT1	COUT0	COUT/SCIN Output Signal
0	0	Internal Sampling Oscillator
0	1	Zero Detect Of L-Counter
1	0	Zero Detect Of H-Counter
1	1	Zero Detect Of H + L-Counter

If the H- and L-Counters are not concatenated and COUT1/COUT0 are both 1, the COUT/SCIN pin is active when either counter reaches zero.

CRUNH — Determines the state of either the H-Counter or the H + L-Counter, depending upon the status of CCON.

CRUNH = 0, then H-Counter or H + L-Counter Halted
CRUNH = 1, then H-Counter or H + L-Counter Running

CRUNL — Effective only when CCON = 0. This bit determines whether the L-Counter is running or halted.

CRUNL = 0, then L-Counter Halted
CRUNL = 1, then L-Counter Running

CDCRH Effective only when CRUNH = 0 (Counter Halted). This bit is the single cycle decrement signal for either the H-Counter or the H + L-Counter.

CDCRH = 0, then No Effect
CDCRH = 1, then Decrement H-Counter or H + L-Counter

CDCRL Effective only when CRUNL = 0 and CCON = 0. This bit is the single cycle decrement signal for the L-Counter.

CDCRL = 0, then No Effect
CDCRL = 1, then Decrement L-Counter

Note:
The bits CDCRL and CDCRH are set when a logic 1 is written into them, but they are automatically cleared after the end of the write operation. This is needed to accomplish the decrement operation. Therefore, these bits always contain 0 when read.

Reset does not affect the CCTL bits.

3.5.2.6.17 CICTL — Counter Interrupt Control Register (R23)

The CICTL register controls the counter interrupts and records counter interrupt status. Interrupts can be generated from either of the 16-bit counters. When the counters are concatenated, the interrupt control is through the H-Counter control bits. In this case the CIEL bit should be set to zero to avoid spurious interrupts from the L-Counter. A bit map of the CICTL register is shown following.

7	6	5	4	3	2	1	0
CERH	CIRH	CIEH	WENH	CERL	CIRL	CIEL	WENL

CERH H-Counter Error Flag. This bit is set (1) when a second interrupt request from the H-Counter (or H + L-Counter) occurs before the first request is acknowledged.

CIRH H-Counter Interrupt Request. This bit is set (1) when an interrupt is pending from the H-Counter (or H + L-Counter). It is automatically reset when the interrupt is acknowledged.

CIEH H-Counter Interrupt Enable. When it is set, the H-Counter (or H + L-Counter) interrupt is enabled.

WENH H-Counter Control Write Enable. When WENH is set to 1, bits CERH, CIRH, and CIEH can be written.

CERL L-Counter Error Flag. This bit is set to 1 when a second interrupt request from the L-Counter occurs before the first request is acknowledged.

CIRL L-Counter Interrupt Request. This bit is set to 1 when an interrupt is pending from the L-Counter. It is automatically reset when the interrupt is acknowledged.

CIEL L-Counter Interrupt Enable. When this bit is set (1), the L-Counter interrupt is enabled.

WENL L-Counter Control Write Enable. When WENL is set to 1, bits CERL, CIRL, and CIEL can be written.

Note:
Setting the write enable bits (WENH or WENL) and writing any of the other CICTL bits are concurrent operations. That is, the ICU will ignore any attempt to alter CICTL bits if the proper write enable bit is not set in the data byte.

At reset, all CICTL bits are set to 0. However, if the counters are running, the bits CIRL, CERL, CIRH, and CERH may be set again after the reset signal is removed.

3.5.2.6.18 LCSV/HCSV – L-Counter Starting Value/H-Counter Current Value Registers (R24, R25, R26, and R27)

The LCSV and HCSV registers store the start values for the L-Counter and H-Counter, respectively. Each time a counter reaches zero, the start value is automatically reloaded from either LCSV or HCSV, one clock cycle after zero count is reached. Loading LCSV or HCSV from the CPU must be synchronized to avoid writing the registers while the reloading of the counters is occurring. One method is to halt the counters while the registers are loaded.

When the 16-bit counters are concatenated, the LCSV and HCSV registers hold the 32-bit start count, with the least significant byte in R24 and the most significant byte in R27.

3.5.2.6.19 LCCV/HCCV – L-Counter Current Value/H-Counter Current Value Registers (R28, R29, R30, and R31)

The LCCV and HCCV registers hold the current value of the counters. If the CFRZ bit in the MCTL register is reset (0), these registers are updated on each clock cycle with the current value of the counters. LCCV and HCCV can be read only when the counter readings are frozen (CFRZ bit in the MCTL register is 1). They can be written only when the counters are halted (CRUNL and/or CRUNH bits in the CCTL register are 0). This last feature allows new initial count values to be loaded immediately into the counters, and can be used during initialization to avoid long initial counts.

When the 16-bit counters are concatenated, the LCCV and HCCV registers hold the 32-bit current value, with the least significant byte in R28 and the most significant byte in R31.

3.5.2.6.20 Register Initialization

Figure 14 shows a recommended initialization procedure for the ICU that sets up all the ICU registers for proper operation.

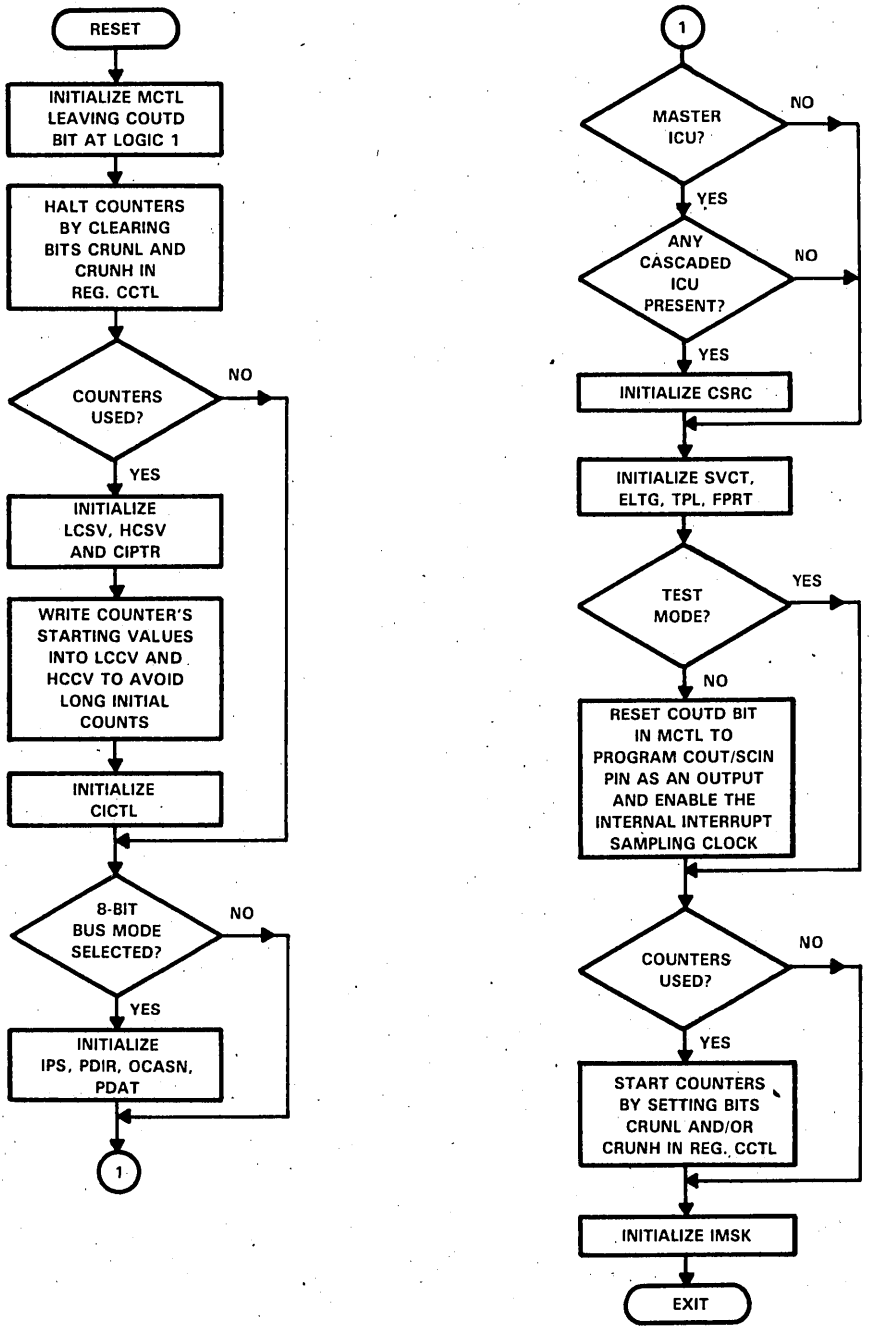


Figure 14. Recommended ICU's Initialization Sequence

3.6 TI32201 Timing Control Unit (TCU)

3.6.1 Power and Grounding

The TI32201 requires a single +5-V power supply, applied to pin 24 (V_{CC}). See the recommended operating characteristics in the TI32201 data sheet. The logic Ground (GND) pin 12 is the common terminal for the TCU.

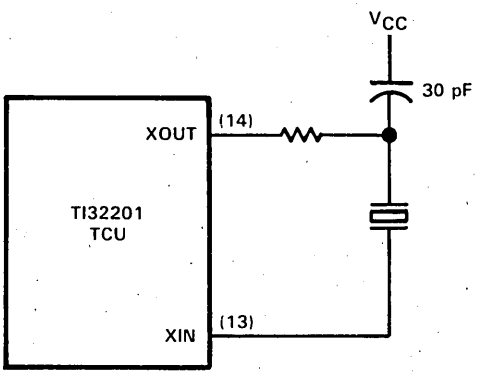
A 0.1-μF ceramic decoupling capacitor must be connected across V_{CC} and GND as close to the TCU as possible.

3.6.2 Crystal Oscillator Characteristics

The TI32201 has a "Pierce"-type oscillator. Connections of the crystal and bias components to XIN and XOUT are shown in Figure 1. It is important that the crystal and the RC components be mounted in close proximity to the XIN, XOUT, and V_{CC} pins to keep printed circuit trace lengths to an absolute minimum.

Typical Crystal Specifications:

Type	At-Cut
Tolerance	0.005% at 25°C
Stability	0.01% from 0°C to 70°C
Resonance	Fundamental (parallel)
Capacitance	20 pF
Maximum Series Resistance	50 Ω



CRYSTAL FREQUENCY (MHz)	R (OHM)
6-12	470
12-18	220
18-24	100
24-30	47

Figure 1. Crystal Connection

3.6.3 Clocks

The TI32201 TCU has four clock output pins. The CLK1 and CLK2 clocks are required by the TI32000 family microprocessor CPUs. These clocks are nonoverlapping as shown in Figure 2.

Each rising edge of CLK1 defines a transition in the timing state of the CPU.

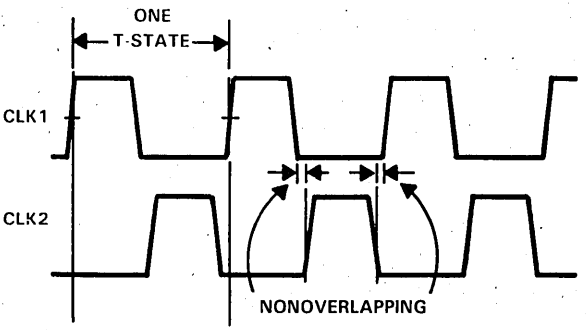


Figure 2. CLK1 and CLK2 Clock Signals

As the TCU generates the various clock signals with very short transition timings, it is recommended that the conductors carrying CLK1 and CLK2 be kept as short as possible. It is also recommended that only the TI32000 CPU and, if used, the MMU (Memory Management Unit) be connected to the CLK1 and CLK2 clocks. In addition to the CPU and MMU, 25-pF ceramic capacitors from these pins to ground are recommended as they provide a better V_{OH} on the outputs. These capacitors should be mounted close to the TCU to minimize trace inductances.

CTTL is a TTL compatible clock signal which runs at the same frequency as CLK1 and is closely balanced with it. CTTL is intended for driving TTL loads.

FCLK is also a TTL compatible clock, running at the frequency of XIN input. This clock is also intended for driving TTL loads and has a frequency that is twice the CTTL clock frequency. The exact phase relationship between CLK1, CLK2, CTTL, and FCLK can be found in the TI32201 data sheet.

3.6.4 Resetting

The TI32201 TCU provides circuitry to meet the reset requirements of the TI32000 family CPUs. If the Reset Input line, $\overline{\text{RSTI}}$ is pulled low, the TCU asserts $\overline{\text{RSTO}}$ which resets the TI32000 CPU. This Reset Output may also be used as a system reset signal. Figure 3 illustrates the reset connections for a non-Memory-Managed system. Figure 4 illustrates the reset connections for a Memory-Managed system.

3.6.5 Synchronizing Two or More TCUs

During reset, (when $\overline{\text{RSTO}}$ is low), one or more TCUs can be synchronized with a reference (Master) TCU. The $\overline{\text{RWEN}}$/SYNC input to the slave TCU(s) is used for synchronization. The Slave TCU samples the $\overline{\text{RWEN}}$/SYNC input on the rising edge

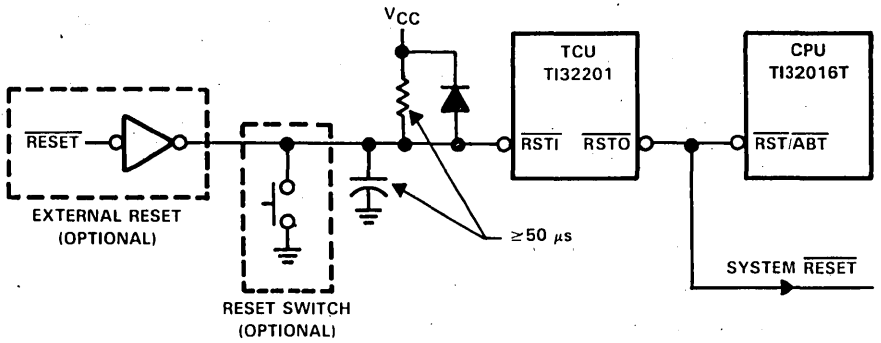


Figure 3. Recommended Reset Connections (Non-Memory Managed System)

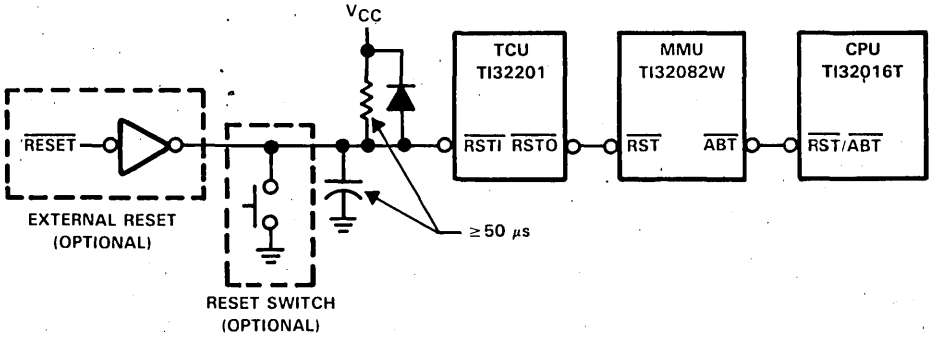


Figure 4. Recommended Reset Connections (Memory-Managed System)

of FCLK when $\overline{RSTO}$ is low and CTTL is high (see Figure 7). If $\overline{RWEN}$/SYNC is sampled high, the phase of CTTL of the slave TCU is shifted by one XIN clock cycle.

Two possible circuits for TCU synchronization are illustrated in Figures 5 and 6. It should be noted that when $\overline{RWEN}$/SYNC is high, the $\overline{RD}$ and $\overline{WR}$ signals will be in the off-state (high-impedance state) at the slave TCU.

In addition to synchronizing two or more TCUs, the $\overline{RWEN}$/SYNC input can be used to "fix" the phase of one TCU to an external pulse. The pulse to be used must be high for only one rising edge of FCLK. Independent of CTTL's state at the FCLK rising edge, its state following the next FCLK rising edge should be low. Figure 8 shows the timing of this sequence.

3.6.6 Bus Cycles

In addition to providing all the necessary clock signals, the TI32201 TCU provides bus control signals to the system. The TCU senses the $\overline{ADS}$ signal from the CPU or MMU to start a bus cycle. The $\overline{DDIN}$ input signal is also sampled to determine whether a Read or Write cycle is to be generated. In addition to $\overline{RD}$ and $\overline{WR}$, other signals are

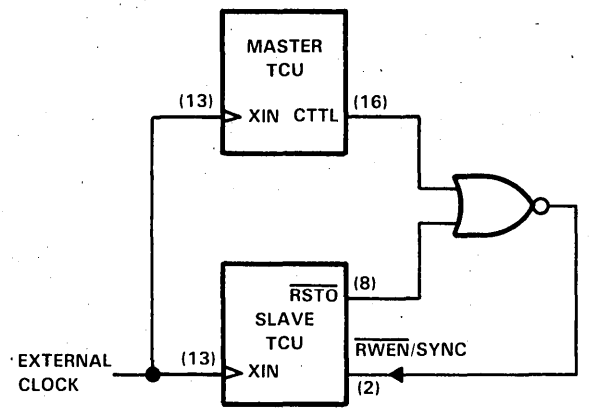
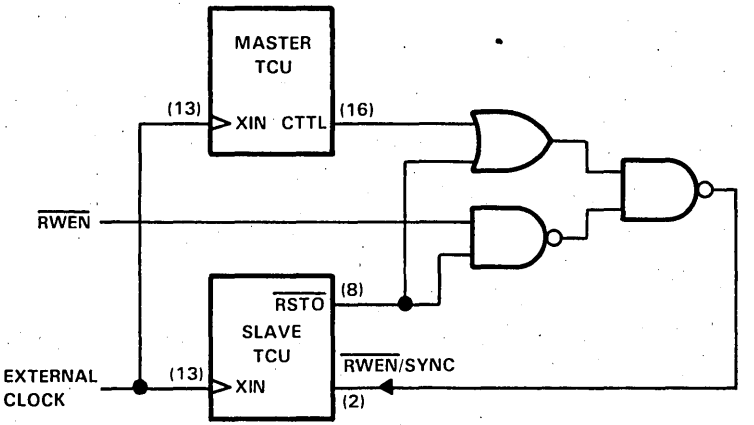


Figure 5. Slave TCU Does Not Use $\overline{\text{RWEN}}$ During Normal Operation



NOTE: When two or more TCUs are to be synchronized, the XIN of all the TCUs should be connected to an external clock source. For details on the external clock, see the TI32201 TCU data sheet.

Figure 6. TCU Uses Both SYNC and $\overline{\text{RWEN}}$

provided: $\overline{\text{DBE}}$ and $\overline{\text{TSO}}$. $\overline{\text{DBE}}$ is used to enable data buffers. The leading edge of $\overline{\text{DBE}}$ is delayed a half clock period during Read cycles to avoid bus conflicts between data buffers and either the CPU or the MMU. This is shown in Figure 9.

The Timing State Output ($\overline{\text{TSO}}$) is a general purpose signal that may be used by external logic for synchronizing to a System cycle. $\overline{\text{TSO}}$ is activated at the beginning of state T2 and returns to the high level at the beginning of state T4 of the CPU cycle. $\overline{\text{TSO}}$ can be used to gate the $\overline{\text{CWAIT}}$ signal when continuous waits are required. Another application of $\overline{\text{TSO}}$ is the control of interface circuitry for dynamic RAMs.

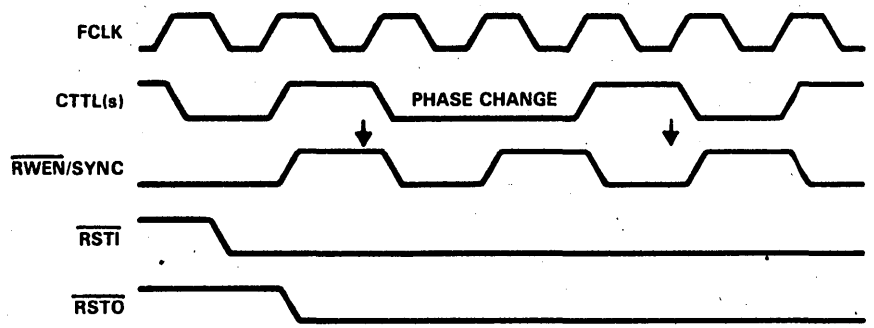


Figure 7. Synchronizing Two TCUs

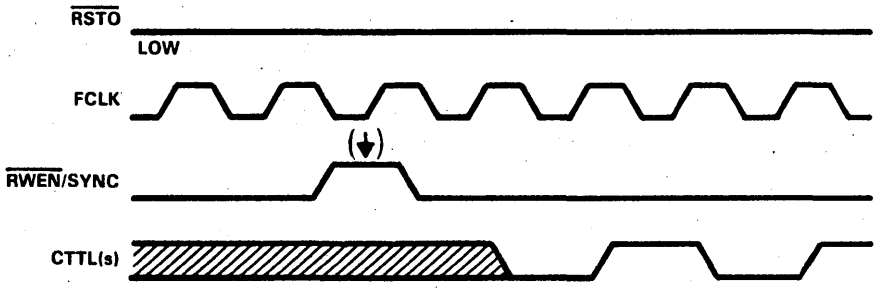


Figure 8. Synchronizing One TCU to an External Pulse

3.6.7 Bus Cycle Extension

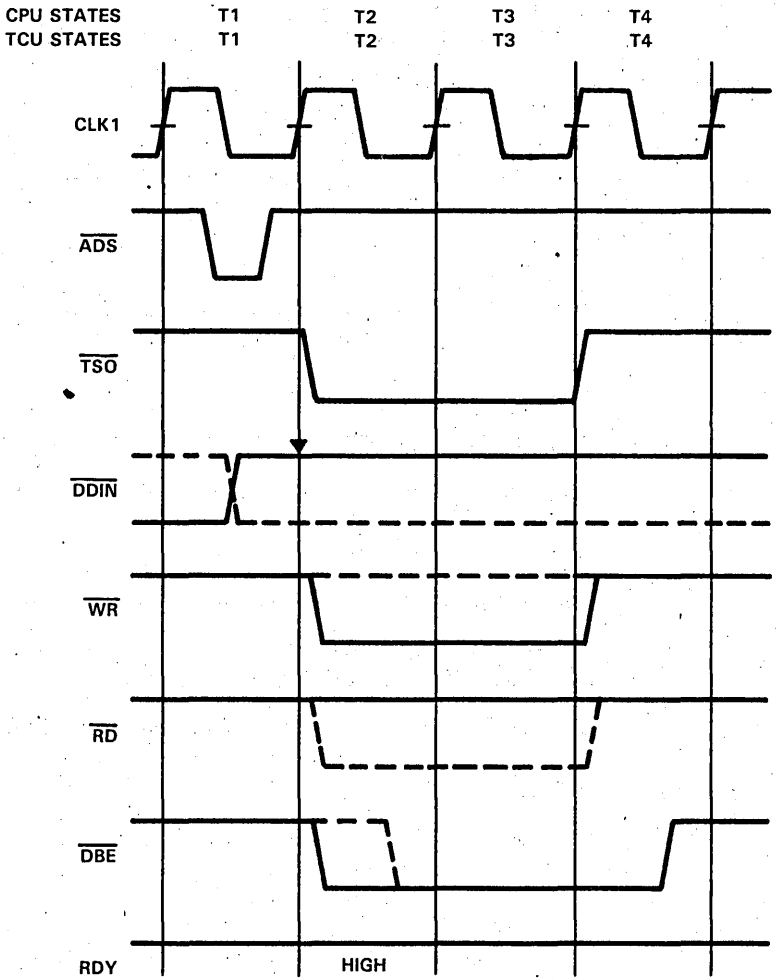
The TI32201 TCU uses the $\overline{\text{WAITn}}$ input signals to extend normal bus cycles. A normal bus cycle consists of four CLK1 clock cycles. Whenever one or more $\overline{\text{WAITn}}$ inputs to the TCU are activated, a bus cycle is extended by at least one CLK1 clock cycle. The purpose is to allow the CPU to access slow memories or peripherals. The TCU responds to the Wait signals by pulling the RDY signal low as long as Wait States are to be inserted in the bus cycle.

There are three basic cycle extension modes provided by the TCU, as described in the following.

3.6.7.1 Normal Wait-States

This is a normal Wait-State insertion mode. It is initiated by pulling $\overline{\text{CWAIT}}$ or any of the $\overline{\text{WAITn}}$ lines low in the middle of T2. Figure 10 shows the timing diagam of a bus cycle when $\overline{\text{CWAIT}}$ is sampled high at the end of T1 and low in the middle of T2.

The RDY signal goes low during T2 and remains low until $\overline{\text{CWAIT}}$ is sampled high by the TCU. RDY is pulled high by the TCU during the same CLK1 cycle in which the $\overline{\text{CWAIT}}$ line is sampled high.



NOTES: 1. The CPU and TCU view some timing states (T-states) differently. For clarity, references to T-states will sometimes be followed by (TCU) or (CPU). (CPU) also implies (MMU).
2. Arrows indicate when the TCU samples the input.
3. $\overline{RWEN}$ is assumed low ($\overline{RD}$ and $\overline{WR}$ enabled) unless specified differently.
4. For clarity, T-states for both the TCU and CPU are shown above the diagrams. (See Note 1).

Figure 9. Basic TCU Cycle (Fast Cycle)

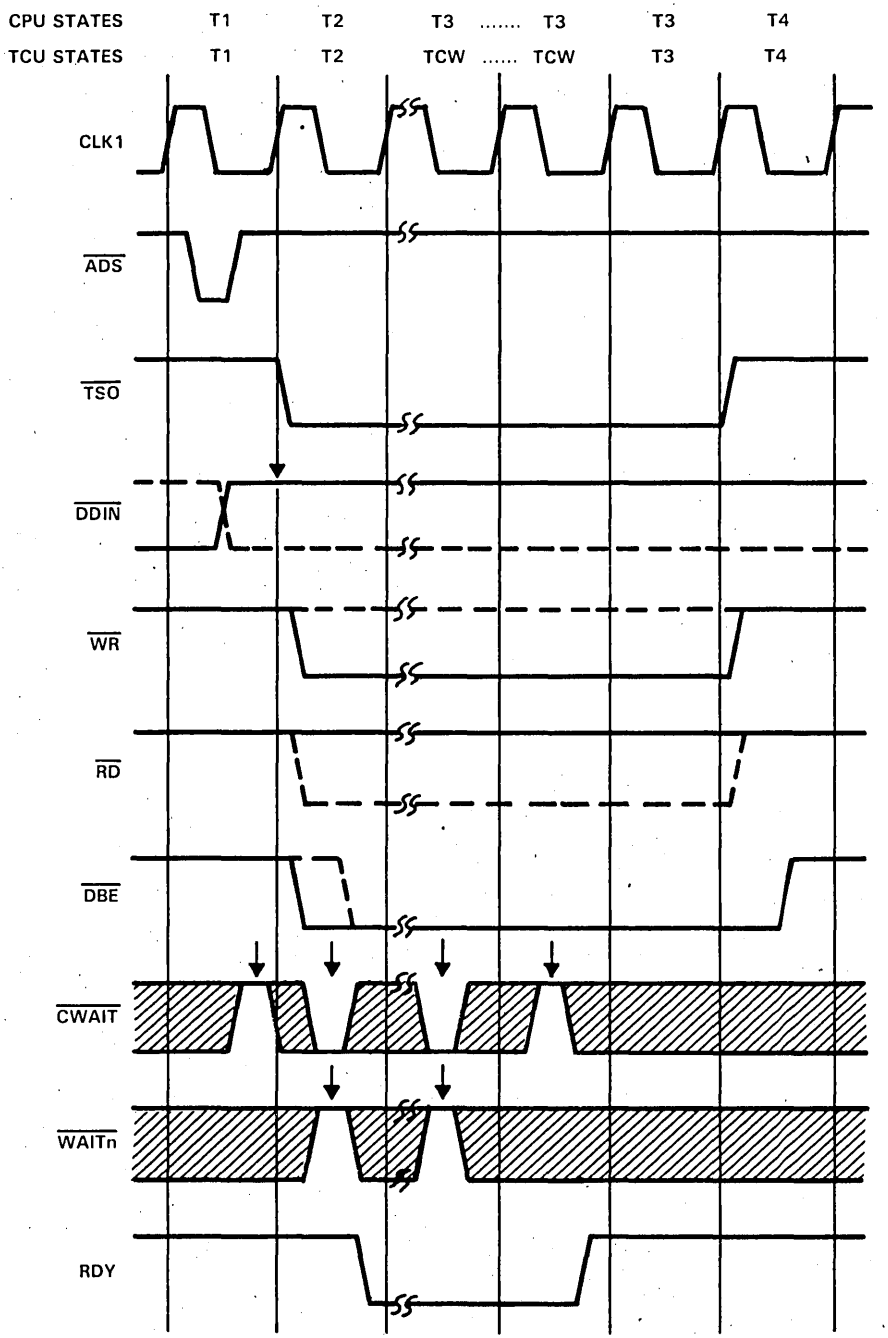


Figure 10. Wait-State Insertion Using $\overline{\text{CWAIT}}$ (Fast Cycle)

If any of the $\overline{\text{WAITn}}$ signals are sampled low during T2 and $\overline{\text{CWAIT}}$ is high during the entire bus cycle, then the RDY Line goes low for 1 to 15 clock cycles, depending on the binary weighted value of $\overline{\text{WAITn}}$. If, for example, $\overline{\text{WAIT1}}$ and $\overline{\text{WAIT4}}$ are sampled low, then five Wait-States will be inserted. This is shown in Figure 11.

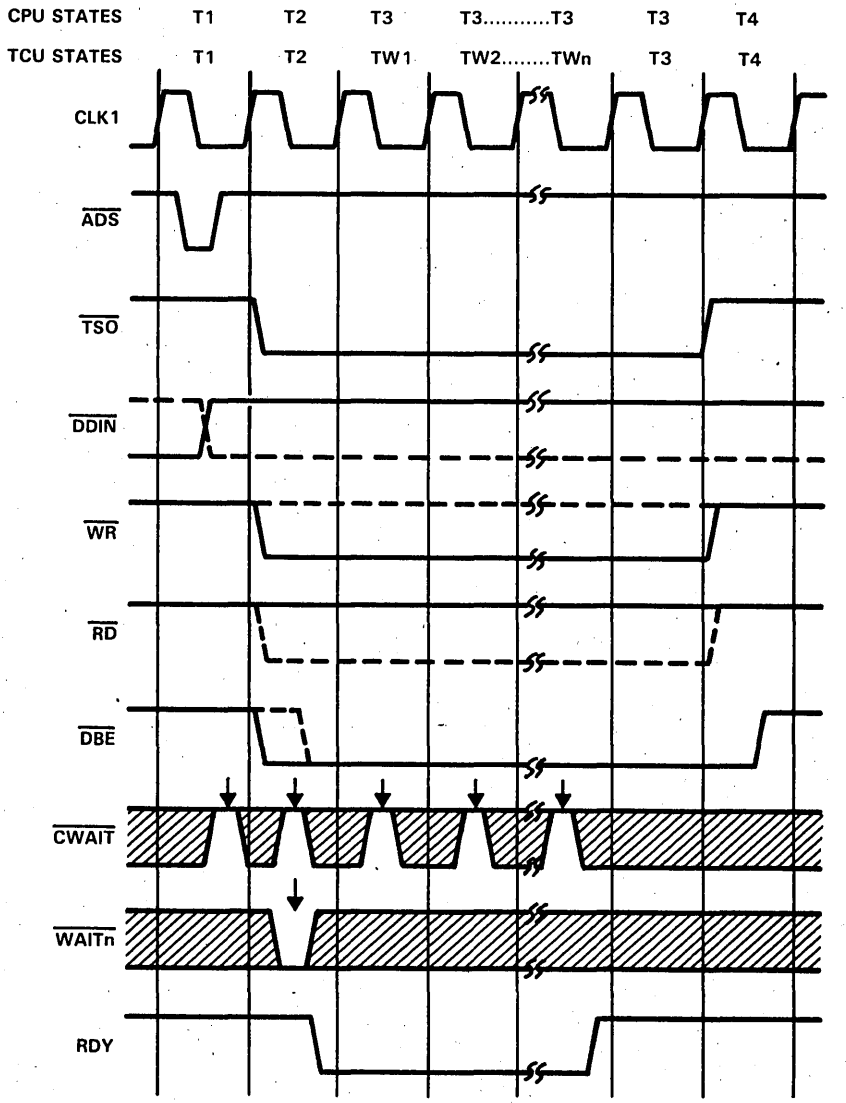


Figure 11. Wait-State Insertion Using $\overline{\text{WAITn}}$ (Fast Cycle)

3.6.7.2 Peripheral Cycle.

This cycle is entered when the $\overline{PER}$ signal line is sampled low at the beginning of T2. The TCU adds five Wait-States identified as TD0-TD4 into a normal bus cycle. The $\overline{RD}$ and $\overline{WR}$ signals are also reshaped so the setup and hold times for address and data will be increased. The use of the $\overline{PER}$ signal may be necessary when interfacing with slower peripherals. Figure 12 shows the timing diagram of a peripheral cycle.

3.6.7.3 Cycle Hold.

If the $\overline{CWAIT}$ input is sampled low at the end of state T1, the TCU will go into cycle hold mode and stay in this mode for as long as $\overline{CWAIT}$ is kept low. During this mode the control signals $\overline{RD}$, $\overline{WR}$, $\overline{TSO}$, and $\overline{DBE}$ are kept inactive; RDY is pulled low, thus causing Wait-States to be inserted into the bus cycle. The cycle hold feature can be used in applications involving dynamic RAMs. A timing diagram showing the cycle hold feature is shown in Figure 13.

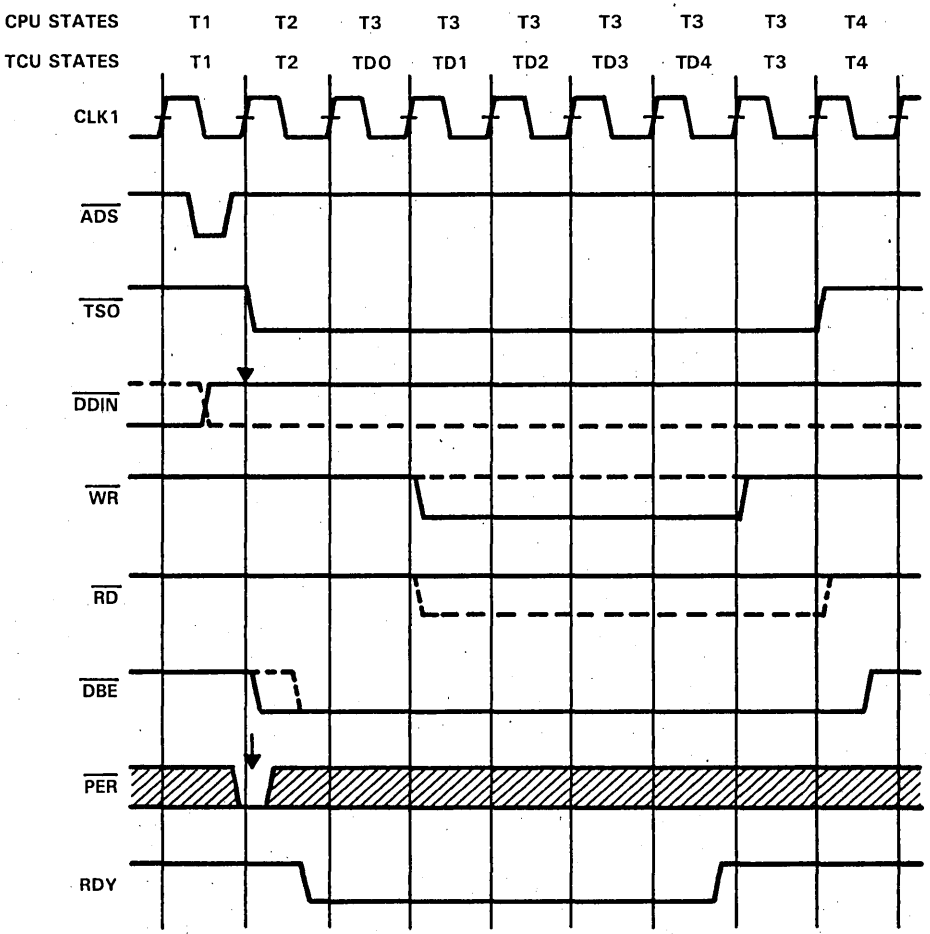


Figure 12. Peripheral Cycle

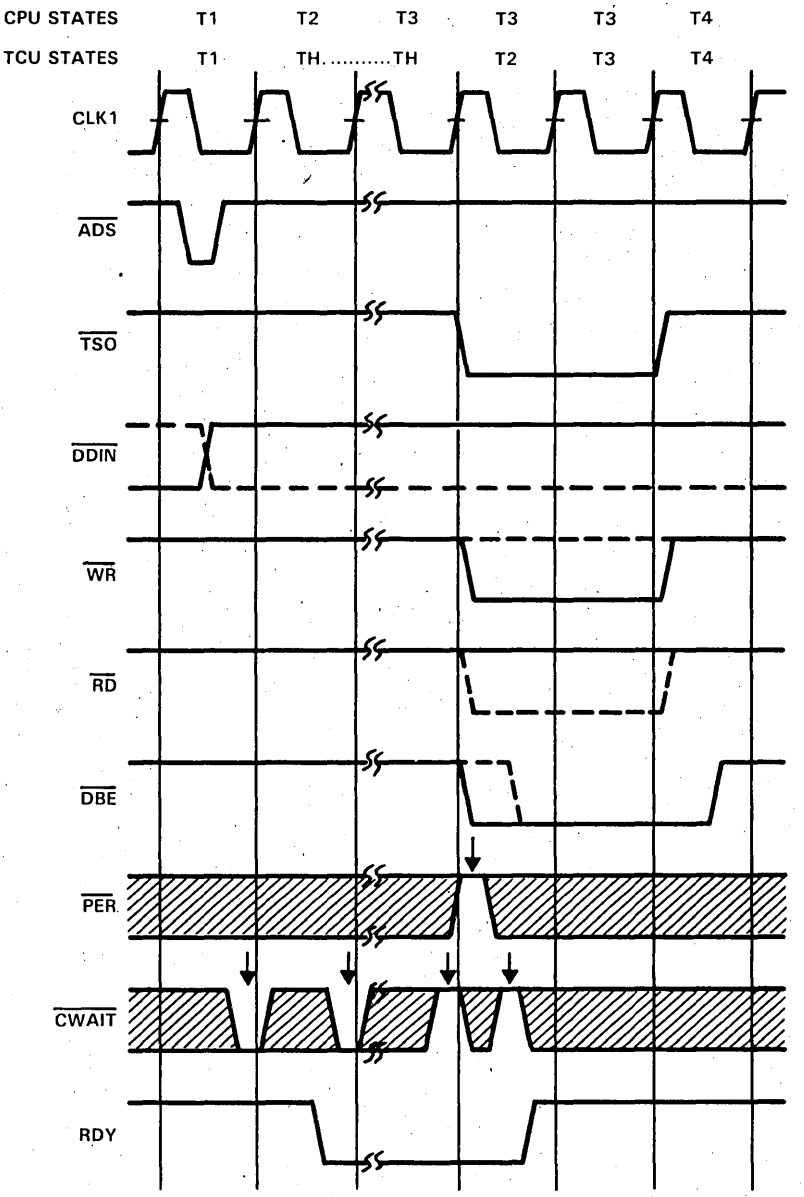


Figure 13. Cycle Hold Timing Diagram

3.6.8 Bus Cycle Extension Combinations

Any combination of the TCU input signals used for extending a bus cycle can be activated at one time. The TCU will honor all of the requests according to a certain priority scheme. A cycle hold request is assigned top priority. It follows a peripheral cycle request, and then $\overline{\text{CWAIT}}$ and $\overline{\text{WAITn}}$ respectively.

If, for example, all the input signals $\overline{\text{CWAIT}}$, $\overline{\text{PER}}$, and $\overline{\text{WAITn}}$ are asserted at the beginning of the cycle, the TCU will enter the cycle hold mode. As soon as $\overline{\text{CWAIT}}$ goes high, the input signal $\overline{\text{PER}}$ is sampled to determine whether a peripheral cycle is requested.

Next, the TCU samples $\overline{\text{CWAIT}}$ again and $\overline{\text{WAITn}}$ to check whether additional Wait-States have to be inserted into the bus cycle. This sampling point depends on whether $\overline{\text{PER}}$ was sampled high or low. If $\overline{\text{PER}}$ was sampled high, then the sampling point will be in the middle of the TCU state T2, (Figure 16), otherwise it will occur three clock cycles later (Figure 17). Figures 14 through 17 show the timing diagrams for different combinations of cycle extensions.

3.6.9 Overriding $\overline{\text{WAITn}}$ Wait-States

The TCU handles the $\overline{\text{WAITn}}$ Wait-States by means of an internal counter that is reloaded with the binary value corresponding to the state of the $\overline{\text{WAITn}}$ inputs each time $\overline{\text{CWAIT}}$ is sampled low, and is decremented when $\overline{\text{CWAIT}}$ is high.

This allows one to either extend a bus cycle by a predefined number of clock cycles, or prematurely terminate it. To terminate a bus cycle, for example, $\overline{\text{CWAIT}}$ must be asserted for at least one clock cycle, and the $\overline{\text{WAITn}}$ inputs must be forced to their inactive state.

At least one Wait-State is always inserted when using this procedure as a result of $\overline{\text{CWAIT}}$ being sampled low. Figure 18 shows the timing diagram of a prematurely terminated bus cycle where eleven Wait-States were inserted.

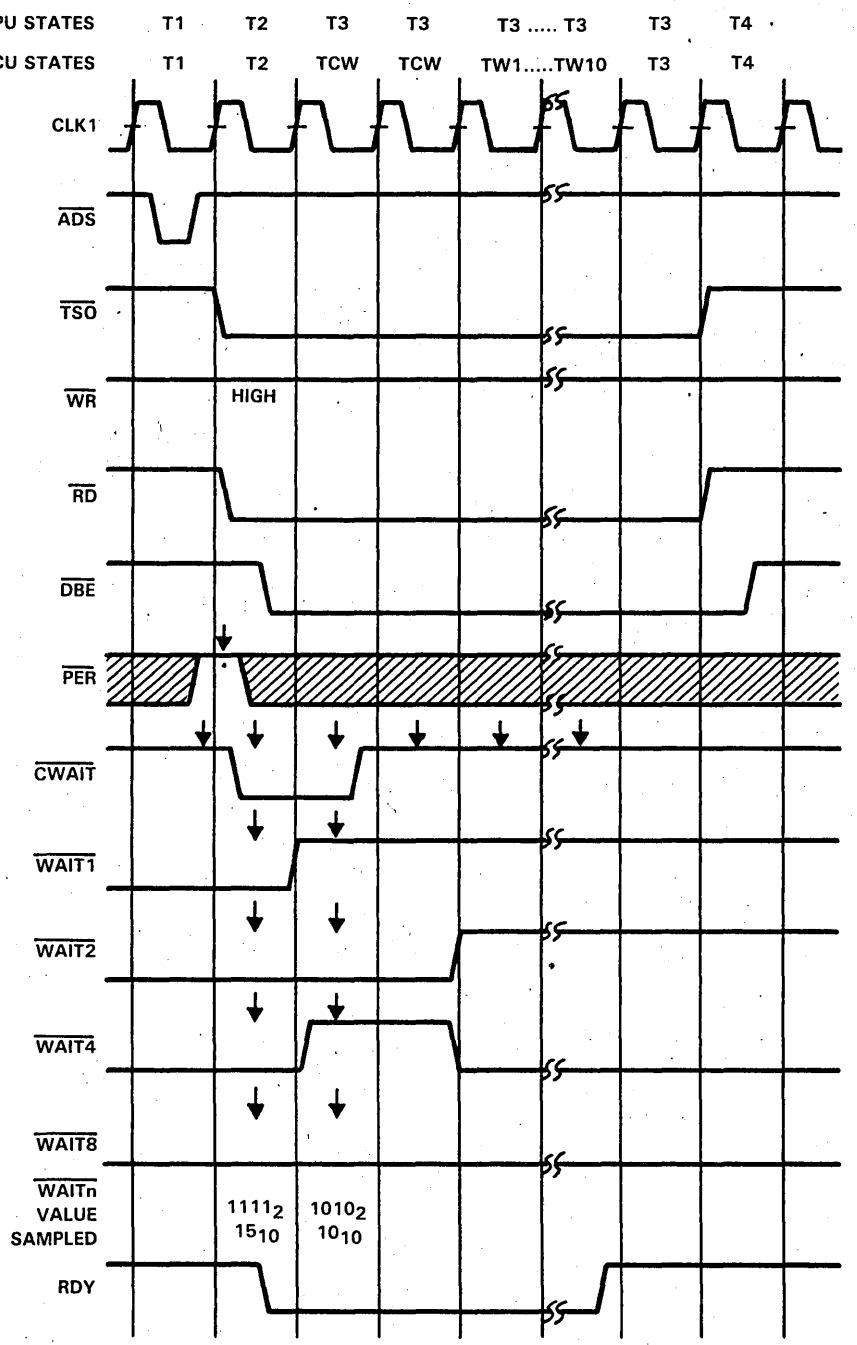


Figure 14. Fast Cycle with 12 Wait-States (2 $\overline{\text{CWAIT}}$ and $\overline{\text{WAIT10}}$) (Read Cycle)

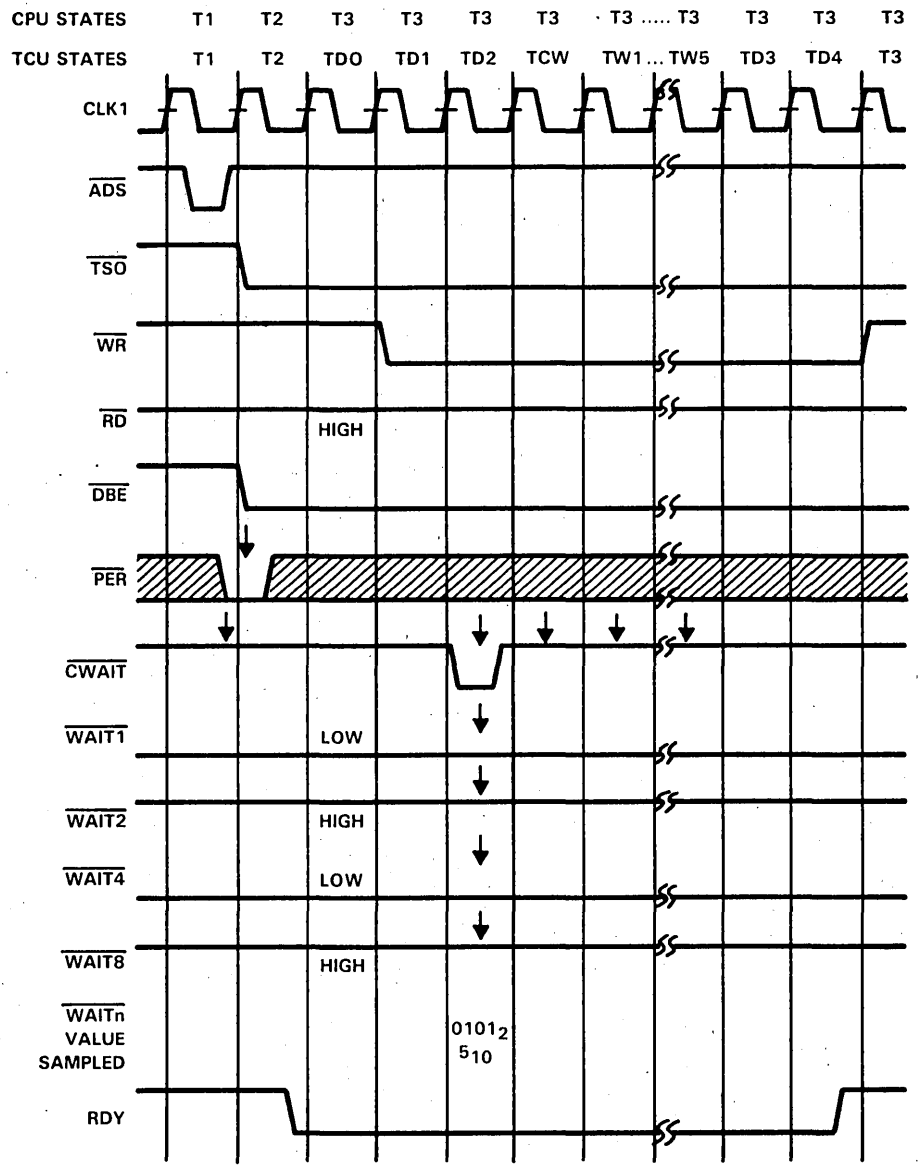


Figure 15. Peripheral Cycle with Six Wait-States (1 $\overline{\text{CWAIT}}$ and $\overline{\text{WAIT5}}$) (Write Cycle)

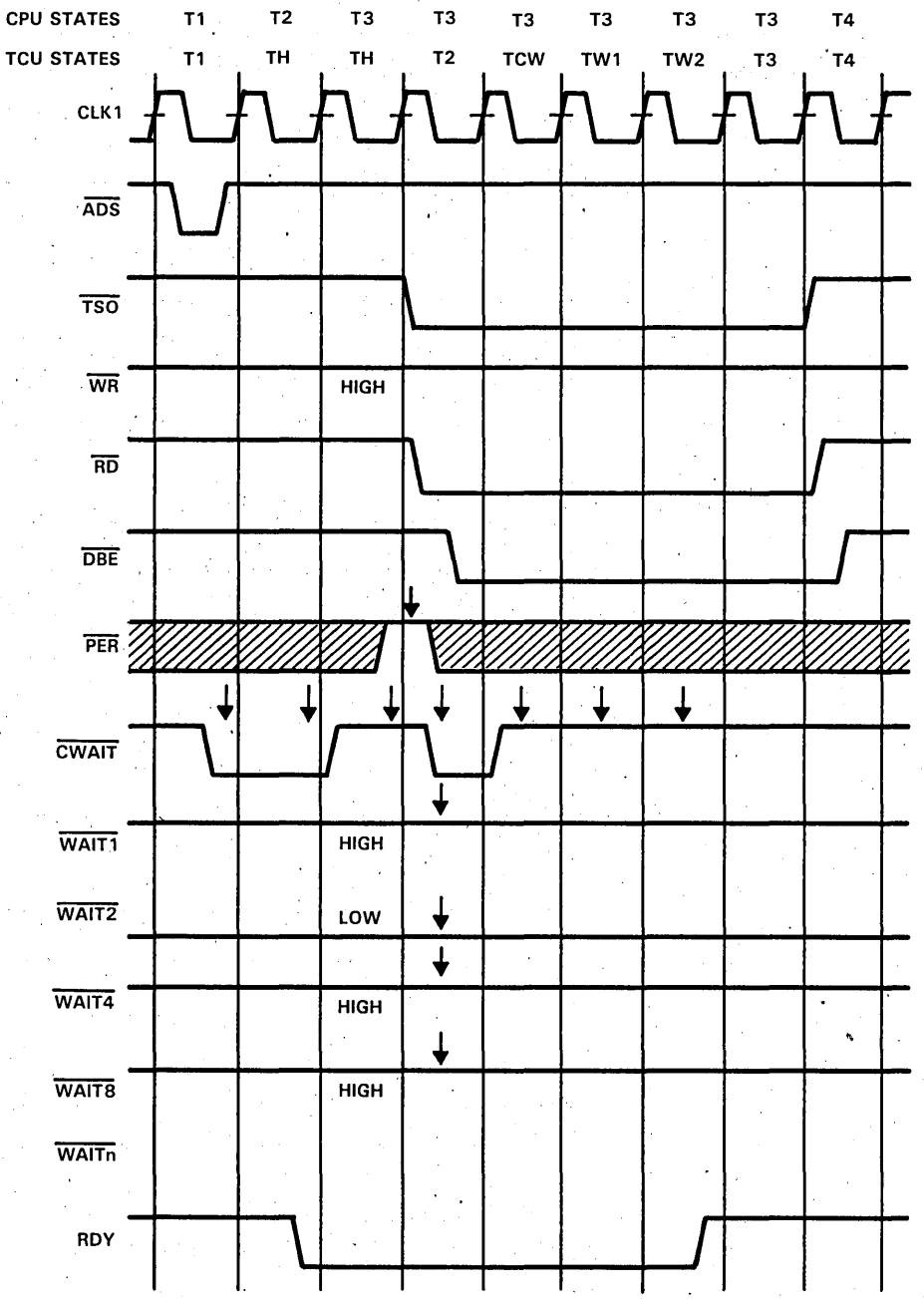


Figure 16. Cycle Hold with Three Wait-States (1 $\overline{\text{CWAIT}}$ and $\overline{\text{WAIT2}}$) (Read Cycle)

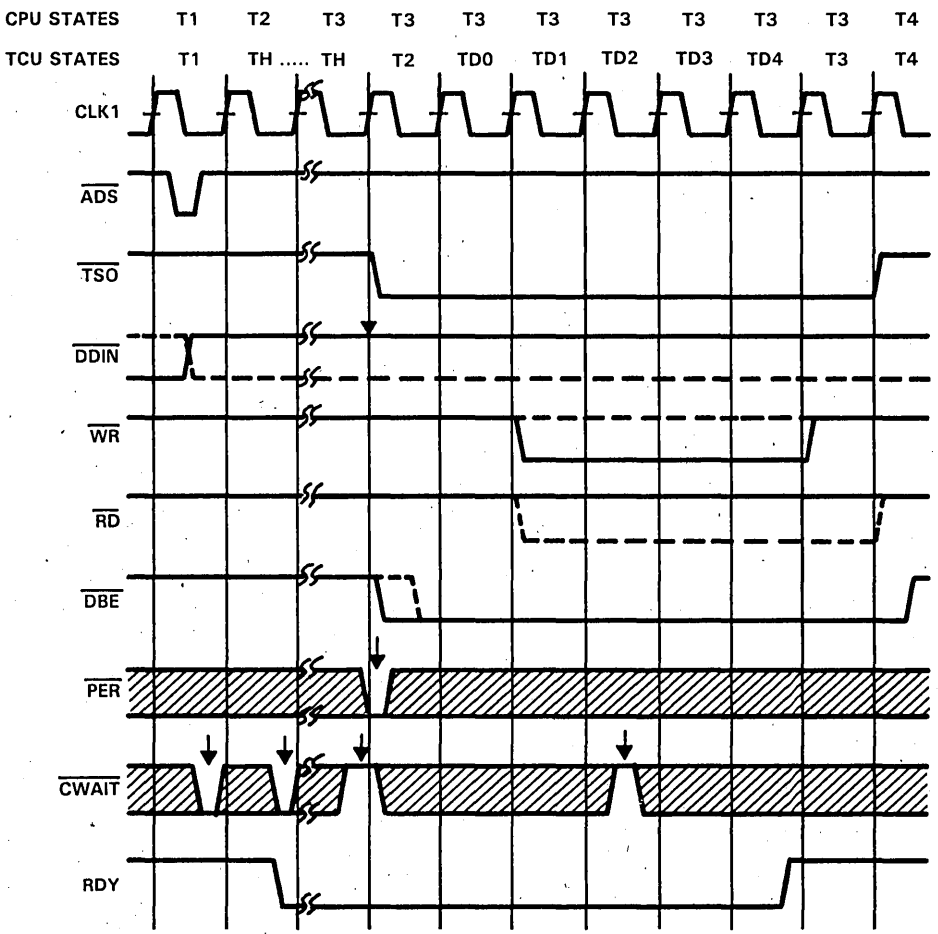


Figure 17. Cycle Hold of a Peripheral Cycle

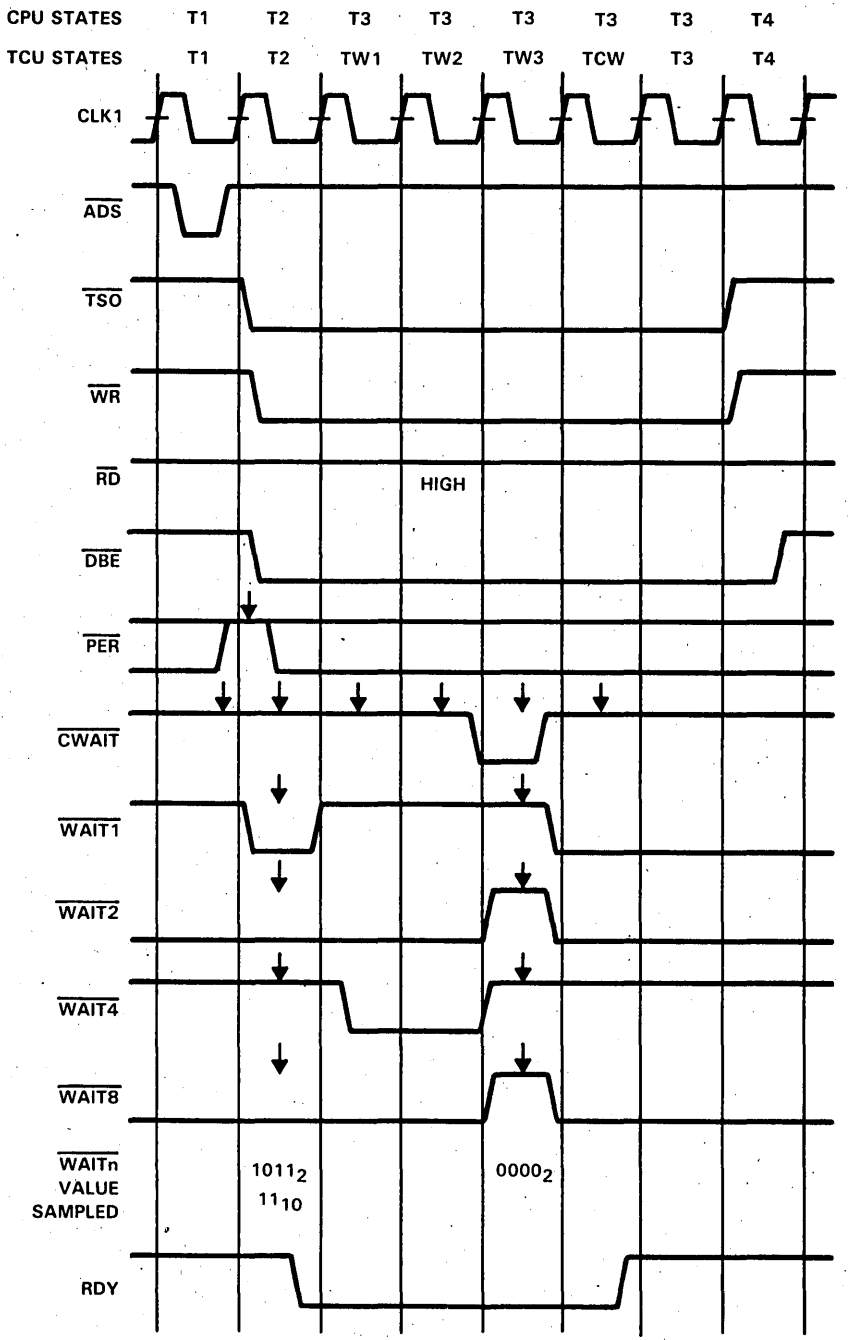


Figure 18. Overriding $\overline{\text{WAITn}}$ Wait-States (Write Cycle)

D2875, APRIL 1985

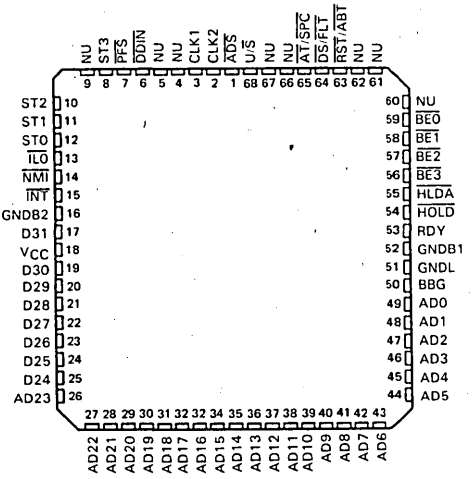
- High-Speed NMOS Technology
- 32-Bit Architecture and Implementation with 24-Bit Address
- 16-Megabyte Uniform Addressing Space
- Powerful Instruction Set with:
  - General 2-Address Capability
  - Very High Degree of Symmetry
  - Addressing Modes Optimized for High-Level Language References
- TI32000 Coprocessor Support
- Single 5-V Operation
- Direct Replacement for National Semiconductor NS32032-10

description

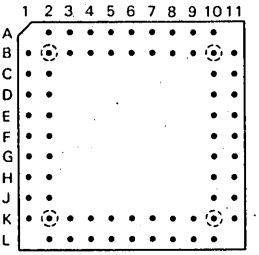
The TI32032T-2 is a monolithic high-performance high-speed NMOS microprocessor designed to function as a central processing unit (CPU) in the TI32000 microprocessor family. The device has been designed primarily to support microprocessor users who require the ability to use a large addressing space for large programs and/or large data structures. Because large programs must realistically be generated and maintained in high-level languages, the TI32000 family architecture provides for very efficient compilation while remaining easy to program at the assembler level. The TI32000 family architecture in conjunction with the TI32082 Memory Management Unit (MMU) provides full support for demand-paged virtual memory management. High-performance floating-point instructions are provided by the TI32081 Floating Point Unit (FPU).

The TI32032T is characterized for operation from 0°C to 70°C.

FK OR FN . . . CHIP-CARRIER PACKAGE
(TOP VIEW)



PIN GRID ARRAY PACKAGE
(TOP VIEW)

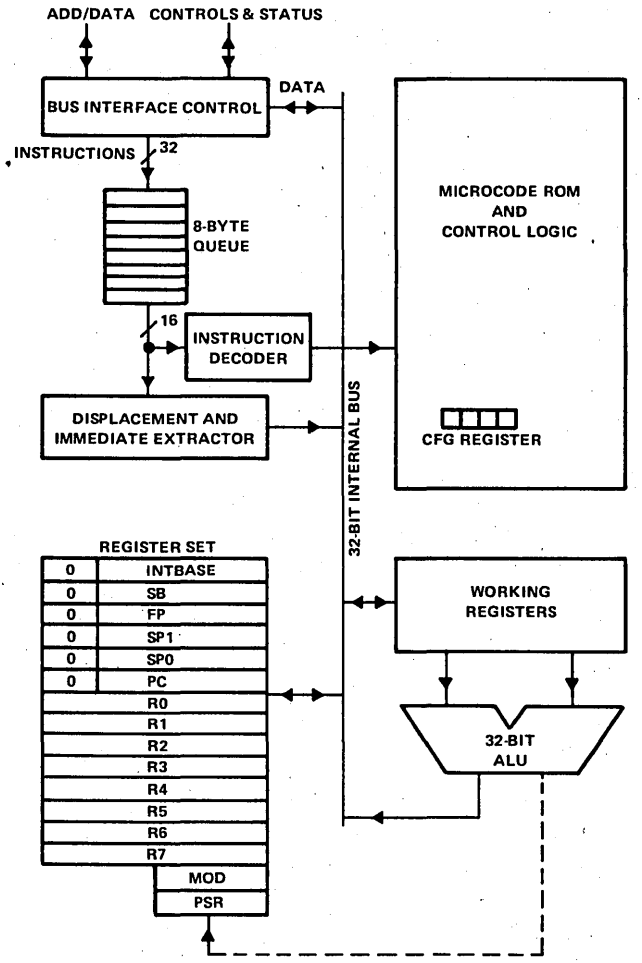


PIN	NAME	PIN	NAME
A2	NU	F10	RDY
A3	$\overline{PFS}$	F11	GNDB1
A4	NU	G1	D29
A5	CLK1	G2	D28
A6	$\overline{ADS}$	G10	GNDL
A7	NU	G11	BBG
A8	$\overline{AT/SPC}$	H1	D27
A9	$\overline{RST/ABT}$	H2	D26
A10	NU	H10	AD0
B1	ST2	H11	AD1
B2	ST1	J1	D25
B3	ST3	J2	D24
B4	$\overline{DDIN}$	J10	AD2
B5	NU	J11	AD3
B6	CLK2	K1	AD23
B7	U/$\overline{S}$	K2	AD21
B8	NU	K3	AD19
B9	$\overline{DS/FLT}$	K4	AD17
B10	NU	K5	AD15
B11	NU	K6	AD13
C1	ST0	K7	AD11
C2	$\overline{ILO}$	K8	AD9
C10	$\overline{BE0}$	K9	AD7
C11	$\overline{BE1}$	K10	AD4
D1	$\overline{NMI}$	K11	AD5
D2	$\overline{INT}$	L2	AD22
D10	$\overline{BE2}$	L3	AD20
D11	$\overline{BE3}$	L4	AD18
E1	GNDB2	L5	AD16
E2	D31	L6	AD14
E10	$\overline{HLDA}$	L7	AD12
E11	$\overline{HOLD}$	L8	AD10
F1	VCC	L9	AD8
F2	D30	L10	AD6

NU—Reserved for future use. Make no external connection.

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

functional block diagram



TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

PIN FUNCTIONAL DESCRIPTION

PIN NO.		NAME	I/O	DESCRIPTION
PIN GRID ARRAY	CHIP CARRIER			
A6	1	$\overline{ADS}$	O	Address strobe output. Available to the system for address latch control. Indicates start of a bus cycle by going low
B6	2	CLK2	I	Clock 2 input. One phase of a two-phase clock input
A5	3	CLK1	I	Clock 1 input. One phase of a two-phase clock input
B5	4	NU		Reserved for future use. Make no external connection.
A4	5	NU		Reserved for future use. Make no external connection.
B4	6	$\overline{DDIN}$	O	Data Direction In output. Status signal that indicates the direction of data transfer during a bus cycle, low for read, high for write
A3	7	$\overline{PFS}$	O	Program Flow Status output. Low-going pulse indicates beginning of an instruction execution.
B3	8	ST3	O	Status bit 3 output. Bus cycle status code
A2	9	NU		Reserved for future use. Make no external connection.
B1	10	ST2	O	Status bit 2 output. Bus cycle status code
B2	11	ST1	O	Status bit 1 output. Bus cycle status code
C1	12	ST0	O	Status bit 0 (least significant bit) output. Bus cycle status code
C2	13	$\overline{ILO}$	O	Interlocked Operation Output. When low, indicates that an interlocked instruction is being executed
D1	14	$\overline{NMI}$	I	Nonmaskable Interrupt input. Nonmaskable interrupt request
D2	15	$\overline{INT}$	I	Interrupt input. Maskable interrupt request
E1	16	GNDB2		Buffer Ground 2. Ground reference for half of the on-chip drivers connected to output pins
E2	17	D31	I/O	Data bit 31 of the data bus
F1	18	V_{CC}		+5 V supply voltage
F2	19	D30	I/O	Data bit 30 of the data bus
G1	20	D29	I/O	Data bit 29 of the data bus
G2	21	D28	I/O	Data bit 28 of the data bus
H1	22	D27	I/O	Data bit 27 of the data bus
H2	23	D26	I/O	Data bit 26 of the data bus
J1	24	D25	I/O	Data bit 25 of the data bus
J2	25	D24	I/O	Data bit 24 of the data bus
K1	26	AD23	I/O	Address bit 23. Multiplexed address/data information
L2	27	AD22	I/O	Address bit 22. Multiplexed address/data information
K2	28	AD21	I/O	Address bit 21. Multiplexed address/data information
L3	29	AD20	I/O	Address bit 20. Multiplexed address/data information
K3	30	AD19	I/O	Address bit 19. Multiplexed address/data information
L4	31	AD18	I/O	Address bit 18. Multiplexed address/data information
K4	32	AD17	I/O	Address bit 17. Multiplexed address/data information
L5	33	AD16	I/O	Address bit 16. Multiplexed address/data information
K5	34	AD15	I/O	Address bit 15. Multiplexed address/data information
L6	35	AD14	I/O	Address bit 14. Multiplexed address/data information
K6	36	AD13	I/O	Address bit 13. Multiplexed address/data information
L7	37	AD12	I/O	Address bit 12. Multiplexed address/data information
K7	38	AD11	I/O	Address bit 11. Multiplexed address/data information
L8	39	AD10	I/O	Address bit 10. Multiplexed address/data information
K8	40	AD9	I/O	Address bit 9. Multiplexed address/data information
L9	41	AD8	I/O	Address bit 8. Multiplexed address/data information
K9	42	AD7	I/O	Address bit 7. Multiplexed address/data information
L10	43	AD6	I/O	Address bit 6. Multiplexed address/data information
K11	44	AD5	I/O	Address bit 5. Multiplexed address/data information

PIN FUNCTIONAL DESCRIPTION

PIN NO.		NAME	T/O	DESCRIPTION
PIN GRID ARRAY	CHIP CARRIER			
K10	45	AD4	I/O	Address bit 4. Multiplexed address/data information
J11	46	AD3	I/O	Address bit 3. Multiplexed address/data information
J10	47	AD2	I/O	Address bit 2. Multiplexed address/data information
H11	48	AD1	I/O	Address bit 1. Multiplexed address/data information
H10	49	AD0	I/O	Address bit 0 (least significant bit). Multiplexed address/data information
G12	50	BBG	O	Back-Bias Generator supply. Output of on-chip substrate voltage generator
G10	51	GNDL		Logic ground. Ground reference for on-chip logic.
F11	52	GNDB1		Buffer Ground 1. Ground reference for half of the on-chip drivers connectd to output pins.
F10	53	RDY	I	READY input. When high, causes the CPU to terminate the bus cycle. when low, causes the CPU to extend the current bus cycle to provide for a slower memory or peripheral reference.
E11	54	$\overline{\text{HOLD}}$	I	Hold request input. When low, causes the CPU to release the bus for DMA or multiprocessing purposes.
E10	55	$\overline{\text{HLDA}}$	O	Hold acknowledge output. Applied by the CPU in response to a $\overline{\text{HOLD}}$ input. Low level indicates that the bus has been released for DMA or multiprocessing purposes.
D11	56	$\overline{\text{BE3}}$	O	Byte Enable 3 output. When low, (with $\overline{\text{BE0}}$ through $\overline{\text{BE2}}$) enables data transfers on individual bus bytes.
D10	57	$\overline{\text{BE2}}$	O	Byte Enable 2 output. When low, (when $\overline{\text{BE0}}$, $\overline{\text{BE1}}$ and $\overline{\text{BE3}}$) enables data transfers on individual bus bytes.
C11	58	$\overline{\text{BE1}}$	O	Byte Enable 1 output. When low, (with $\overline{\text{BE0}}$, $\overline{\text{BE2}}$ and $\overline{\text{BE3}}$) enables data transfers on individual bus bytes.
C10	59	$\overline{\text{BE0}}$	O	Byte Enable 0 output. When low, (with $\overline{\text{BE1}}$ through $\overline{\text{BE3}}$) enables data transfers on individual bus bytes.
B11	60	NU	I	Reserved for future use. Make no external connection.
A10	61	NU		Reserved for future use. Make no external connection.
B10	62	NU		Reserved for future use. Make no external connection.
A9	63	$\overline{\text{RST}}/\overline{\text{ABT}}$	I	Reset/Abort input. If held low for one clock cycle and released, causes an abort command. If held low for longer than one clock cycle, initiates a reset.
B9	64	$\overline{\text{DS}}/\overline{\text{FLT}}$	I/O	Data Strobe/Float. Data strobe output or float command input. Function is selected by $\overline{\text{AT}}/\overline{\text{SPC}}$ input.
A8	65	$\overline{\text{AT}}/\overline{\text{SPC}}$	I/O	Address Translation/Coprocessor Control. Used by the CPU as the data strobe output for coprocessor transfers. Used by coprocessors to acknowledge completion of an instruction. Sampled on the trailing edge of reset pulse as address translation strap.
B8	66	NU		Reserved for future use. Make no external connection.
A7	67	NU		Reserved for future use. Make no external connection.
B7	68	U/$\overline{\text{S}}$	O	User/Supervisor status output. When high, indicates user mode. When low, indicates supervisor mode.

absolute maximum ratings over recommended operating conditions (unless otherwise noted)

Supply voltage, V_{CC} (see Note 1) . . . . . 7 V
Input voltage range . . . . . −5 V to 7 V
Output voltage range . . . . . −5 V to 7 V
Continuous total dissipation at (or below) 70°C free-air temperature . . . . . 1.5 W
Operating free-air temperature range, T_A . . . . . 0°C to 70°C

NOTE 1: All voltage values are with respect to the common ground.

recommended operating conditions

		MIN	NOM	MAX	UNIT
Supply voltage, V_{CC}		4.75	5	5.25	V
High-level input voltage, V_{IH}	CLK1, CLK2	$V_{CC}-0.5$		$V_{CC}+0.5$	V
	All other inputs	2		$V_{CC}+0.5$	
Low-level input voltage, V_{IL}	CLK1, CLK2	−0.5		0.3	V
	All other inputs	−0.5		0.8	
Operating free-air temperature, T_A		0		70	°C

electrical characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

PARAMETER			TEST CONDITION	MIN	TYP†	MAX	UNIT
V_{OH}	High-level output voltage		I_{OH} = −400 μA	2.4			V
V_{OL}	Low-level output voltage		I_{OL} = 2 mA			0.45	V
I_I	Input current	$\overline{AT}/SPC$	V_I = 0.4 V	50		1000	μA
		All other inputs except CLK1, CLK2, and $\overline{AT}/SPC$	V_I = 0 to V_{CC}			±20	
$I_{O(off)}$	Off-state output current		V_O = 0.4 V to V_{CC}			±20	μA
I_{CC}	Supply current		I_O = 0, T_A = 25°C		180	300	mA

†All typical values are at V_{CC} = 5 V, T_A = 25°C.

Timing parameter symbols have been created in accordance with JEDEC Standard 100. In order to shorten the symbols, the pin names have been further abbreviated as follows:

AD	A	FCLK	F
CLK1	C or C1	$\overline{HLDA}$	HA
CLK2	C2	$\overline{HOLD}$	HD
$\overline{DDIN}$	DD	U/$\overline{S}$	US

Lowercase subscripts and their meaning are:

c	cycle time (period)
dis	disable time (H or L to Z)
en	enable time (Z to H or L)
f	fall time
p	propagation delay time
pwr	supply voltage stable
r	rise time
su	setup time
w	pulse duration

Parentheses have been omitted except when required for separation of a final subscript representing special conditions. The columns titled NSC SYMBOL show the symbols used by National Semiconductor for the parameters shown in the timing requirements and switching characteristics tables.

timing requirements over recommended operating free-air temperature range (unless otherwise noted)

	PARAMETER	FIGURE	NSC SYMBOL	MIN	MAX	UNIT
$t_{suD(RD)}$	Setup time, read data before CLK2↓	2,11	t_{DIs}	10		ns
$t_{hD(RD)}$	Hold time, read data after CLK1↑	2,11	t_{DIh}	10		ns
t_{suHDL}	Setup time, $\overline{HOLD}$ Low before CLK2↓ (see Note 3)	3, 4	t_{HLDa}	25		ns
t_{hHD}	Hold time, $\overline{HOLD}$ low or high after CLK1↑	3,4,5	t_{HLDh}	0		ns
t_{suHDH}	Setup time, $\overline{HOLD}$ high before CLK2↓	5	t_{HLDia}	25		ns
t_{suFLTL}	Setup time, $\overline{FLT}$ low before CLK2↓	6	t_{FLTa}	25		ns
t_{suFLTH}	Setup time, $\overline{FLT}$ high before CLK2↓	7	t_{FLTia}	25		ns
t_{suRDY}	Setup time, RDY low or high before CLK2↓	8, 9	t_{RDYs}	15		ns
t_{hRDY}	Hold time, RDY low or high after CLK1↓	8, 9	t_{RDYh}	0		ns
t_{wSPC}	Pulse duration, $\overline{SPC}$ low at 0.8 V (both edges)	10	t_{SPCw}	20		ns
t_{suAT}	Setup time, $\overline{AT}/\overline{SPC}$ low before CLK1↑	13	t_{ATs}	$1t_{cC}$		
t_{hAT}	Hold time, $\overline{AT}/\overline{SPC}$ low after CLK1↓	13	t_{ATh}	$2t_{cC}$		
t_{rC}	Rise time, CLK1 or CLK2	14	t_{CLr}		7	ns
t_{fC}	Fall time, CLK1 or CLK2	14	t_{CLf}		7	ns
t_{cC}	Clock period	14	t_{Cp}	100	5000	ns
t_{wC1H}	Pulse duration, CLK1 high	14	t_{CLh}	$0.5t_{cC}-10$		ns
t_{wC2H}	Pulse duration, CLK2 high	14	t_{CLl}	$0.5t_{cC}-10$		ns
	Clock pulse asymmetry, t_{wC1H}-t_{wC2H}		t_{CLwas}		±5	ns
t_{C1LC2H}	Nonoverlap time at 10% of CLK1↓ and CLK2↑	14	$t_{nOVL(1)}$	0	7	ns
t_{C1LC2L}	Nonoverlap time at 10% of CLK2↓ and CLK1↑	14	$t_{nOVL(2)}$	0	7	ns
	Nonoverlap asymmetry, $t_{C1LC2H}-t_{C12LC1H}$		t_{nOVLas}		±4	ns
t_{suABT}	Setup time, $\overline{ABT}$ low before CLK2↓	22,23	t_{ABTs}	20		ns
t_{hABT}	Hold time, $\overline{ABT}$ Low after CLK1↑	22,23	t_{ABTh}	0		ns
t_{pwr}	Supply voltage stable (above 4.5 V) to $\overline{RST}$ high	24	t_{PWR}	50		μs
t_{suRST}	Setup time, $\overline{RST}$ high before CLK1↓	24, 25	t_{RSTs}	20		ns
t_{wRST}	Puls duration, $\overline{RST}$ low at 0.8 V (both edges)	25	t_{RSTw}	$64t_{cC}$		
t_{suINT}	Setup time, $\overline{INT}$ low before CLK1↓	26	t_{INTs}	20		ns
t_{wNMI}	Pulse duration, $\overline{NMI}$ low at 0.8 V (both edges)	27	t_{NMIw}	70		ns
t_{LX-PFS}	Last operand transfer of an instruction to next $\overline{PFS}$ clock cycle	28	t_{LXPF}	$0t_{cC}$		

NOTE 3: This setup time is necessary to ensure prompt acknowledgement via $\overline{HLDA}$ and the ensuing floating of the CPU off the buses. The time from the receipt of the $\overline{HOLD}$ signal until the CPU floats is a function of the time $\overline{HOLD}$ signal goes low, the state of the RDY input (in MMU systems), and the length of the current MMU cycle.

switching characteristics over recommended ranges of supply voltage and operating free-air temperature, C_L = 100 pF (unless otherwise noted)

PARAMETER		FIGURE	NSC SYMBOL	MIN	MAX	UNIT
t_{aA}	Access time, AD0 through AD23 after CLK1↑	1	t_{ALv}		50	ns
t_{vA}	Valid time, AD0 through AD23 after CLK1↑	1	t_{ALh}	10		ns
t_{aD}	Access time, AD0 through AD23 and D24 through D31 after CLK1↑	1	t_{Dv}		50	ns
t_{vD}	Valid time, AD0 through AD23 and D24 through D31 after CLK1↑	1	t_{Dh}	0		ns
t_{aBE}	Access time, $\overline{BE0}$ through $\overline{BE3}$ after CLK2↑	1	t_{BEv}		45	ns
t_{vBE}	Valid time, $\overline{BE0}$ through $\overline{BE3}$ after CLK2↑	1	t_{BEh}	0		ns
t_{aST}	Access time, ST0 through ST3 after CLK1↑	1	t_{STv}		45	ns
t_{vST}	Valid time, ST0 through ST3 after CLK1↑	1	t_{STh}	0		ns
t_{pADSL}	Propagation delay time, CLK1↑ to $\overline{ADS}$ low	1	t_{ADSa}		35	ns
t_{pADSH}	Propagation delay time, CLK2↑ to $\overline{ADS}$ high	1	t_{ADSia}	15	45	ns
t_{wADS}	Pulse duration, $\overline{ADS}$ low at 0.8 V (both edges)	1	t_{ADSw}	35		ns
t_{pDSL}	Propagation delay time, CLK1↑ to $\overline{DS}$ low	1	t_{DSa}		45	ns
t_{pDSH}	Propagation delay time, CLK1↑ to $\overline{DS}$ high	1	t_{DSia}	†	40	ns
$t_{AV\text{-}ADSH}$	Time interval, AD0 through AD23 valid to $\overline{ADS}$ high	2	t_{ALADSs}	25		ns
t_{disA}	Disable time, AD0 through AD23 after CLK1↑ (without MMU)	2	t_{ALf}		25	ns
t_{disD}	Disable time, D24 through D31 after CLK1↑ (without MMU)	2	t_{ADf}		25	ns
t_{aDD}	Access time, $\overline{DDIN}$ after CLK1↑	2	t_{DDINv}		65	ns
t_{vDD}	Valid time, $\overline{DDIN}$ after CLK1↑	2	t_{DDINh}	0		ns
t_{disA}	Disable time, AD0 through AD23 after CLK1↑ ($\overline{HOLD}$ low)	3	t_{ALf}		25	ns
t_{disD}	Disable time, D24 through D31 after CLK1↑ ($\overline{HOLD}$ low)	3	t_{ADf}		25	ns
t_{disADS}	Disable time, ADS after CLK1↑ ($\overline{HOLD}$ low)	3,4	t_{ADSf}		55	ns
t_{disBE}	Disable time, $\overline{BE0}$ through $\overline{BE3}$ after CLK1↑ ($\overline{HOLD}$ low)	3,4	t_{BEf}		55	ns
t_{disDD}	Disable time, $\overline{DDIN}$ after CLK1↑ ($\overline{HOLD}$ low)	3,4	t_{DDINf}		55	ns
t_{pHAL}	Propagation delay time, CLK1↑ to $\overline{HLDA}$ low	3,4	t_{HLDAa}		75	ns
t_{pHAH}	Propagation delay time, CLK1↑ to $\overline{HLDA}$ high	5	t_{HLDAia}		75	ns
t_{enADS}	Enable time, $\overline{ADS}$ after CLK1↑ ($\overline{HOLD}$ high)	5	t_{ADSr}		55	ns
t_{enBE}	Enable time, $\overline{BE0}$ through $\overline{BE3}$ after CLK1↑($\overline{HOLD}$ high)	5	t_{BEr}		55	ns
t_{enDD}	Enable time, $\overline{DDIN}$ after CLK1↑ ($\overline{HOLD}$ high)	5	t_{DDINr}		55	ns
$t_{vADSH\text{-}A}$	Valid time, AD0 through AD23 after $\overline{ADS}$↑	6	t_{ALADSh}	10		ns
$t_{disA(1)}$	Disable time, AD0 through AD23 after CLK1↑ (with MMU)	6	t_{ALMf}		25	ns
t_{disD}	Disable time, D24 through D31 after CLK1↑ (with MMU)	6	t_{ADMf}		25	ns
$t_{disA(2)}$	Disable time, AD0 through AD23 after CLK1↑ ($\overline{FLT}$ Low)	6	t_{ALf}		30	ns
t_{disDD}	Disable time, $\overline{DDIN}$ after $\overline{FLT}$↑	6	t_{DDINf}		55	ns
t_{enDD}	Enable time, $\overline{DDIN}$ after $\overline{FLT}$↑	7	t_{DDINr}		50	ns
t_{pSPCL}	Propagation delay time, CLK1↑ to $\overline{SPC}$ low	10	t_{SPCa}		35	ns
t_{pSPCH}	Propagation delay time, CLK1↑ to $\overline{SPC}$ high	10	t_{SPCia}		35	ns
t_{aD}	Access time, write data (bits AD0 through AD15) after CLK1↑	10	t_{Dv}		50	ns
t_{vD}	Valid time, write data (bits AD0 through AD15) after CLK1↑	10	t_{Dh}	0		ns

switching characteristics (continued)

PARAMETER		FIGURE	NSC SYMBOL	MIN	MAX	UNIT
t_{disSPC}	Disable time, $\overline{SPC}$ output nonforcing after CLK2↑	12	t_{SPCnf}		10	ns
t_{wPFS}	Pulse duration, $\overline{PFS}$ low at 0.8 V (both edges)	15	t_{PFSw}	70		ns
t_{pPFSL}	Propagation delay time, CLK2↑ to $\overline{PFS}$ low	15	t_{PFSa}		50	ns
t_{pPFSH}	Propagation delay time, CLK2↑ to $\overline{PFS}$ high	15	t_{PFSia}		50	ns
t_{PFS-NS}	Time interval, $\overline{PFS}$ clock cycle to next nonsequential fetch	16	t_{PFNS}	$4t_{cC}$		
t_{NS-PFS}	Time interval, nonsequential fetch to next $\overline{PFS}$ clock cycle	17	t_{NSPF}	$4t_{cC}$		
$t_{ILOL-C1}$	Time interval, $\overline{ILO}$ low to CLK1↑	18	t_{ILOs}	30		ns
t_{vILO}	Valid time, $\overline{ILO}$ low after CLK1↑	19	t_{ILOh}	10		ns
t_{pILOL}	Propagation delay time, CLK1↑ to $\overline{ILO}$ low	20	t_{ILOa}		70	ns
t_{pILOH}	Propagation delay time, CLK1↑ to $\overline{ILO}$ high	20	t_{ILOia}		70	ns
t_{aUS}	Access time, U/$\overline{S}$ after CLK1↑	21	t_{USv}		70	ns
t_{vUS}	Valid time, U/$\overline{S}$ after CLK1↑	21	t_{USh}	10		ns

PARAMETER MEASUREMENT INFORMATION†

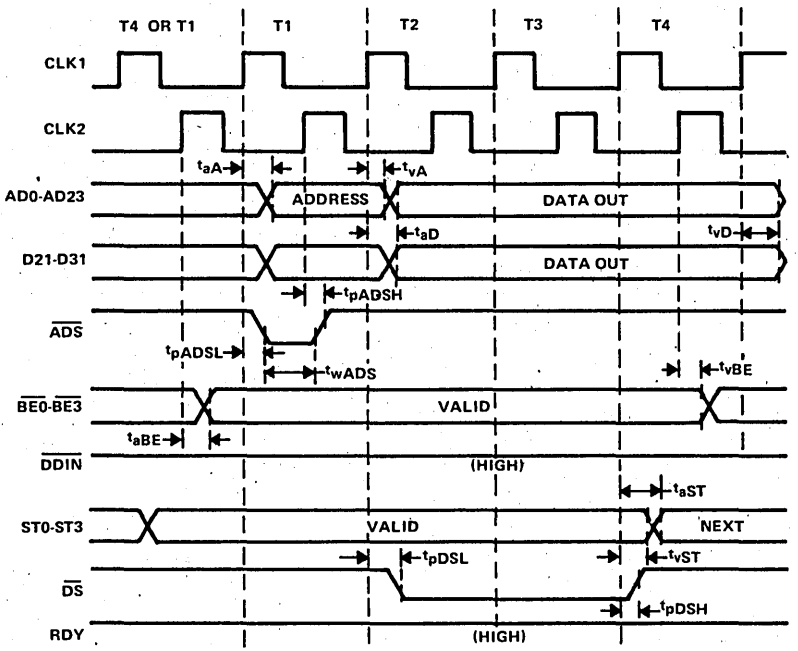


FIGURE 1. WRITE CYCLE

†In Figures 1 through 28, time intervals are defined with respect to the following reference points:
For clock signals, the 50% points.
For all other signals, 2 V if the high level is indicated and 0.8 V if the low level is indicated.

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

PARAMETER MEASUREMENT INFORMATION

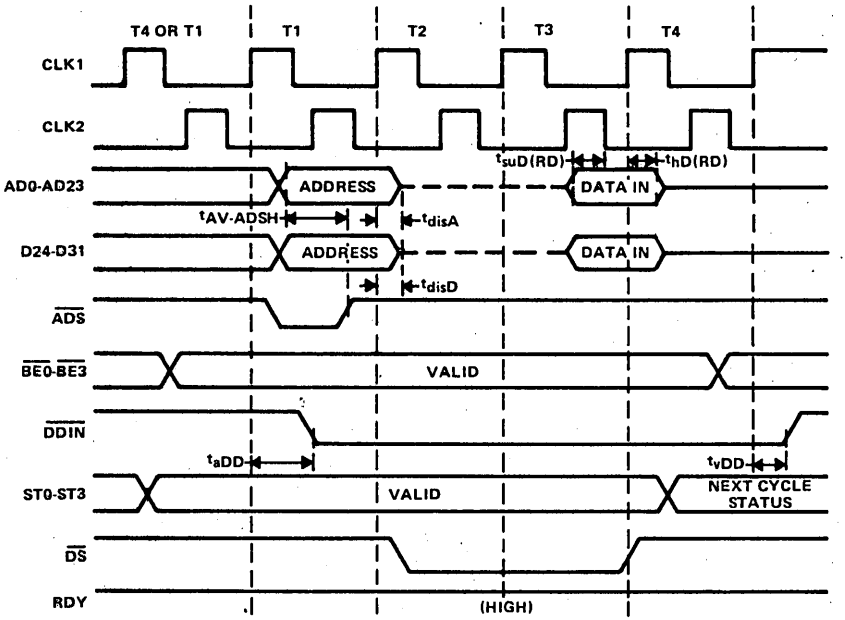
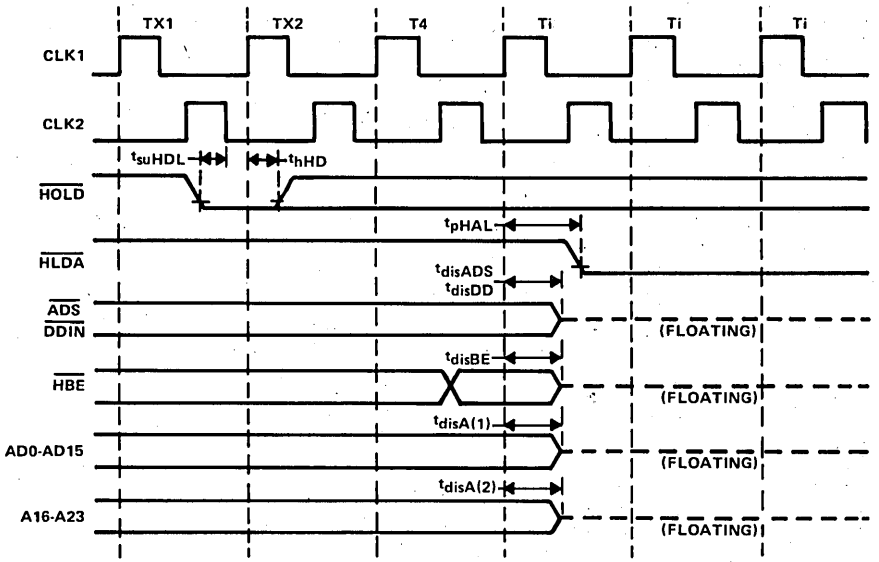


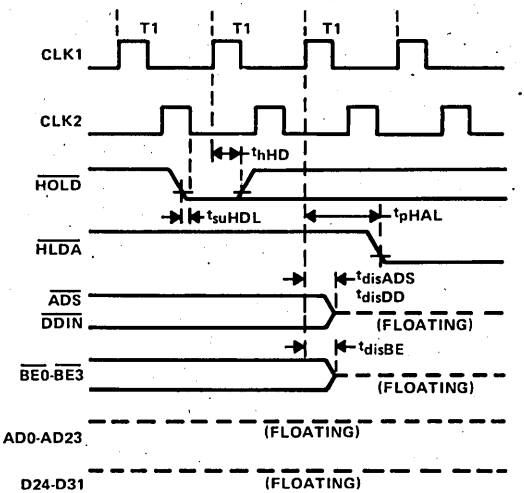
FIGURE 2. READ CYCLE

PARAMETER MEASUREMENT INFORMATION



NOTE: When the CPU is not idling (not in Ti), the $\overline{HOLD}$ request ($\overline{HOLD}$ low) must be active before the trailing edge of CLK2 that appears two clock cycles before T4 (TX1) and stay low until t_{hHDL} after the leading edge of CLK1 that precedes T4 (TX2) for the request to acknowledge.

FIGURE 3. FLOATING BY $\overline{HOLD}$ TIMING (CPU NOT IDLE INITIALLY)



NOTE: During T1 the CPU is already idling.

FIGURE 4. FLOATING BY $\overline{HOLD}$ TIMING (CPU INITIALLY IDLE)

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

PARAMETER MEASUREMENT INFORMATION

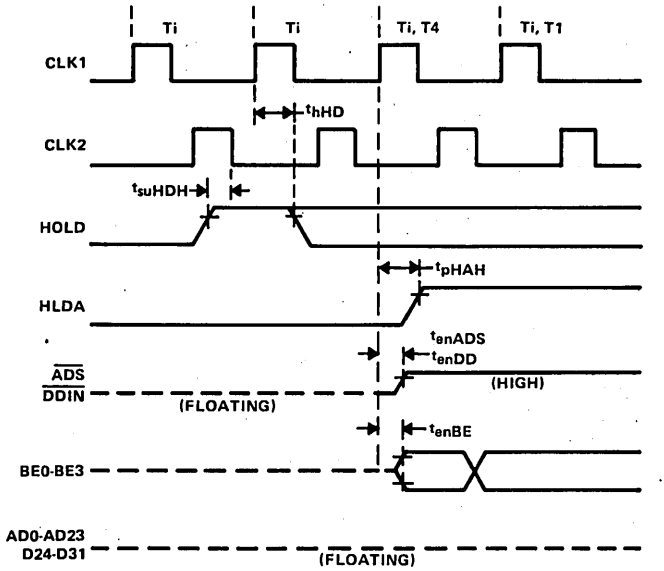


FIGURE 5. RELEASE FROM $\overline{HOLD}$

PARAMETER MEASUREMENT INFORMATION

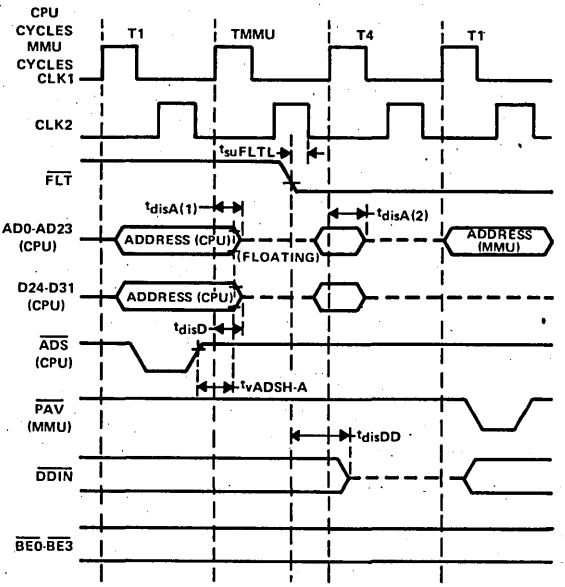


FIGURE 6. $\overline{FLT}$ INITIATED FLOAT CYCLE TIMING

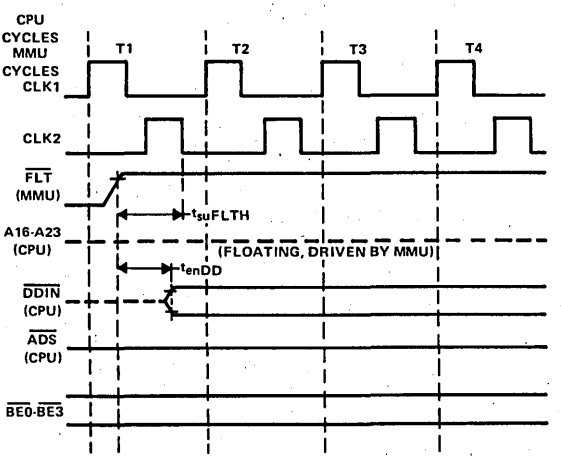


FIGURE 7. RELEASE FROM $\overline{FLT}$ TIMING

NOTE: When $\overline{FLT}$ goes high, the CPU restarts driving $\overline{DDIN}$ before the MMU releases it. This does not cause any conflict since both CPU and MMU force $\overline{DDIN}$ to the same logic level.

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

PARAMETER MEASUREMENT INFORMATION

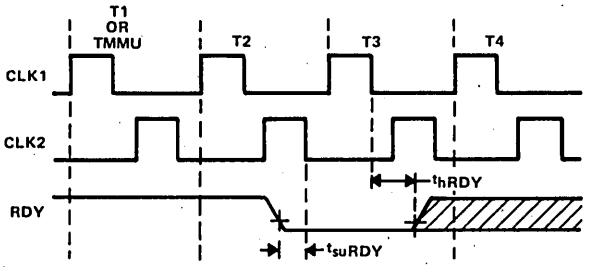


FIGURE 8. READY SAMPLING (CPU INITIALLY READY)

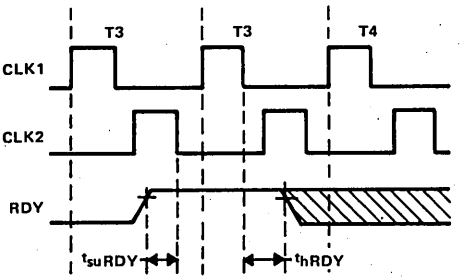


FIGURE 9. READY SAMPLING (CPU INITIALLY NOT READY)

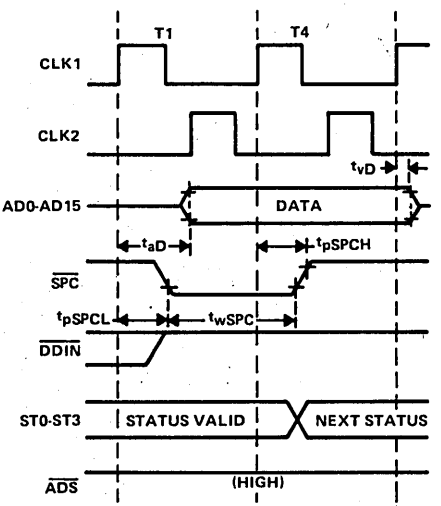


FIGURE 10. COPROCESSOR WRITE TIMING

PARAMETER MEASUREMENT INFORMATION

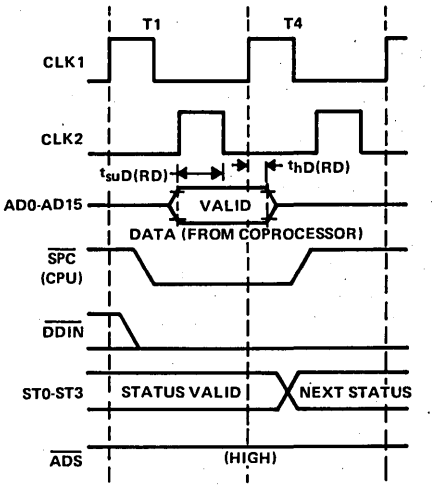


FIGURE 11. COPROCESSOR READ TIMING

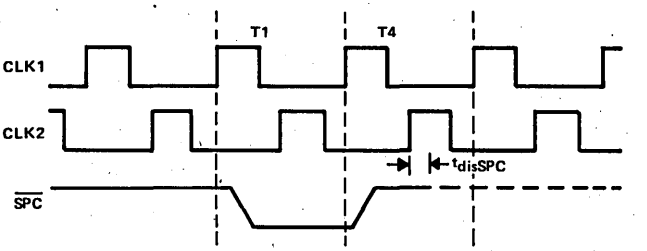


FIGURE 12. $\overline{SPC}$ NONFORCING DELAY

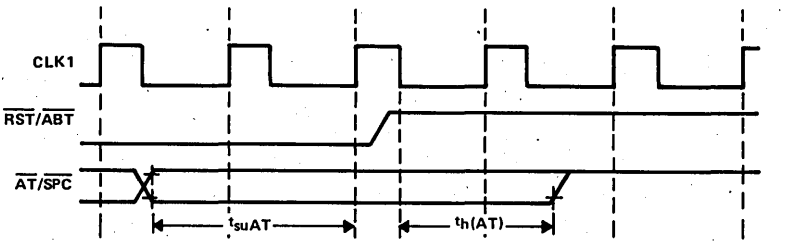


FIGURE 13. RESET CONFIGURATION TIMING

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

PARAMETER MEASUREMENT INFORMATION

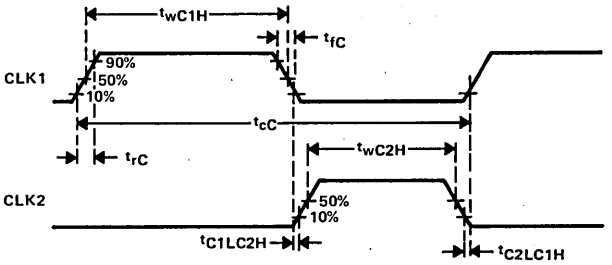


FIGURE 14. CLOCK WAVEFORMS

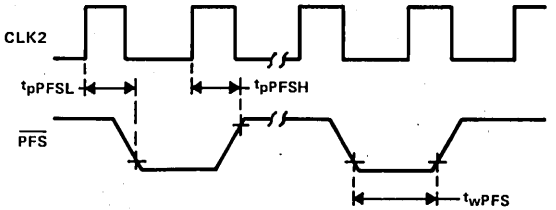


FIGURE 15. RELATIONSHIP OF $\overline{PFS}$ TO CLOCK CYCLES

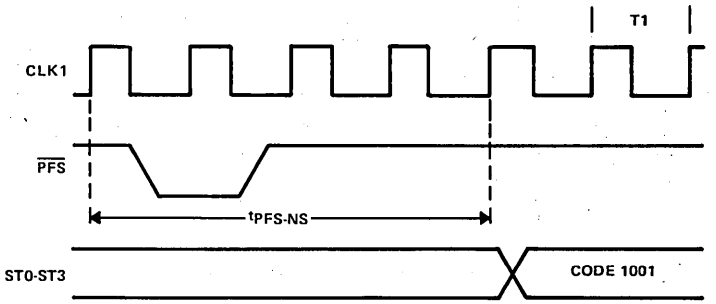


FIGURE 16. GUARANTEED DELAY, $\overline{PFS}$ TO NONSEQUENTIAL FETCH

PARAMETER MEASUREMENT INFORMATION

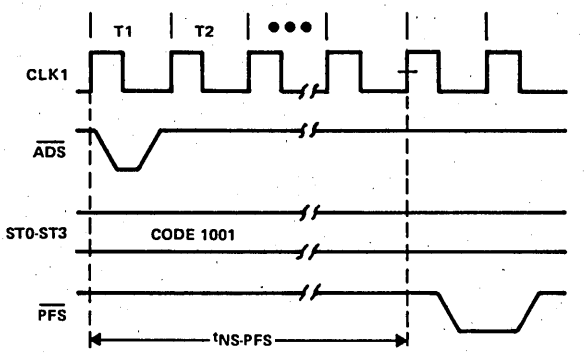


FIGURE 17. GUARANTEED DELAY, NONSEQUENTIAL FETCH TO $\overline{PFS}$

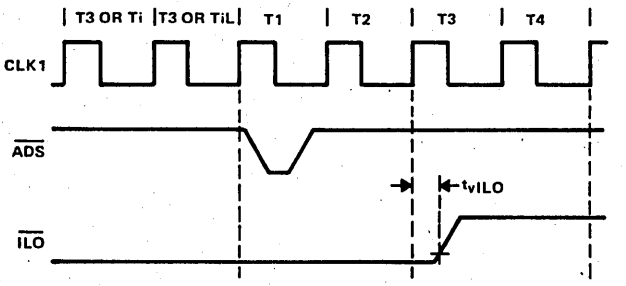


FIGURE 18. RELATIONSHIP OF ILO TO FIRST OPERAND CYCLE OF AN INTERLOCKED INSTRUCTION

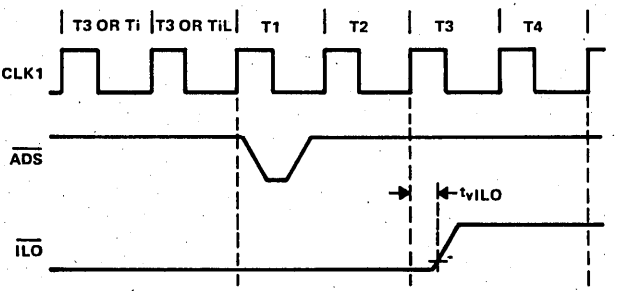


FIGURE 19. RELATIONSHIP OF $\overline{ILO}$ TO LST OPERAND CYCLE OF AN INTERLOCKED INSTRUCTION

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

PARAMETER MEASUREMENT INFORMATIN

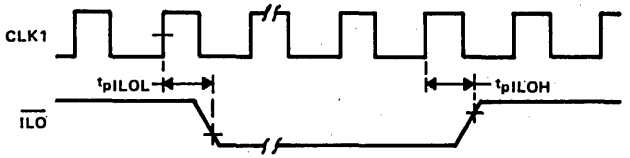


FIGURE 20. RELATIONSHIP OF $\overline{ILO}$ TO ANY CLOCK CYCLE

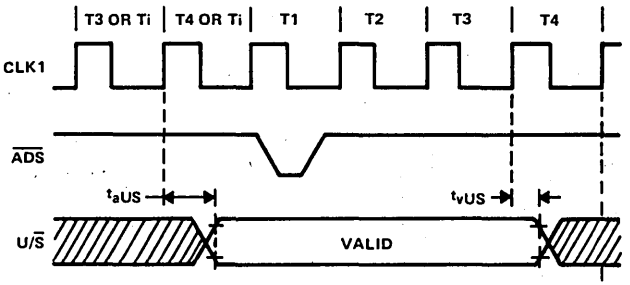


FIGURE 21. RELATIONSHIP TO ANY BUS CYCLE, GUARANTEED VALID INTERVAL

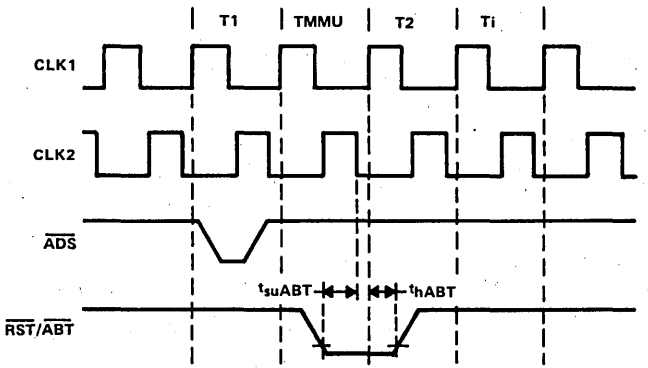


FIGURE 22. ABORT TIMING, $\overline{FLT}$ NOT APPLIED

PARAMETER MEASUREMENT INFORMATION

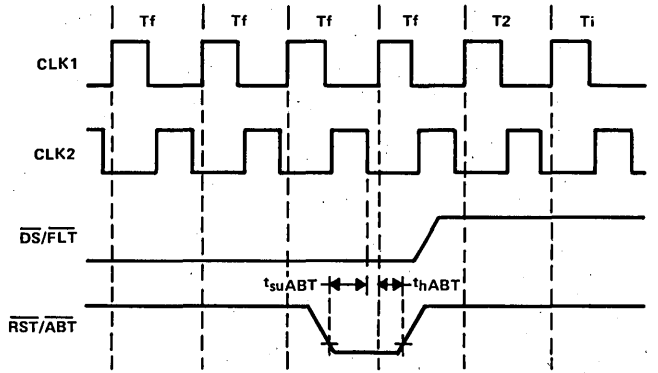


FIGURE 23. ABORT TIMING, $\overline{FLT}$ APLIED

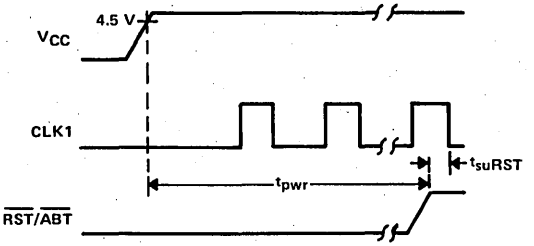


FIGURE 24. POWER-ON RESET

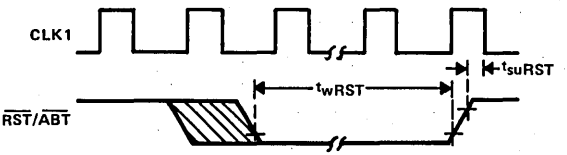


FIGURE 25. NONPOWER-ON RESET

PARAMETER MEASUREMENT INFORMATION

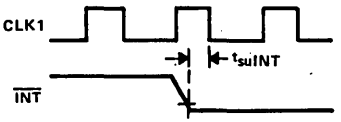


FIGURE 26. $\overline{INT}$ INTERRUPT SIGNAL DETECTION

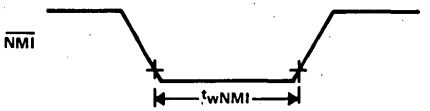


FIGURE 27. $\overline{NMI}$ INTERRUPT SIGNAL TIMING

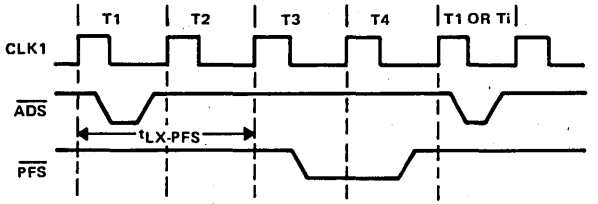
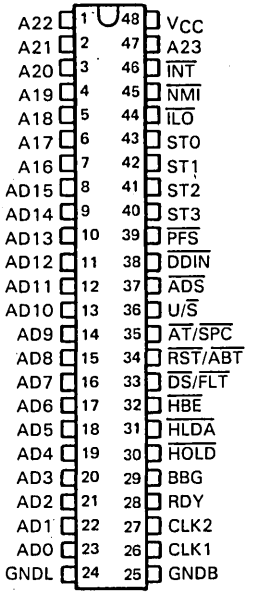


FIGURE 28. RELATIONSHIP BETWEEN LAST DATA TRANSFER OF AN INSTRUCTION AND $\overline{PFS}$ ON NEXT INSTRUCTION

D2878, APRIL 1985

- High-Speed NMOS Technology
- 32-Bit Architecture and Implementation with 24-Bit Address and 16-Bit Data Bus
- 16-Megabyte Uniform Addressing Space
- Powerful Instruction Set with:
  - General 2-Address Capability
  - Very High Degree of Symmetry
  - Addressing Modes Optimized for High-Level Language References
- TI32000 Coprocessor Support
- Single 5-V Operation
- Direct Replacement for National Semiconductor NS32016-10

JD DUAL-IN-LINE PACKAGE
(TOP VIEW)

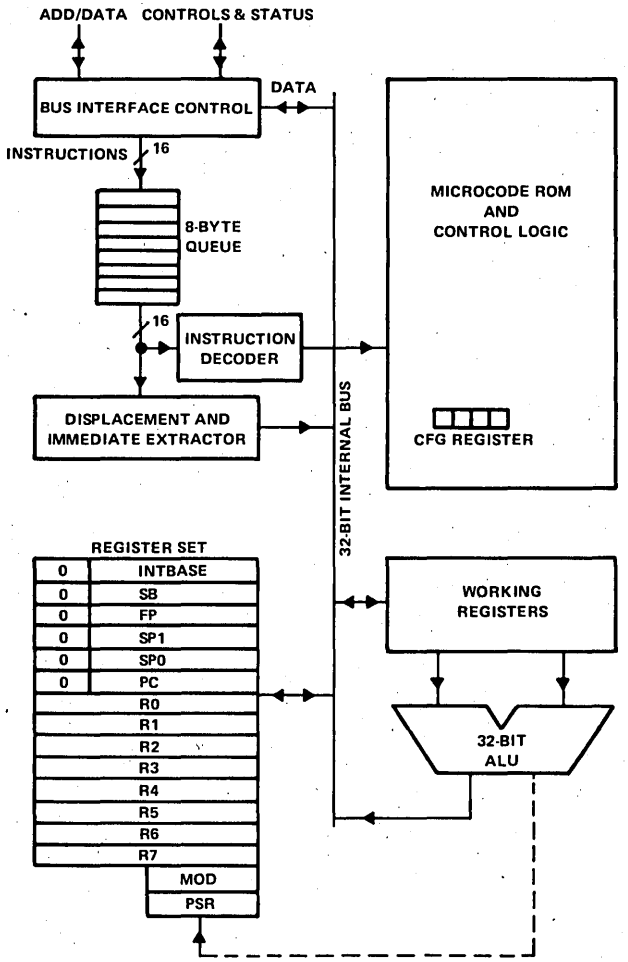


description

The TI32016T-2 is a monolithic high-performance high-speed NMOS microprocessor designed to function as a central processing unit (CPU) in the TI32000 microprocessor family. The device has been designed primarily to support microprocessor users who require the ability to use a large addressing space for large programs and/or large data structures. Because large programs must realistically be generated and maintained in high-level languages, the TI32000 family architecture provides for very efficient compilation while remaining easy to program at the assembler level. The TI32000 family architecture in conjunction with the TI32082 Memory Management Unit (MMU) provides full support for demand-paged virtual memory capability. High-performance floating-point instructions are provided with the TI32081 Floating Point Unit (FPU).

The TI32016T is characterized for operation from 0°C to 70°C.

functional block diagram



PIN FUNCTIONAL DESCRIPTION

Pin NO.	Pin NAME	I/O	DESCRIPTION
1	A22	O	Address bit 22 of the address bus
2	A21	O	Address bit 21 of the address bus
3	A20	O	Address bit 20 of the address bus
4	A19	O	Address bit 19 of the address bus
5	A18	O	Address bit 18 of the address bus
6	A17	O	Address bit 17 of the address bus
7	A16	O	Address bit 16 of the address bus
8	AD15	I/O	Address/Data bit 15. Multiplexed address/data information
9	AD14	I/O	Address/Data bit 14. Multiplexed address/data information
10	AD13	I/O	Address/Data bit 13. Multiplexed address/data information
11	AD12	I/O	Address/Data bit 12. Multiplexed address/data information
12	AD11	I/O	Address/Data bit 11. Multiplexed address/data information
13	AD10	I/O	Address/Data bit 10. Multiplexed address/data information
14	AD9	I/O	Address/Data bit 9. Multiplexed address/data information
15	AD8	I/O	Address/Data bit 8. Multiplexed address/data information
16	AD7	I/O	Address/Data bit 7. Multiplexed address/data information
17	AD6	I/O	Address/Data bit 6. Multiplexed address/data information
18	AD5	I/O	Address/Data bit 5. Multiplexed address/data information
19	AD4	I/O	Address/Data bit 4. Multiplexed address/data information
20	AD3	I/O	Address/Data bit 3. Multiplexed address/data information
21	AD2	I/O	Address/Data bit 2. Multiplexed address/data information
22	AD1	I/O	Address/Data bit 1. Multiplexed address/data information
23	AD0	I/O	Address bit 0 (least significant bit). Multiplexed address/data information
24	GNDL		Logic ground. Ground reference for on-chip logic.
25	GNDB		Buffer Ground. Ground reference for the on-chip drivers connected to output pins.
26	CLK1	I	Clock 1 input. One phase of a two-phase clock input
27	CLK2	I	Clock 2 input. One phase of a two-phase clock input
28	RDY	I	READY input. When high, causes the CPU to terminate the bus cycle. When low, causes the CPU to extend the current bus cycle to provide for a slower memory or peripheral reference.
29	BBG	O	Back-Bias Generator supply. Output of on-chip substrate voltage generator
30	$\overline{\text{HOLD}}$	I	Hold request input. When low, causes the CPU to release the bus for DMA or multiprocessing purposes
31	HLDA	O	Hold acknowledge output. Applied by the CPU in response to a $\overline{\text{HOLD}}$ input. Low level indicates that the bus has been released for DMA or multiprocessing purposes.
32	$\overline{\text{HBE}}$	O	High Byte Enable output. When low, enables data transfers on most significant byte of data bus.
33	$\overline{\text{DS/FLT}}$	I/O	Data Strobe/Float. Data strobe output or float command input. Function is selectd by $\overline{\text{AT/SPC}}$ input.
34	$\overline{\text{RST/ABT}}$	I	Reset/Abort input. If held low for one clock cycle and released, causes an abort command. If held low for longer than one clock cycle, initiates a reset.
35	$\overline{\text{AT/SPC}}$	I/O	Address Translation/Coprocessor Control. Used by the CPU as the data strobe output for coprocessor transfers. Used by coprocessors to acknowledge completion of an instruction. Sampled on the trailing edge of reset pulse as address translation strap.

PIN FUNCTIONAL DESCRIPTION

PIN NO.	PIN NAME	I/O	DESCRIPTION
36	U/$\overline{S}$	O	User/Supervisor status output. When high, indicates user mode. When low, indicates supervisor mode.
37	$\overline{ADS}$	O	Address strobe output. Available to the system for address latch control. Indicates start of a bus cycle by going low.
38	$\overline{DDIN}$	O	Data Direction In output. Status signal that indicates the direction of data transfer during a bus cycle, low for read, high for write.
39	$\overline{PFS}$	O	Program Flow Status Output. Low-going pulse indicates beginning of an instruction execution.
40	ST3	O	Status bit 3 output. Bus-cycle status code
41	ST2	O	Status bit 2 output. Bus cycle status code
42	ST1	O	Status bit 1 output. Bus cycle status code
43	ST0	O	Status bit 0 (least significatn bit) output. Bus cycle status code
44	$\overline{ILO}$	O	Interlocked Operation Output. When low, indicates that an interlocked instruction is being executed.
45	$\overline{NMI}$	I	Nonmaskable Interrupt input. Nonmaskable interrupt request
46	$\overline{INT}$	I	Interrupt input. Maskable interrupt request
47	A23	O	Address bit 23 of the address bus
48	V_{CC}		+5 V supply voltage

absolute maximum ratings over recommended operating conditions (unless otherwise noted)

Supply voltage, V_{CC} (see Note 1) . . . . . 7 V
Input voltage range . . . . . −5 V to 7 V
Output voltage range . . . . . −5 V to 7 V
Continuous total dissipation . . . . . 1.5 W
Operating free-air temperature range, T_A . . . . . 0°C to 70°C

NOTE 1: All voltage values are with respect to the common ground.

recommended operating conditions

		MIN	NOM	MAX	UNIT
Supply voltage, V_{CC}		4.75	5	5.25	V
High-level input voltage, V_{IH}	CLK1, CLK2	$V_{CC}-0.4$		$V_{CC}+0.5$	V
	All other inputs	2		$V_{CC}+0.5$	
Low-level input voltage, V_{IL}	CLK1, CLK2	−0.5		0.3	V
	All other inputs	−0.5		0.8	
Operating free-air temperature, T_A		0		70	°C

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

electrical characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

PARAMETER			TEST CONDITION	MIN	TYP†	MAX	UNIT
V_{OH}	High-level output voltage		I_{OH} = −400 μA	2.4			V
V_{OL}	Low-level output voltage		I_{OL} = 2 mA			0.45	V
I_I	Input current	$\overline{AT}/SPC$	V_I = 0.4 V	50		1000	μA
		All other inputs except CLK1, CLK2, and $\overline{AT}/SPC$	V_I = 0 to V_{CC}			±20	μA
$I_{O(off)}$	Off-state output current		V_O = 0.4 V to V_{CC}			±20	μA
I_{CC}	Supply current		I_O = 0, T_A = 25°C		180	300	mA

†All typical values are at V_{CC} = 5 V, T_A = 25°C.

Timing parameter symbols have been created in accordance with JEDEC Standard 100. In order to shorten the symbols, the pin names have been further abbreviated as follows:

AD	A	$\overline{HBE}$	BE
CLK1	C or C1	$\overline{HLDA}$	HA
CLK2	C2	$\overline{HOLD}$	HD
$\overline{DDIN}$	DD	U/$\overline{S}$	US
FCLK	F		

Lowercase subscripts and their meaning are:

c	cycle time (period)
dis	disable time (H or L to Z)
en	enable time (Z to H or L)
f	fall time
p	propagation delay time
pwr	supply voltage stable
r	rise time
su	setup time
w	pulse duration

Parentheses have been omitted except when required for separation of a final subscript representing special conditions. The columns titled NSC SYMBOL show the symbols used by National Semiconductor for the parameters shown in the timing requirements and switching characteristics tables.

timing requirements over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER		FIGURE	NSC SYMBOL	MIN	MAX	UNIT
$t_{suD(RD)}$	Setup time, read data before CLK2↓	2,11	t_{DIs}	10		ns
$t_{hD(RD)}$	Hold time, read data after CLK1↑	2,11	t_{DIh}	10		ns
t_{suHDL}	Setup time, $\overline{HOLD}$ low before CLK2↓ (see Note 3)	3, 4	t_{HLDa}	25		ns
t_{hHD}	Hold time, $\overline{HOLD}$ low or high after CLK1↑	3,4,5	t_{HLDh}	0		ns
t_{suHDH}	Setup time, $\overline{HOLD}$ high before CLK2↓	5	t_{HLDia}	25		ns
t_{suFLTL}	Setup time, $\overline{FLT}$ low before CLK2↓	6	t_{FLTa}	25		ns
t_{suFLTH}	Setup time, $\overline{FLT}$ high before CLK2↓	7	t_{FLTia}	25		ns
t_{suRDY}	Setup time, RDY low or high before CLK2↓	8, 9	t_{RDYs}	15		ns
t_{hRDY}	Hold time, RDY low or high after CLK1↓	8, 9	t_{RDYh}	0		ns
t_{wSPC}	Pulse duration, $\overline{SPC}$ low at 0.8 V (both edges)	10	t_{SPCw}	20		ns
t_{suAT}	Setup time, $\overline{AT}/\overline{SPC}$ low before CLK1↑	13	t_{ATs}	$1t_{cC}$		
t_{hAT}	Hold time, $\overline{AT}/\overline{SPC}$ low after CLK1↓	13	t_{ATh}	$2t_{cC}$		
t_{rC}	Rise time, CLK1 or CLK2	14	t_{CLr}		7	ns
t_{fC}	Fall time, CLK1 or CLK2	14	t_{CLf}		7	ns
t_{cC}	Clock period	14	t_{Cp}	100	5000	ns
t_{wC1H}	Pulse duration, CLK1 high	14	$t_{CLw(1)}$	$0.5t_{cC}-10$		ns
t_{wC2H}	Pulse duration, CLK2 high	14	$t_{CLw(2)}$	$0.5t_{cC}-10$		ns
	Clock pulse asymmetry, t_{wC1H}-t_{wC2H}	14	t_{CLwas}		±5	ns
t_{C1LC2H}	Nonoverlap time at 10% of CLK1↓ and CLK2↑	14	$t_{nOVL(1)}$	0	7	ns
t_{C2LC1H}	Nonoverlap time at 10% of CLK2↓ and CLK1↑	14	$t_{nOVL(2)}$	0	7	ns
	Nonoverlap asymmetry, $t_{C1LC2H}-t_{C12LC1H}$	14	t_{nOVLas}		±4	ns
t_{suABT}	Setup time, $\overline{ABT}$ low before CLK2↓	22,23	t_{ABTs}	20		ns
t_{hABT}	Hold time, $\overline{ABT}$ Low after CLK1↑	22,23	t_{ABTh}	0		ns
t_{pwr}	Supply voltage stable (above 4.5 V) to $\overline{RST}$ high	24	t_{PWR}	50		μs
t_{suRST}	Setup time, $\overline{RST}$ high before CLK1↓	24, 25	t_{RSTs}	20		ns
t_{wRST}	Puls duration, $\overline{RST}$ low at 0.8 V (both edges)	25	t_{RSTw}	$64t_{cC}$		
t_{suINT}	Setup time, $\overline{INT}$ low before CLK1↓	26	t_{INTs}	20		ns
t_{wNMI}	Pulse duration, $\overline{NMI}$ low at 0.8 V (both edges)	27	t_{NMIw}	70		ns

NOTE 3: This setup time is necessary to ensure prompt acknowledgement via $\overline{HLDA}$ and the ensuing floating of the CPU off the buses. The time from the receipt of the $\overline{HOLD}$ signal until the CPU floats is a function of the time $\overline{HOLD}$ signal goes low, the state of the RDY input (in MMU systems), and the length of the current MMU cycle.

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

switching characteristics over recommended ranges of supply voltage and operating free-air temperature, C_L = 100 pF (unless otherwise noted)

PARAMETER		FIGURE	NSC SYMBOL	MIN	MAX	UNIT
$t_{aA(1)}$	Access time, AD0 through AD15 after CLK1↑	1	t_{ALv}		50	ns
$t_{vA(1)}$	Valid time, AD0 through AD15 after CLK1↑	1	t_{ALh}	10		ns
t_{aD}	Access time, AD0 through AD15 after CLK1↑	1	t_{Dv}		50	ns
t_{vD}	Valid time, AD0 through AD15 after CLK1↑	1	t_{Dh}	0		ns
$t_{aA(2)}$	Access time, A16 through A23 after CLK1↑	1	t_{AHv}		50	ns
$t_{vA(2)}$	Valid time, A16 through A23 after CLK1↑	1	t_{AHh}	0		ns
t_{aBE}	Access time, $\overline{HBE}$ after CLK1↑	1	t_{HBEv}		70	ns
t_{vBE}	Valid time, $\overline{HBE}$ after CLK1↑	1	t_{HBEh}	0		ns
t_{aST}	Access time, ST0 through ST3 after CLK1↑	1	t_{STv}		45	ns
t_{vST}	Valid time, ST0 through ST3 after CLK1↑	1	t_{STh}	0		ns
t_{pADSL}	Propagation delay time, CLK1↑ to $\overline{ADS}$ low	1	t_{ADSa}		35	ns
t_{pADSH}	Propagation delay time, CLK2↑ to $\overline{ADS}$ high	1	t_{ADSia}	15	45	ns
t_{wADS}	Pulse duration, $\overline{ADS}$ low at 0.8 V (both edges)	1	t_{ADSw}	35		ns
t_{pDSL}	Propagation delay time, CLK1↑ to $\overline{DS}$ low	1	t_{DSa}		45	ns
t_{pDSH}	Propagation delay time, CLK1↑ to $\overline{DS}$ high	1	t_{DSia}	10	40	ns
$t_{AV\text{-}ADSH(1)}$	Time interval, AD0 through AD15 valid to $\overline{ADS}$ high	2	t_{ALADSs}	25		ns
$t_{AV\text{-}ADSH(2)}$	Time interval, A16 through A23 valid to $\overline{ADS}$ high	2	t_{AHADSs}	25		ns
$t_{disA(1)}$	Disable time, AD0 through AD15 after CLK1↑ (without MMU)	2	t_{ALf}		25	ns
t_{aDD}	Access time, $\overline{DDIN}$ after CLK1↑	2	t_{DDINv}		65	ns
t_{vDD}	$\overline{DDIN}$ valid after CLK1↑	2	t_{DDINh}	0		ns
$t_{disA(1)}$	Disable time, AD0 through AD15 after CLK1↑ ($\overline{HOLD}$ low)	3	t_{ALf}		25	ns
$t_{disA(2)}$	Disable time, A16 through A23 after CLK↑ ($\overline{HOLD}$ low)	3	t_{AHf}		25	ns
t_{disADS}	Disable time, $\overline{ADS}$ after CLK1↑ ($\overline{HOLD}$ low)	3,4	t_{ADSf}		55	ns
t_{disBE}	Disable time, $\overline{HBE}$ after CLK1↑ ($\overline{HOLD}$ low)	3,4	t_{HBEf}		55	ns
t_{disDD}	Disable time, $\overline{DDIN}$ after CLK1↑ ($\overline{HOLD}$ low)	3,4	t_{DDINf}		55	ns
t_{pHAL}	Propagation delay time, CLK1↑ to $\overline{HLDA}$ low	3,4	t_{HLDAa}		75	ns
t_{pHAH}	Propagation delay time, CLK1↑ to $\overline{HLDA}$ high	5	t_{HLDAia}		75	ns
t_{enADS}	Enable time, $\overline{ADS}$ after CLK1↑ ($\overline{HOLD}$ high)	5	t_{ADSr}		55	ns
t_{enBE}	Enable time, $\overline{HBE}$ after CLK1↑($\overline{HOLD}$ high)	5	t_{HBEr}		55	ns
t_{enDD}	Enable time, $\overline{DDIN}$ after CLK1↑ ($\overline{HOLD}$ high)	5	t_{DDINr}		55	ns
$t_{vADSH\text{-}A(1)}$	Valid time, AD0 through AD15 after $\overline{ADS}$↑	6	t_{ALADSh}	10		ns
$t_{vADSH\text{-}A(2)}$	Valid time, A16 through A23 after $\overline{ADS}$↑	6	t_{AHADSh}	10		ns
$t_{disA(1)}$	Disable time, AD0 through AD15 after CLK1↑ (with MMU)	6	t_{ALMf}		25	ns
$t_{disA(2)}$	Disable time, A16 through A23 after CLK1↑ (with MMU)	6	t_{AHMf}		25	ns
$t_{disA(1)}$	Disable time, AD0 through AD15 after CLK1↑ ($\overline{FLT}$ Low)	6	t_{ALf}		30	ns
t_{disDD}	Disable time, $\overline{DDIN}$ after $\overline{FLT}$↑	6	t_{DDINf}		50	ns
$t_{FLTL\text{-}BEL}$	Time interval, $\overline{FLT}$ low to $\overline{HBE}$ low	6	t_{HBEl}		65	ns
t_{enDD}	Enable time, $\overline{DDIN}$ after $\overline{FLT}$↑	7	t_{DDINr}		50	ns
t_{enBE}	Enable time, $\overline{HBE}$ after $\overline{FLT}$↑	7	t_{HBEr}		75	ns
t_{pSPCL}	Propagation delay time, CLK1↑ to $\overline{SPC}$ low	10	t_{SPCa}		35	ns
t_{pSPCH}	Propagation delay time, CLK1↑ to $\overline{SPC}$ high	10	t_{SPCia}		35	ns

switching characteristics (continued)

PARAMETER		FIGURE	NSC SYMBOL	MIN	MAX	UNIT
t_{aD}	Access time, write data (bits AD0 through AD15) after CLK1↑	10	t_{Dv}		50	ns
t_{vD}	Valid time, write data (bits AD0 through AD15) after CLK1↑	10	t_{Dh}	0		ns
t_{disSPC}	Disable time, $\overline{SPC}$ output nonforcing after CLK2↑	12	t_{SPCnf}		10	ns
t_{wPFS}	Pulse duration, $\overline{PFS}$ low at 0.8 V (both edges)	15	t_{PFSw}	70		ns
t_{pPFSL}	Propagation delay time, CLK2↑ to $\overline{PFS}$ low	15	t_{PFSa}		50	ns
t_{pPFSH}	Propagation delay time, CLK2↑ to $\overline{PFS}$ high	15	t_{PFSia}		50	ns
t_{PFS-NS}	Time interval, $\overline{PFS}$ clock cycle to next nonsequential fetch	16	t_{PFNS}	$4t_{cC}$		
t_{NS-PFS}	Time interval, nonsequential fetch to next $\overline{PFS}$ clock cycle	17	t_{NSPF}	$4t_{cC}$		
$t_{ILOL-C1}$	Time interval, $\overline{ILO}$ low to CLK1↑	18	t_{ILOs}	30		ns
t_{vILO}	Valid time, $\overline{ILO}$ low after CLK1↑	19	t_{ILOh}	10		ns
t_{pILOL}	Propagation delay time, CLK1↑ to $\overline{ILO}$ low	20	t_{ILOa}		70	ns
t_{pILOH}	Propagation delay time, CLK1↑ to $\overline{ILO}$ high	20	t_{ILOia}		70	ns
t_{aUS}	Access time, U/$\overline{S}$ after CLK1↑	21	t_{USs}		70	ns
t_{vUS}	Valid time, U/$\overline{S}$ after CLK1↑	21	t_{USh}	10		ns
t_{LX-PFS}	Last operand transfer of an insruction to next $\overline{PFS}$ clock cycle	28	t_{LXPF}	$0t_{cC}$		

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

PARAMETER MEASUREMENT INFORMATION†

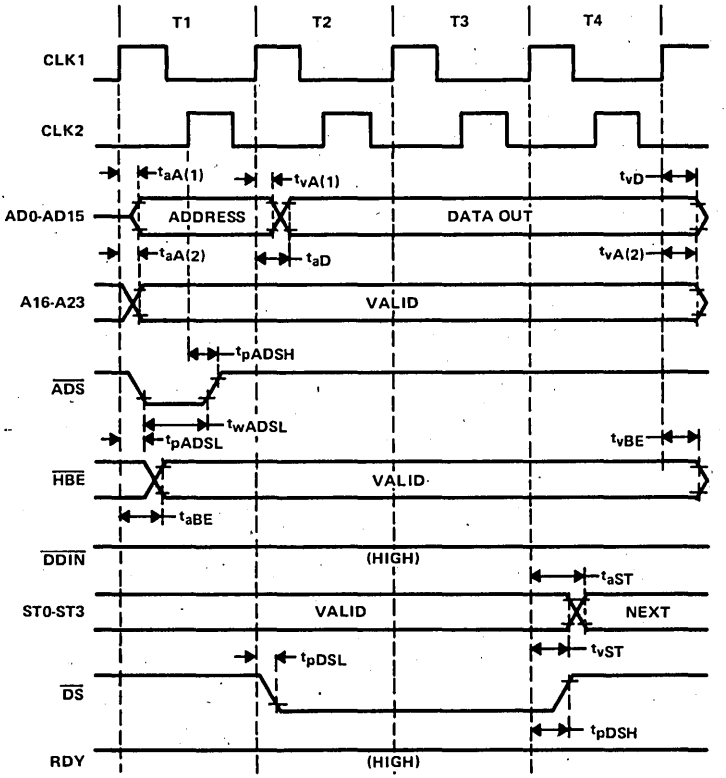


FIGURE 1. WRITE CYCLE

†In Figures 1 through 28, time intervals are defined with respect to the following reference points:
For clock signals, the 50% points.
For all other signals, 2 V if the high level is indicated and 0.8 V if the low level is indicated.

PARAMETER MEASUREMENT INFORMATION

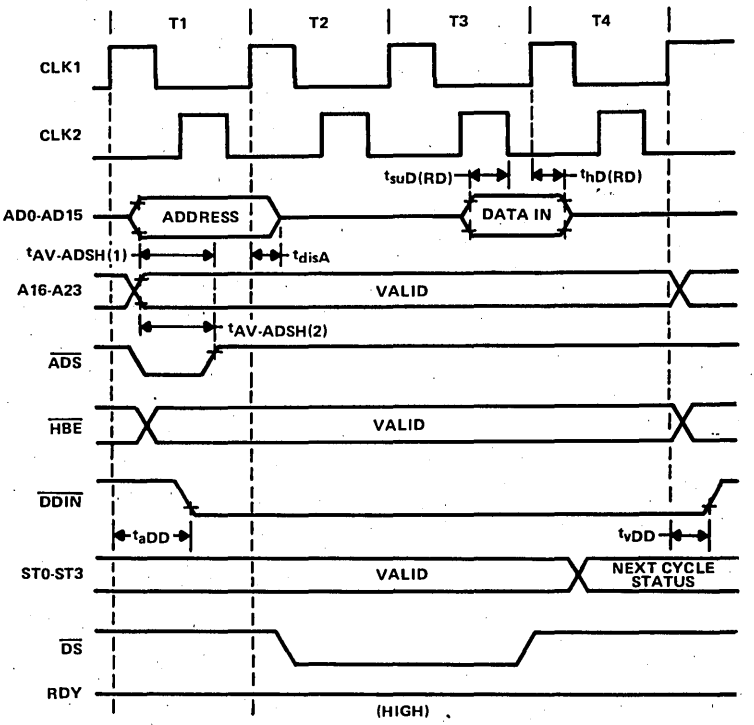
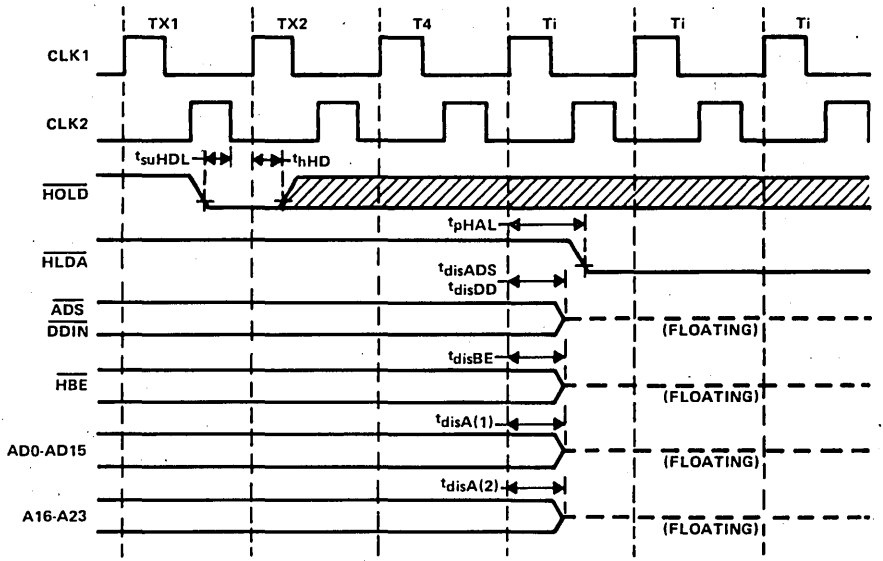


FIGURE 2. READ CYCLE

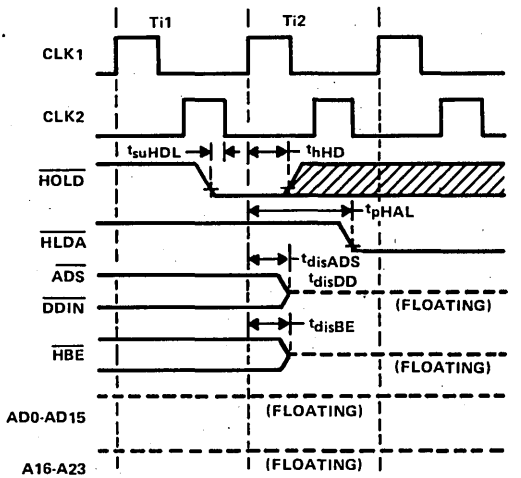
TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

PARAMETER MEASUREMENT INFORMATION



NOTE: When the CPU is not idling (not in Ti), the $\overline{HOLD}$ request ($\overline{HOLD}$ low) must be active before the trailing edge of CLK2 that appears two clock cycles before T4 (TX1) and stay low until t_{hHDL} after the leading edge of CLK1 that precedes T4 (TX2) for the request to be acknowledge.

FIGURE 3. FLOATING BY $\overline{HOLD}$ TIMING (CPU NOT IDLE INITIALLY)



NOTE: During T1 the CPU is already idling.

FIGURE 4. FLOATING BY $\overline{HOLD}$ TIMING (CPU INITIALLY IDLE)

PARAMETER MEASUREMENT INFORMATION

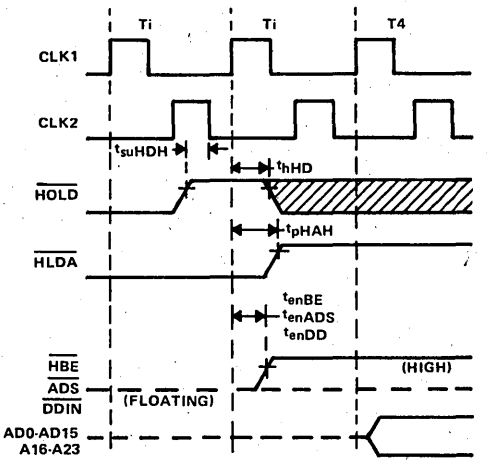
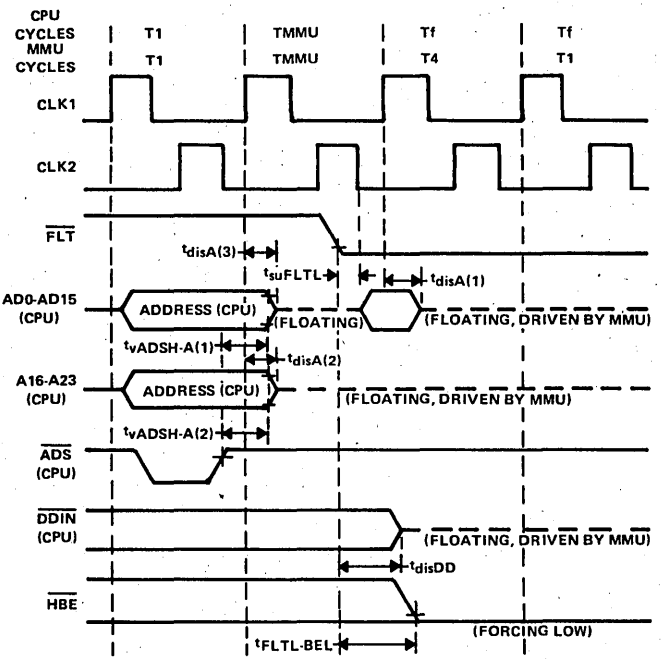


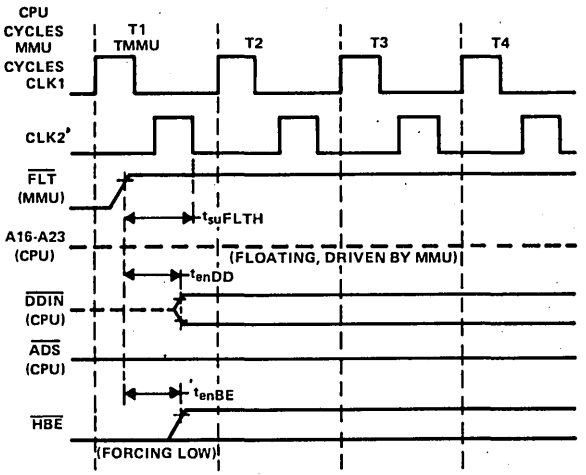
FIGURE 5. RELEASE FROM $\overline{\text{HOLD}}$



NOTE: When $\overline{\text{FLT}}$ goes high, the CPU restarts driving $\overline{\text{DDIN}}$ before the MMU releases it. This does not cause any conflict since both CPU and MMU force $\overline{\text{DDIN}}$ to the same logic level.

FIGURE 6. $\overline{\text{FLT}}$ INITIATED FLOAT CYCLE TIMING

PARAMETER MEASUREMENT INFORMATION



NOTE: When $\overline{FLT}$ goes high, the CPU restarts driving $\overline{DDIN}$ before the MMU releases it. This does not cause any conflict since both CPU and MMU force $\overline{DDIN}$ to the same logic level.

FIGURE 7. RELEASE FROM $\overline{FLT}$ TIMING

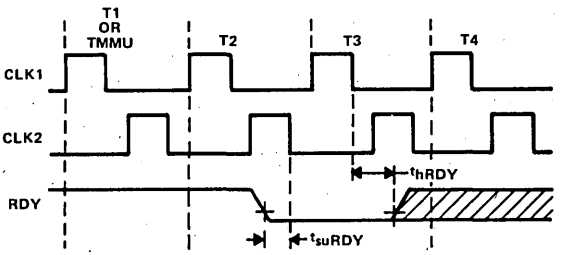


FIGURE 8. READY SAMPLING (CPU INITIALLY READY)

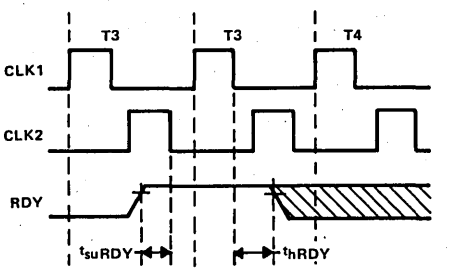


FIGURE 9. READY SAMPLING (CPU INITIALLY NOT READY)

PARAMETER MEASUREMENT INFORMATION

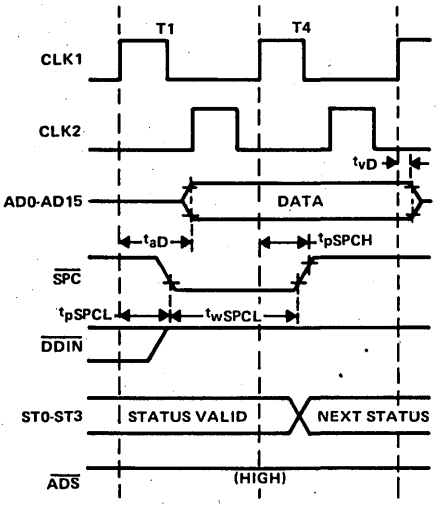


FIGURE 10. COPROCESSOR WRITE TIMING

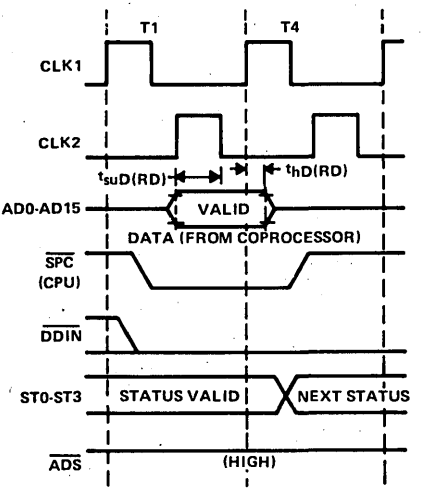


FIGURE 11. COPROCESSOR READ TIMING

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

PARAMETER MEASUREMENT INFORMATION

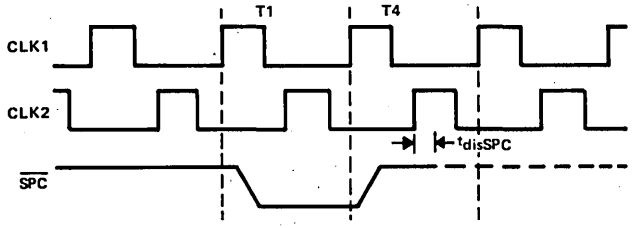


FIGURE 12. $\overline{SPC}$ NONFORCING DELAY

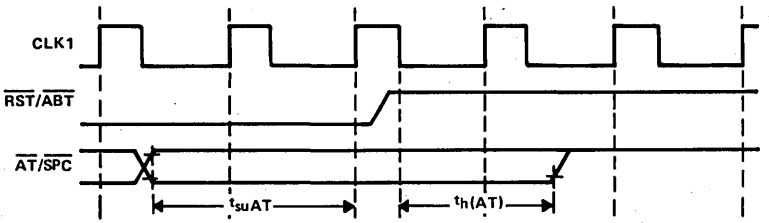


FIGURE 13. RESET CONFIGURATION TIMING

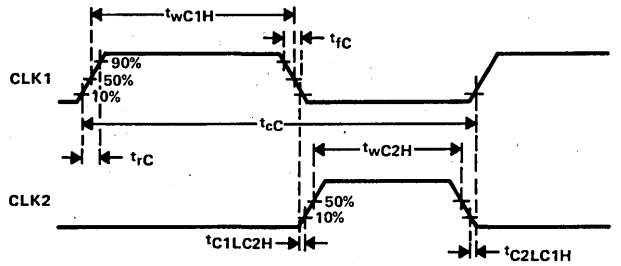


FIGURE 14. CLOCK WAVEFORMS

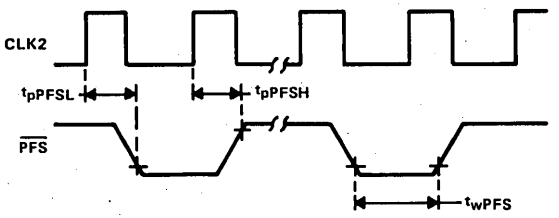


FIGURE 15. RELATIONSHIP OF $\overline{PFS}$ TO CLOCK CYCLES

PARAMETER MEASUREMENT INFORMATION

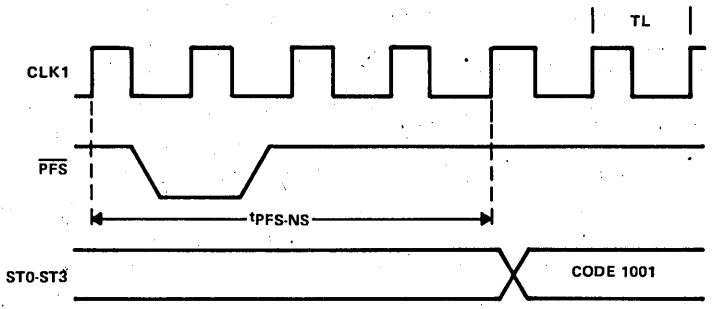


FIGURE 16. GUARANTEED DELAY, $\overline{PFS}$ TO NONSEQUENTIAL FETCH

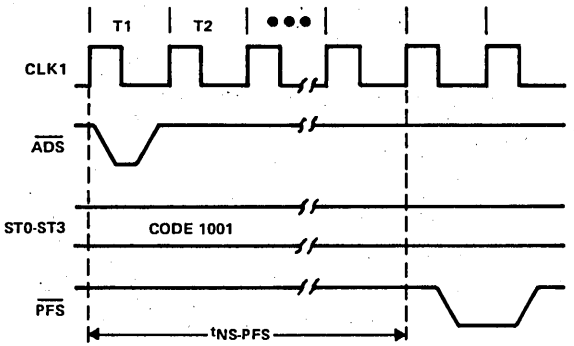


FIGURE 17. GUARANTEED DELAY, NONSEQUENTIAL FETCH TO $\overline{PFS}$

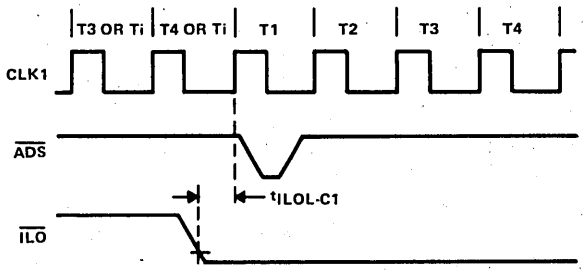


FIGURE 18. RELATIONSHIP OF $\overline{ILO}$ TO FIRST OPERAND CYCLE OF AN INTERLOCKED INSTRUCTION

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

PARAMETER MEASUREMENT INFORMATIN

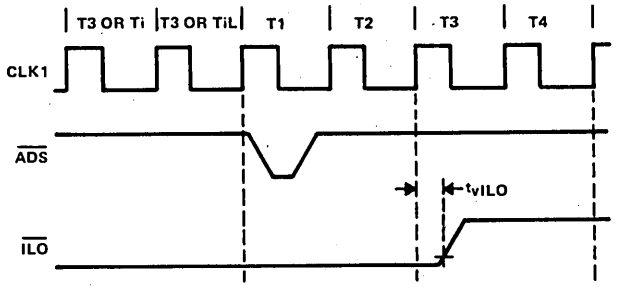


FIGURE 19. RELATIONSHIP OF $\overline{ILO}$ TO LST OPERAND CYCLE OF AN INTERLOCKED INSTRUCTION

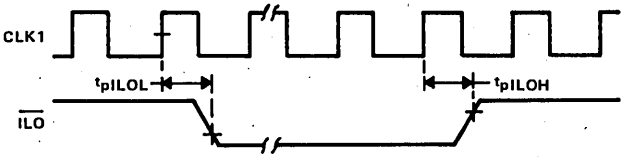


FIGURE 20. RELATIONSHIP OF $\overline{ILO}$ TO ANY CLOCK CYCLE

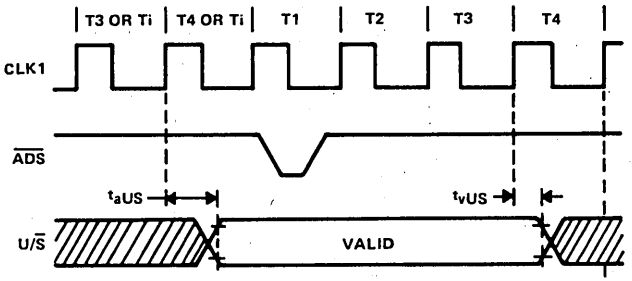


FIGURE 21. RELATIONSHIP TO ANY BUS CYCLE, GUARANTEED VALID INTERVAL

4 Data Sheets

PARAMETER MEASUREMENT INFORMATION

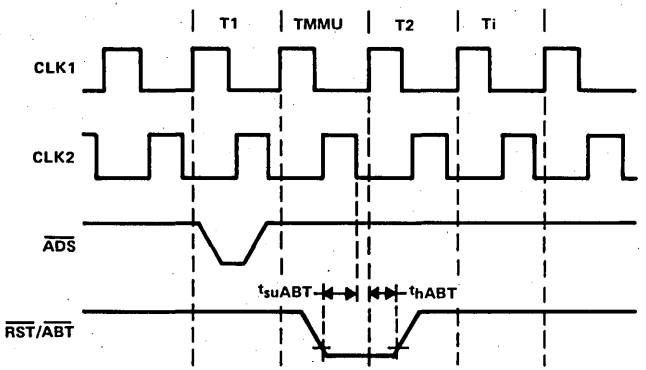


FIGURE 22. ABORT TIMING, $\overline{FLT}$ NOT APPLIED

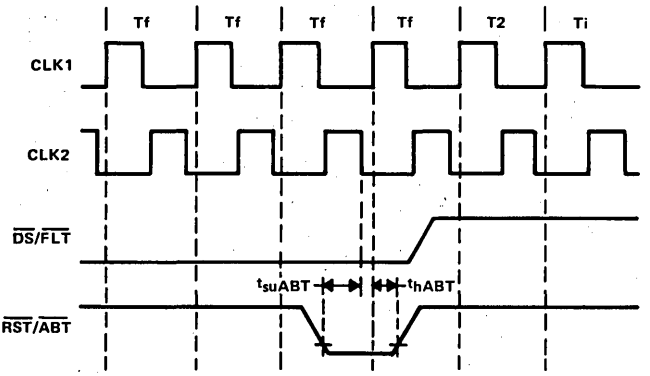


FIGURE 23. ABORT TIMING, $\overline{FLT}$ APPLIED

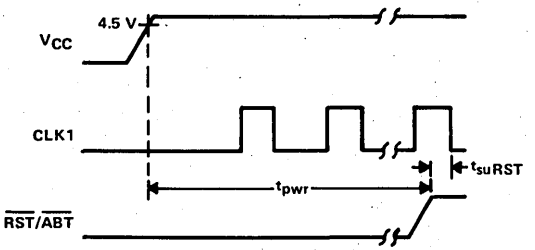


FIGURE 24. POWER-ON RESET

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

PARAMETER MEASUREMENT INFORMATION

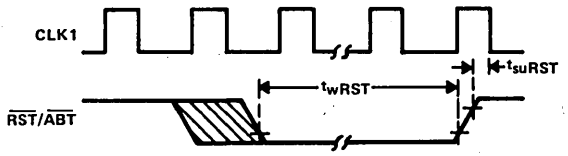


FIGURE 25. NONPOWER-ON RESET

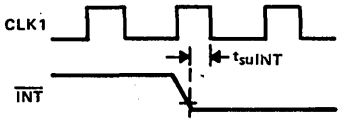


FIGURE 26. INT INTERRUPT SIGNAL DETECTION

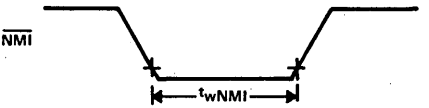


FIGURE 27. NMI INTERRUPT SIGNAL TIMING

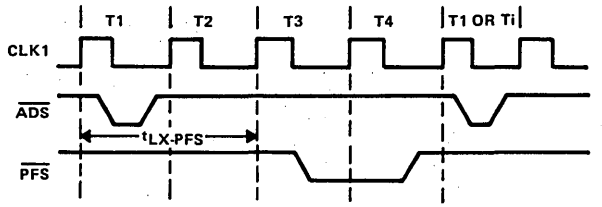
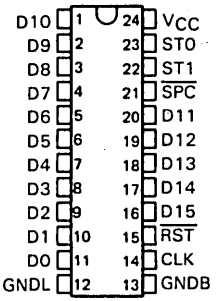


FIGURE 28. RELATIONSHIP BETWEEN LAST DATA TRANSFER OF AN INSTRUCTION AND PFS ON NEXT INSTRUCTION

D2871, APRIL 1985

- High-Speed NMOS Technology
- Supports Proposed IEEE Standard (Task P754) for Binary Floating-Point Arithmetic
- Standard (32-Bit) and Long (64-Bit) Data Floating-Point Operation
- 8 On-Chip Data Registers
- Direct Compatibility with TI32032D-2, TI32032T-2, and TI32016T-2 CPUs
- Direct Replacement for National Semiconductor NS32081-10

J OR N DUAL-IN-LINE PACKAGE
(TOP VIEW)



description

The TI32081W-2 is a monolithic high-speed NMOS Floating-Point Unit (FPU) integrated circuit that functions as a coprocessor. The TI32081W-2 provides both standard (32-bit, single-precision) and long (64-bit, double-precision) operations, including add, subtract, multiply, divide, move, convert, and compare functions. It is designed to support the proposed IEEE standard for binary floating-point arithmetic (Task P754) and to be architecturally consistent with the full two-address architecture and addressing modes of the TI32000 microprocessor family.

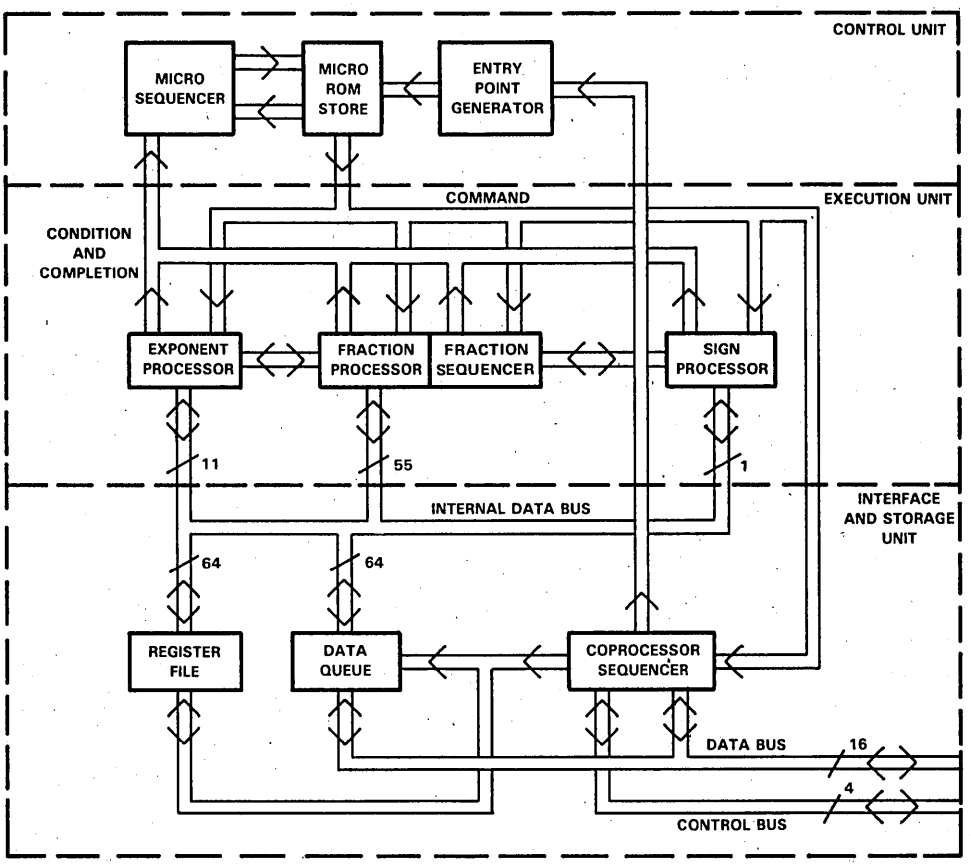
The operand and result information transfers take place on a 16-bit bidirectional data bus. The status and control are handled via two status lines and one control line. The CPU initiates the coprocessor cycle and transfers the operation word and operands to the TI32081W-2. The TI32081W-2 then executes the floating-point instruction and signals the CPU upon completion. The CPU reads the error status and traps, if appropriate. Otherwise, it reads the results and transfers it to the destination. This coprocessor cycle is directly supported by TI32000 microprocessor family CPUs or can be emulated by other CPUs.

The TI32081W-2 features single 5-volt supply operation. It also features operation from a single-phase TTL-level clock that may be from any source. Typically, the clock signal is provided by the TI32201 Timing Control Unit. The TI32081W-2 also features power-up reset capabilities to ensure that all on-chip voltages are completely stable before operation.

The TI32081W-2 is characterized for operation from 0°C to 70°C.

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

functional block diagram



TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

PIN FUNCTIONAL DESCRIPTION

PIN NO.	PIN NAME	I/O	DESCRIPTION
1	D10	I/O	Data bit 10 of the data bus
2	D9	I/O	Data bit 9 of the data bus
3	D8	I/O	Data bit 8 of the data bus
4	D7	I/O	Data bit 7 of the data bus
5	D6	I/O	Data bit 6 of the data bus
6	D5	I/O	Data bit 5 of the data bus
7	D4	I/O	Data bit 4 of the data bus
8	D3	I/O	Data bit 3 of the data bus
9	D2	I/O	Data bit 2 of the data bus
10	D1	I/O	Data bit 1 of the data bus
11	D0	I/O	Data bit 0 (LSB) of the data bus
12	GNDL		Logic ground. Ground reference for on-chip logic
13	GNDB		Buffer ground. Ground reference for on-chip output data bus drivers connected to output pins
14	CLK	I	TTL-level clock signal
15	$\overline{RST}$	I	Reset input. When low, initiates a reset.
16	D15	I/O	Data bit 15 (MSB) of the data bus
17	D14	I/O	Data bit 14 of the data bus
18	D13	I/O	Data bit 13 of the data bus
19	D12	I/O	Data bit 12 of the data bus
20	D11	I/O	Data bit 11 of the data bus
21	$\overline{SPC}$	I/O	Coprocessor Control. Used by the CPU as the data strobe for transfers to and from the FPU. Used by FPU to signal completion of an instruction.
22	ST1	I	Status bit 1 input. Bus cycle status code from CPU
23	ST0	I	Status bit 0 input. Bus cycle status code from CPU
24	V_{CC}		+5 V supply voltage

absolute maximum ratings over recommended operating conditions (unless otherwise noted)

Supply voltage, V_{CC} (see Note 1) . . . . . 7 V
Input voltage range . . . . . −0.5 V to 7 V
Output voltage range . . . . . −0.5 V to 7 V
Continuous total dissipation . . . . . 1.5 W
Operating free-air temperature range, T_A . . . . . 0°C to 70°C

NOTE 1: All voltage values are with respect to the network ground terminals.

recommended operating conditions

	MIN	NOM	MAX	UNIT
Supply voltage, V_{CC}	4.75	5	5.25	V
High-level input voltage, V_{IH}	2		V_{CC}+0.5	V
Low-level input voltage, V_{IL}	−0.5		0.8	V
High-level output current, I_{OH}			−400	μA
Low-level output current, I_{OL}			2	mA
Operating free-air temperature, T_A	0		70	°C

The algebraic convention, where the less positive (more negative) limit is designated minimum, is used in this data sheet for logic voltage levels only.

electrical characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

	PARAMETER	TEST CONDITION	MIN	MAX	UNIT
V_{OH}	High-level output voltage	I_{OH} = −400 μA	2.4		V
V_{OL}	Low-level output voltage	I_{OL} = 2 mA		0.45	V
I_I	Input current	V_I = 0 to V_{CC}		±10	μA
$I_{O(off)}$	Off-state output current	V_O = 0.45 V to 2.4 V		±20	μA
I_{CC}	Supply current	I_O = 0, T_A = 25°C		300	mA

Timing parameter symbols have been created in accordance with JEDEC Standard 100. In order to shorten the symbols, the pin names have been further abbreviated as follows:

CLK C

Lowercase subscripts and their meaning are:

c	cycle time (period)
dis	disable time (H or L to Z)
en	enable time (Z to H or L)
f	fall time
p	propagation delay time
r	rise time
su	setup time
w	pulse duration

Parentheses have been omitted except when required for separation of a final subscript representing special conditions. The columns titled NSC SYMBOL show the symbols used by National Semiconductor for the parameters shown in the timing requirements and switching characteristics tables.

timing requirements over recommended operating free-air temperature range (unless otherwise noted)

	PARAMETER	NSC SYMBOL	MIN	MAX	UNIT
f_{clock}	Clock frequency			10	MHz
t_{cC}	Clock period	t_{CLKp}	100	2000	ns
t_{wCH}	Pulse duration, CLK high	t_{CLKh}	42		ns
t_{wCL}	Pulse duration, CLK low	t_{CLKl}	42		ns
t_{pwr}	Power-on before $\overline{RST}$↑	t_{PWR}	50		μs
t_{wRST}	Pulse duration, $\overline{RST}$ low	t_{RSTw}	$64t_{cC}$		
t_{suST}	Setup time, status before $\overline{SPC}$↓	t_{Ss}	50		ns
t_{hST}	Hold time, status after $\overline{SPC}$↑	t_{Sh}	40		ns
t_{suD}	Setup time, data before $\overline{SPC}$↑	t_{Ds}	40		ns
t_{hD}	Hold time, data after $\overline{SPC}$	t_{Dh}	50		ns
t_{wSPC}	Pulse duration, $\overline{SPC}$ low, from CPU	t_{SPCw}	70		ns
t_{suSPC}	Setup time, $\overline{SPC}$ before CLK↑	t_{sPCs}	40		ns
t_{hSPC}	Hold time, $\overline{SPC}$ after CLK↑	t_{SPCh}	0		ns
t_{suRSTH}	Setup time, $\overline{RST}$ high before CLK↓	t_{RSTs}	10		ns
t_{hRSTL}	Hold time, $\overline{RST}$ low after CLK↑	t_{RSTh}	0		ns
t_{dSPC}	Delay time between $\overline{SPC}$ pulses during operand transfer			10	ms

switching characteristics over recommended operating free-air temperature range

PARAMETER		NSC SYMBOL	MIN	MAX	UNIT
t_{aD}	Access time, D0 through D15 after $\overline{SPC}\downarrow$	t_{Dv}	10	50	ns
t_{wSPC}	Pulse duration, $\overline{SPC}$ low from FPU	t_{SPCw}	$t_{cC}-50$	$t_{cC}+50$	ns
t_{disD}	Disable time, D0 through D15 after $\overline{SPC}\uparrow$	t_{Df}	0		ns
t_{pSPCL}	Propagation delay time, $\overline{SPC}$ high-to-low-level output after CLK$\uparrow$	t_{SPCF1}		70	ns
t_{pSPCH}	Propagation delay time, $\overline{SPC}$ low-to-high-level output after CLK$\uparrow$	t_{SPCFh}		70	ns
t_{pSPCHZ}	Propagation delay time, $\overline{SPC}$ high level to high impedance after CLK$\uparrow$	t_{SPCFnf}		45	ns

PARAMETER MEASUREMENT INFORMATION

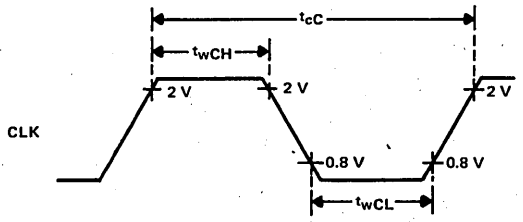


FIGURE 1. CLOCK TIMING

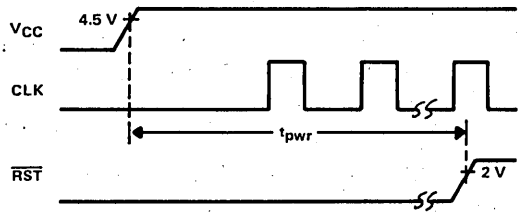


FIGURE 2. POWER-ON RESET

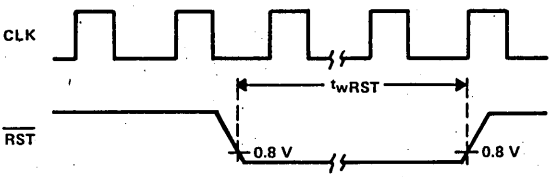


FIGURE 3. NONPOWER-ON RESET

PARAMETER MEASUREMENT INFORMATION

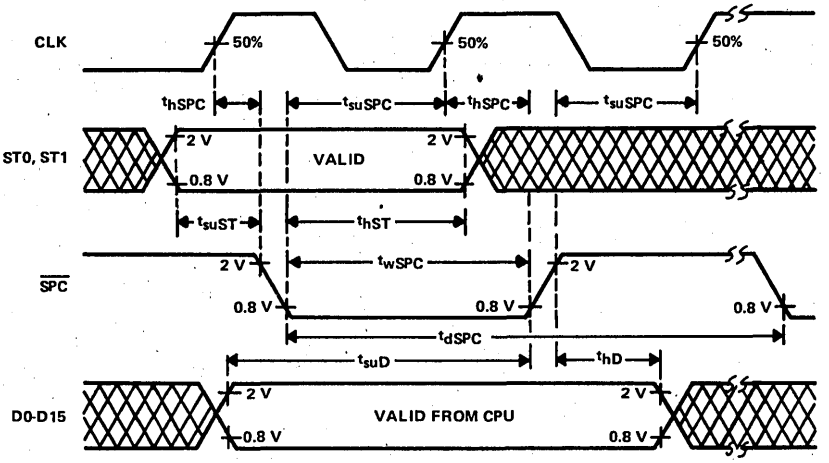


FIGURE 4. WRITE CYCLE TO FPU

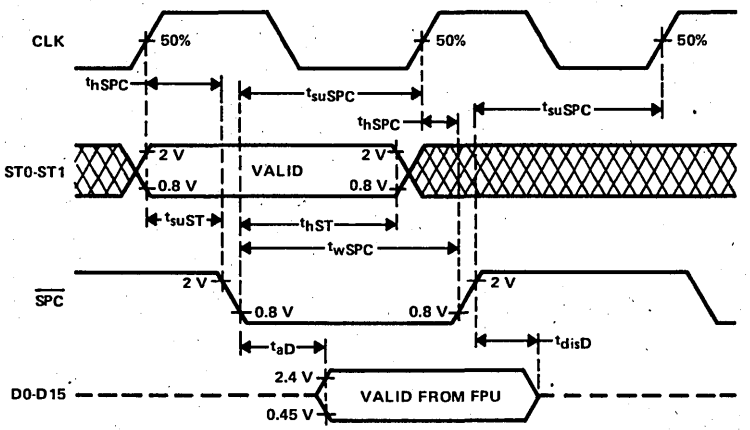


FIGURE 5. READ CYCLE FROM FPU

PARAMETER MEASUREMENT INFORMATION

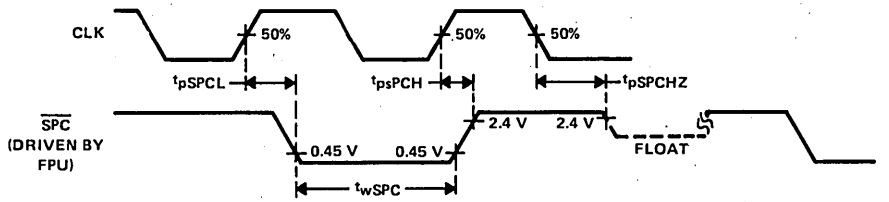


FIGURE 6. $\overline{SPC}$ PULSE FROM FPU.

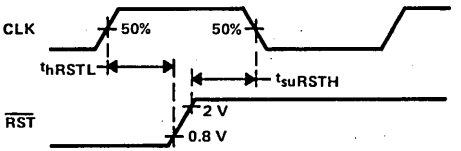


FIGURE 7. RESET RELEASE TIMING

TYPICAL APPLICATION DATA

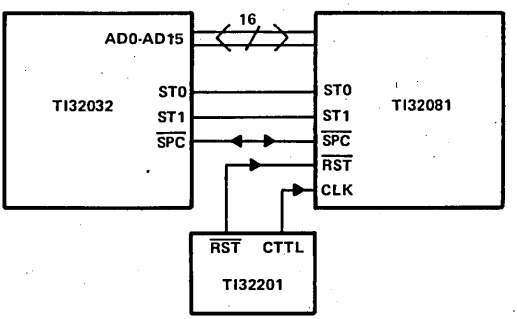
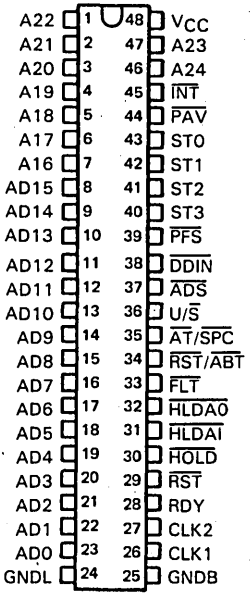


FIGURE 8. TYPICAL SYSTEM CONFIGURATION

D2877, APRIL 1985

- **High-Speed NMOS Technology**
- **Dynamic Address Translation**
- **32-Entry On-Chip Translation Cache, Updated Automatically from Tables in Memory**
- **Full Hardware Support for Demand-Paged Virtual Memory Management**
- **Security Mechanisms Implemented via Access-Level Checking and Dual-Space Mapping**
- **Program Debugging Support:**
  Two Breakpoint Registers
  "Break on Branch" Mode
- **Single 5-V Supply Operation**

JD DUAL-IN-LINE PACKAGE
(TOP VIEW)



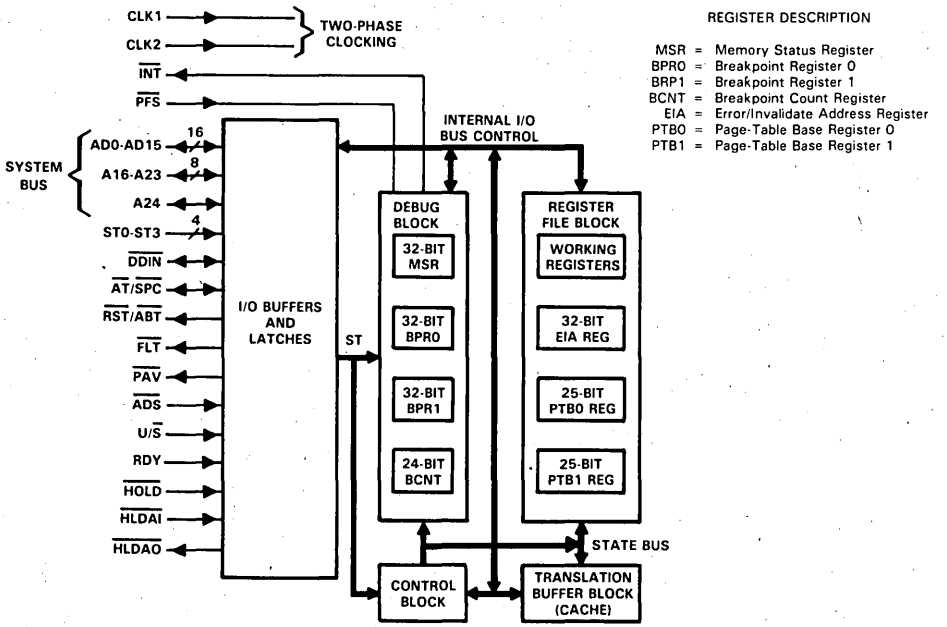
description

The TI32082W-2 Memory Management Unit (MMU) supports demand-page virtual memory management in systems using TI32016T and TI32032T CPU's. Specific capabilities include fast, dynamic address translation, protection on individual 512-byte pages, and status reporting to assist an operating system in efficiently managing up to 16-megabytes of physical memory. It provides support for virtual machine implementations and features comprehensive software debugging capabilities.

High-speed address translation is performed on-chip through a 32-entry associative cache memory that maintains itself (with no software intervention) from tables in memory. Protection violations and page faults (references to nonresident pages) are automatically detected by the MMU, invoking the abort instruction along with the retry feature of the CPU. This fault-handling mechanism provides the necessary support for virtual memory and virtual machines.

Additional program debugging features include two hardware breakpoint registers that provide the programmer with powerful stand-alone debugging capability even without expensive test equipment.

functional block diagram



PIN FUNCTIONAL DESCRIPTION TABLE

PIN			DESCRIPTION
NO.	NAME	I/O	
1	A22	I/O	Multiplexed logical/physical address bus: Eight-bit bus including A23 (pin 47) which holds high byte of 24-bit address. During T1 receives the logical address from the CPU. During TMMU-T4 delivers the physical address from the MMU.
2	A21	I/O	
3	A20	I/O	
4	A19	I/O	
5	A18	I/O	
6	A17	I/O	
7	A16	I/O	

PIN FUNCTIONAL DESCRIPTION (Continued)

PIN NO.	PIN NAME	PIN I/O	DESCRIPTION
8	AD15	I/O	
9	AD14	I/O	
10	AD13	I/O	
11	AD12	I/O	
12	AD11	I/O	
13	AD10	I/O	
14	AD9	I/O	Multiplexed I/O Data Bus: 16-bit bus for data transfer.
15	AD8	I/O	During clock period T1, receives the logical address from the CPU.
16	AD7	I/O	During clock period TMMU, delivers the physical address from the MMU.
17	AD7	I/O	During clock period T2-T4, carries data from the CPU, memory, or MMU.
18	AD5	I/O	
19	AD4	I/O	
20	AD3	I/O	
21	AD2	I/O	
22	AD1	I/O	
23	AD0	I/O	
24	GNDL	I	Logic ground. Ground reference for on-chip logic.
25	GNDB	I	Buffer ground. Ground reference for on-chip output.
26	CLK1	I	Clock 1. One of two 2-phase clocking signals with frequency of 0.5 to 10 MHz.
27	CLK2	I	Clock 2. One of two 2-phase clocking signals with frequency of 0.5 to 10 MHz.
28	RDY	I	Ready. Used by slow memories to extend memory cycle more than four clock periods. It is synchronized externally and sampled at the beginning of T3.
29	$\overline{RST}$	I	System reset, active low. Initiates a reset.
30	$\overline{HOLD}$	I	Hold request, active low. Used in DMA transfers.
31	$\overline{HLDAI}$	I	Hold acknowledge input: Active low. Used for DMA transfers. Originates from CPU $\overline{HLDA}$ output.
32	$\overline{HLDAO}$	O	Hold acknowledge output. Active low. Used for DMA transfers.
33	$\overline{FLT}$	O	Float: Active low output to CPU that floats CPU from the bus when the MMU requires bus access.
34	$\overline{RST/ABT}$	O	MMU reset or abort. Active low. Output accepted by CPU during T2 or TMMU.
35	$\overline{AT/SPC}$	I/O	Address translation/coprocessor control. Active low. Driven by the CPU as the data strobe for bus transfers to and from the MMU. Driven by the MMU to signal completion of an operation.
36	U/$\overline{S}$	I	Used by the MMU for memory protection and selection of the user/operating system mode.
37	$\overline{ADS}$	I	Address strobe. Active low input from CPU during T1 of CLK1 latches logical address.
38	$\overline{DDIN}$	I/O	Data direction in. As an input, a low indicates read and a high indicates write. When the CPU places this line in the high-impedance state, it permits this line to be driven by the MMU. This permits a read/write action into the memory page independent of the CPU.
39	$\overline{PFS}$	I	Program flow status. Active-low pulse issued by CPU at beginning of each instruction.
40	ST3	I	
41	ST2	I	Status (ST0, ST1, ST2, and ST3). Input from CPU that is monitored by the MMU to keep track of the status of the instruction being executed.
42	ST1	I	
43	ST0	I	
44	$\overline{PAV}$	O	Physical address valid. Active-low pulse generated during TMMU.
45	$\overline{INT}$	O	Interrupt Output. Active-low pulse. Informs CPU (when connected to its NMI input) or external hardware that a break condition has occurred.
46	A24	I/O	As an output, the most significant bit of the physical address, valid from clock period TMMU to T4. During reset, this pin should be held high to configure the MMU for use with the TI32016T or be held low to configure the MMU for use with the TI32032T.
47	A23	I/O	Address bit 23 of logical/physical address multiplexed bus (A16-A23).
48	V_{CC}	I	Supply voltage (5 V).

absolute maximum ratings over recommended operating conditions (unless otherwise noted)

Supply voltage, V_{CC} (see Note 1) . . . . . 7 V
Input voltage range . . . . . −0.5 V to 7 V
Output voltage range . . . . . −0.5 V to 7 V
Continuous total dissipation . . . . . 1.5 W
Operating free-air temperature range . . . . . 0°C to 70°C

NOTE 1: All voltage values are with respect to the common ground.

recommended operating conditions

		MIN	NOM	MAX	UNIT
Supply voltage, V_{CC}		4.75	5	5.25	V
High-level input voltage, V_{IH}	CLK1, CLK2 inputs	$V_{CC}-0.45$		5.25	V
	Any other input	2		$V_{CC}+0.5$	
Low-level input voltage, V_{IL}	CLK1, CLK2 inputs	−0.3		0.35	V
	Any other input	−0.5		0.8	
Operating free-air temperature, T_A		0		70	°C

electrical characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

PARAMETER		TEST CONDITIONS	MIN	MAX	UNIT
V_{OH}	High-level output voltage	I_{OH} = −400 μA	2.4		V
V_{OL}	Low-level output voltage	I_{OL} = 2 mA		0.45	V
I_I	Input current, other than clocks	V_I = 0 to V_{CC}		10	μA
$I_{O(off)}$	Off-state output current	V_O = 0.45 V to V_{CC}		10	μA
I_{CC}	Supply current	I_O = 0, T_A = 25°C		300	mA

Timing parameter symbols have been created in accordance with JEDEC Standard 100. In order to shorten the symbols, the pin names have been further abbreviated.

Pin name abbreviations are as follows:

$\overline{ABT}$	AB
$\overline{ADS}$	AS
CLK1	C or C1
CLK2	C2
Data	D
$\overline{DDIN}$	DD
$\overline{FLT}$	F
$\overline{HLDA}$	HA
$\overline{PAV}$	P
Physical Address (A0-A15)	A1
Physical Address (A16-A23)	A2
$\overline{RDY}$	R
$\overline{SPC}$	S

Lowercase subscripts and their meanings are:

a	access time
c	cycle time (period)
dis	disable time (H or L to Z)
en	enable time (Z to H or L)
f	fall time
p	propagation delay time
r	rise time
su	setup time
w	pulse duration

Parentheses have been omitted. The columns titled NSC SYMBOL show the symbols used by National Semiconductor for the parameters shown in the timing requirements and switching characteristics tables.

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

timing requirements over recommended operating free-air temperature range (unless otherwise noted)

PARAMETER		FROM	TO	NSC SYMBOL	FIGURE NO.	MIN	MAX	UNIT
f_{clock}	Clock frequency					5	10	MHz
t_{cC}	Clock period	50% CLK1↑	50% CLK1↓	t_{CP}	1	100	2000	ns
t_{wC1H}	CLK1 high pulse duration	50% CLK1↑	50% CLK1↓	t_{Ch}	1	$0.5t_{cC}-10$		ns
t_{wC2H}	CLK2 high pulse duration	50% CLK2↑	50% CLK2↓	t_{Ch}	1	$0.5t_{cC}-10$		ns
t_{wCL}	CLK1 or CLK2 low pulse duration	50% CLK1↓	50% CLK1↑	t_{Cl}	1	$0.5t_{cC}-10$		ns
	Clock pulse asymmetry, $t_{wC1H}-t_{wC2H}$				1		±4	ns
t_{C1LC2H}	Nonoverlap time of CLK1↓ to CLK2↑	10% CLK1↓	10% CLK2↑		1	0		ns
t_{C2LC1H}	Nonoverlap time of CLK2↓ to CLK1↑	10% CLK2↓	10% CLK		1	0		ns
t_{rC}	CLK1 or CLK2 rise time	10% CLK1↑	90% CLK1↑	t_{Cr}	1		9	ns
t_{fC}	CLK1 or CLK2 fall time	90% CLK1↓	10% CLK1↓	t_{Cf}	1		9	ns
t_{suR}	RDY setup time	0.8/2 V RDY	50% CLK1↑	t_{RDYs}	8,9	30		ns
t_{hR}	RDY hold time	50% CLK1↑	2 V RDY↓	t_{RDYh}	9	0		ns
t_{C1HSL}	CLK1 high to $\overline{SPC}$ low delay	50% CLK1↑	0.8 V $\overline{SPC}$↓	t_{SPCa}	10		35	ns
t_{C1HSH}	CLK1 high to $\overline{SPC}$ high delay	50% CLK1↑	2 V $\overline{SPC}$↑	t_{SPCia}	10		35	ns
t_{suD}	Read data (D0-D15) setup time	0.8/2 V D0-D15	50% CLK2↓	t_{Ds}	5,11	15		ns
t_{hD}	Read data (D0-D15) hold time	50% CLK1↑	0.8/2 V D0-D15	t_{Dh}, t_{D1h}	5,11	0		ns
t_{wASL}	$\overline{ADS}$ pulse duration	0.8 V $\overline{ADS}$↓	0.8 V $\overline{ADS}$↑		2,3	35		ns

4 Data Sheets

switching characteristics over recommended ranges of supply voltage and operating free-air temperature, C_L = 100 pF (unless otherwise noted)

	PARAMETER	FROM	TO	NSC SYMBOL	FIGURE NO.	MIN	MAX	UNIT
t_{pA1}	Physical address (A0-A15) delay	50% CLK1↑	0.8/2 V A0-A15	t_{Av}, t_{ALv}	2,3,4,5		50	ns
t_{pA2}	Physical address (A16-A23) delay	50% CLK1↑	0.8/2 V A16-A23	t_{Av}, t_{AHv}	2,3,4,5		50	ns
t_{disA}	Read address (A0-A15) disable time	50% CLK2↑	Hi-Z A0-A15	t_{Af}, t_{ALf}	2,3,5		25	ns
t_{pPL}	$\overline{PAV}$ high-to-low level delay	50% CLK1↑	0.8 V $\overline{PAV}$↓	t_{PAVa}	2,3,4,5		35	ns
t_{pPH}	$\overline{PAV}$ low-to-high level delay	50% CLK1↑	2 V $\overline{PAV}$↑	t_{PAVia}	2,3,4,5		95	ns
t_{wP}	$\overline{PAV}$ low pulse duration	0.8 V $\overline{PAV}$↓	0.8 V $\overline{PAV}$↑	t_{PAVw}	2,3,4,5	40		ns
t_{APH}	Physical address to $\overline{PAV}$ high delay	0.8/2 V A0-A23	2 V $\overline{PAV}$↑	t_{PAVs}	2,3,4,5	20		ns
t_{vPH-A}	Physical address valid after $\overline{PAV}$ high	2 V $\overline{PAV}$↑	0.8/2 V A0-A15	$t_{AvPAVia}$	2,3,4,5	10		ns
t_{enDD}	$\overline{DDIN}$ enable time	50% CLK↑	0.8/2 V $\overline{DDIN}$	t_{DDINa}	6		50	ns
t_{disDD}	$\overline{DDIN}$ disable time	50% CLK1↑	0.8/2 V $\overline{DDIN}$	t_{DDINia}	7		50	ns
t_{pD}	Write data delay	50% CLK1↑	0.8/2 V D0-D15	t_{Dv}	4,9		50	ns
t_{vD}	Write data (D0-D15) valid time	50% CLK2↓	0.8/2 V D0-D15	t_{Dh}	4,9	0		ns
t_{pFL}	CLK1 to $\overline{FLT}$ low delay	50% CLK1↑	0.8 V $\overline{FLT}$↓	t_{FLTa}	6		45	ns
t_{pFH}	CLK1 to $\overline{FLT}$ high delay	50% CLK1↑	2 V $\overline{FLT}$↑	t_{FLTia}	7		45	ns
t_{wF}	$\overline{FLT}$ low pulse duration	0.8 V $\overline{FLT}$↓	2 V $\overline{FLT}$ ↑	t_{FLTw}		40		ns
t_{pABL}	CLK1 to $\overline{ABT}$ low delay	50% CLK1↑	0.8 V $\overline{ABT}$↓	t_{ABTa}			55	ns
t_{pABH}	CLK1 to $\overline{ABT}$ high delay	50% CLK1↓	2 V $\overline{ABT}$↑	t_{ABTia}			45	ns
t_{wABL}	$\overline{ABT}$ low pulse duration	0.8 V $\overline{ABT}$↓	0.8 V $\overline{ABT}$↑	t_{ABTw}		70		ns
t_{pHA}	$\overline{HLDAI}$ to $\overline{HLDAO}$ delay	0.8 V $\overline{HLDAI}$↓	0.8 V $\overline{HLDAO}$↓	t_{HLDd}		30		ns

PARAMETER MEASUREMENT INFORMATION†

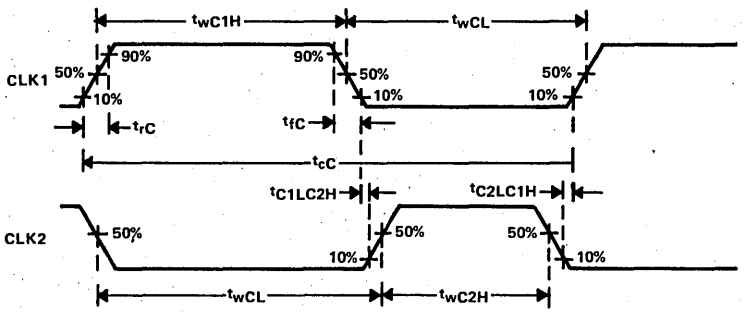


FIGURE 1. CLOCK CYCLE

†In Figures 2 through 11, time intervals are defined with respect to the following reference points:
For Clock signals, the 50% points.
For all other signals, 2 V if the high level is indicated and 0.8 V if the low level is indicated.

PARAMETER MEASUREMENT INFORMATION

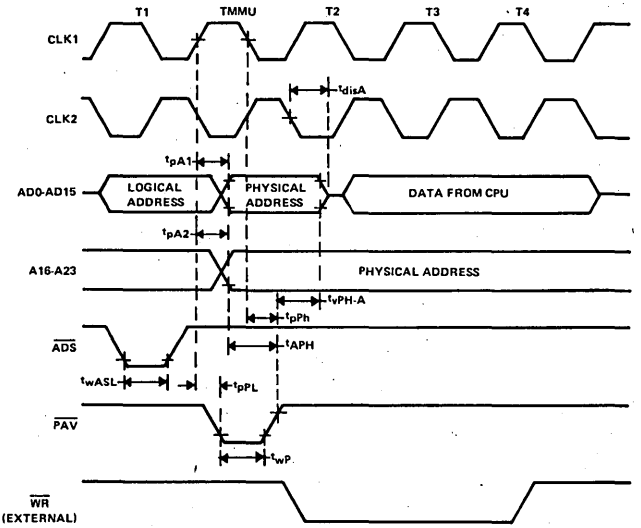


FIGURE 2. MEMORY WRITE CYCLE WITH LOGICAL ADDRESS IN TRANSLATION BUFFER

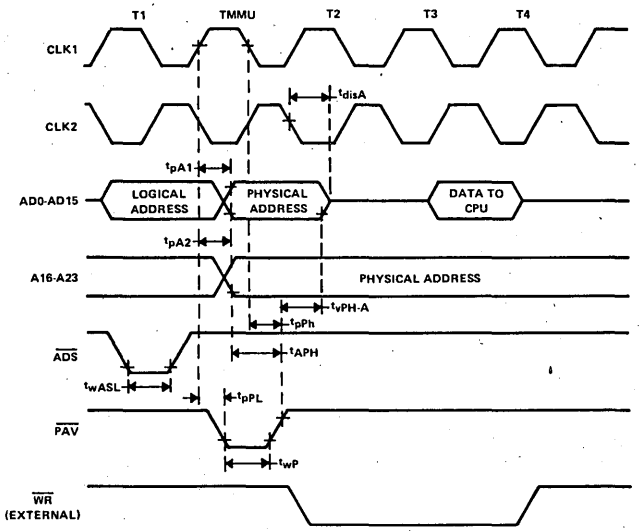


FIGURE 3. MEMORY READ CYCLE WITH LOGICAL ADDRESS IN TRANSLATION BUFFER

PARAMETER MEASUREMENT INFORMATION

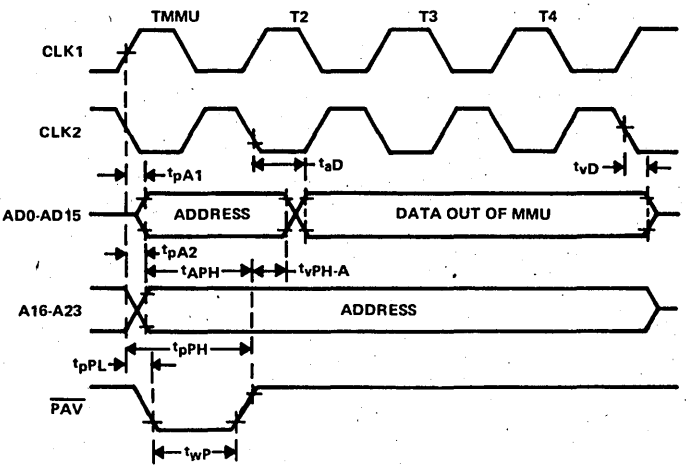


FIGURE 4. MEMORY MANAGEMENT UNIT WRITE CYCLE

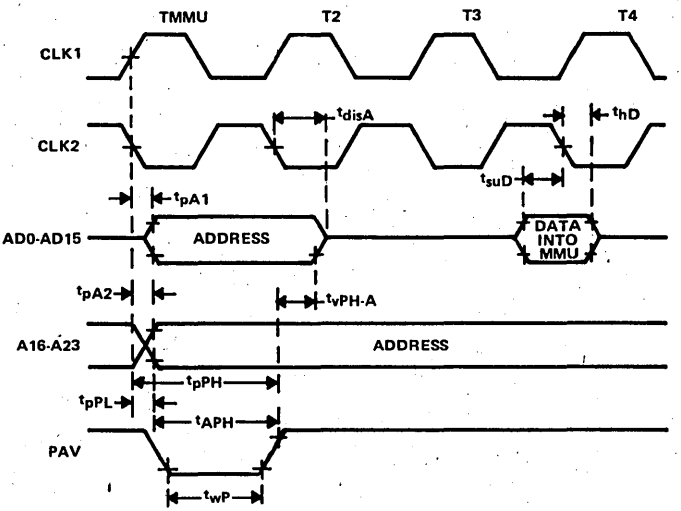


FIGURE 5. MEMORY MANAGEMENT UNIT READ CYCLE

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

PARAMETER MEASUREMENT INFORMATION

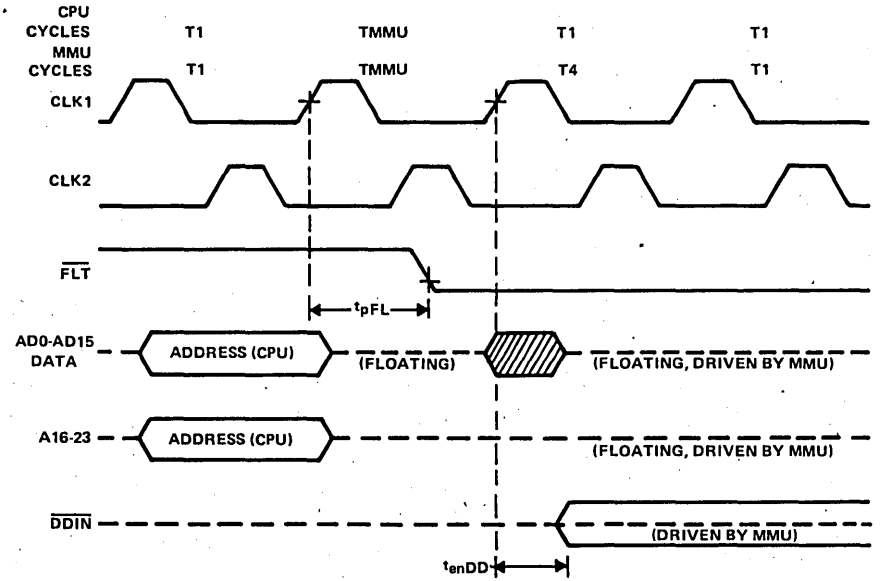


FIGURE 6. $\overline{FLT}$ INITIATED FLOAT CYCLE TIMING

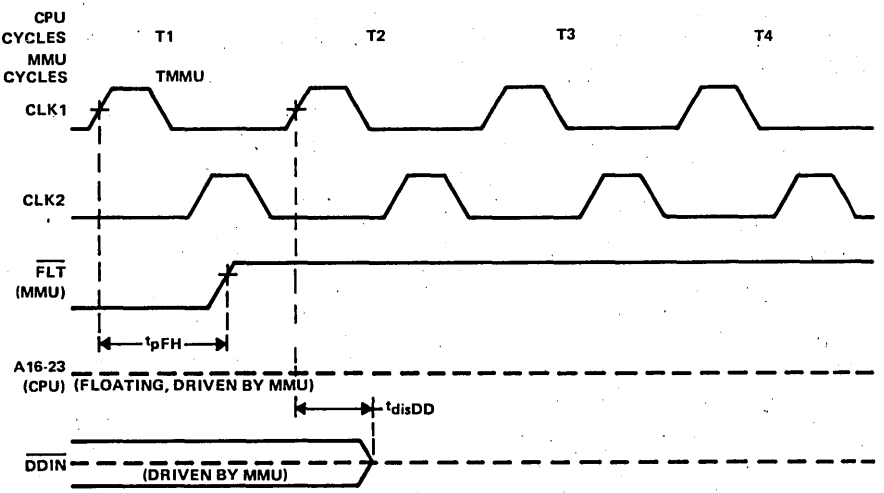


FIGURE 7. RELEASE FROM $\overline{FLT}$ TIMING

PARAMETER MEASUREMENT INFORMATION

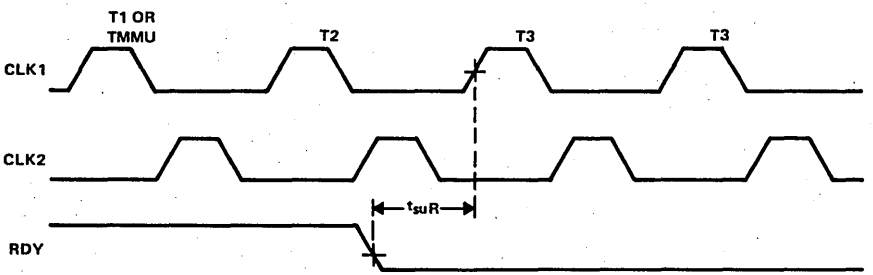


FIGURE 8. READY SAMPLING (MMU INITIALLY READY)

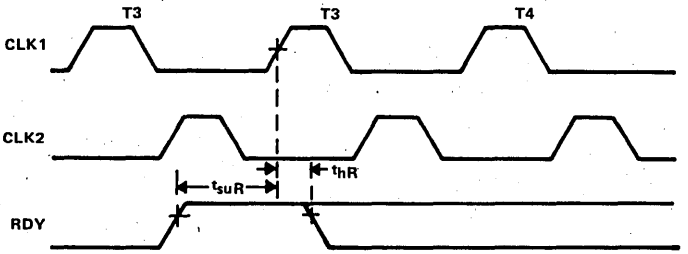
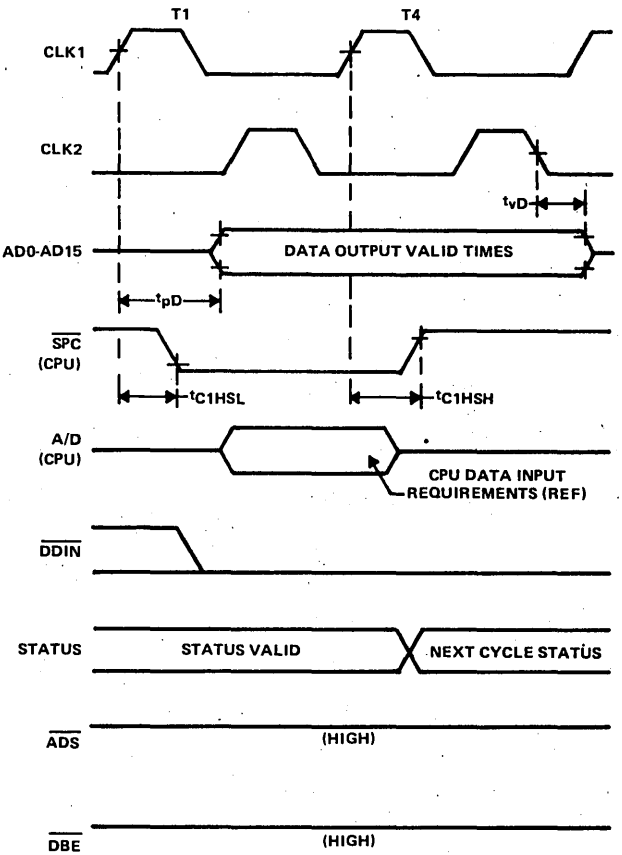


FIGURE 9. READY SAMPLING (MMU INITIALLY NOT READY)

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

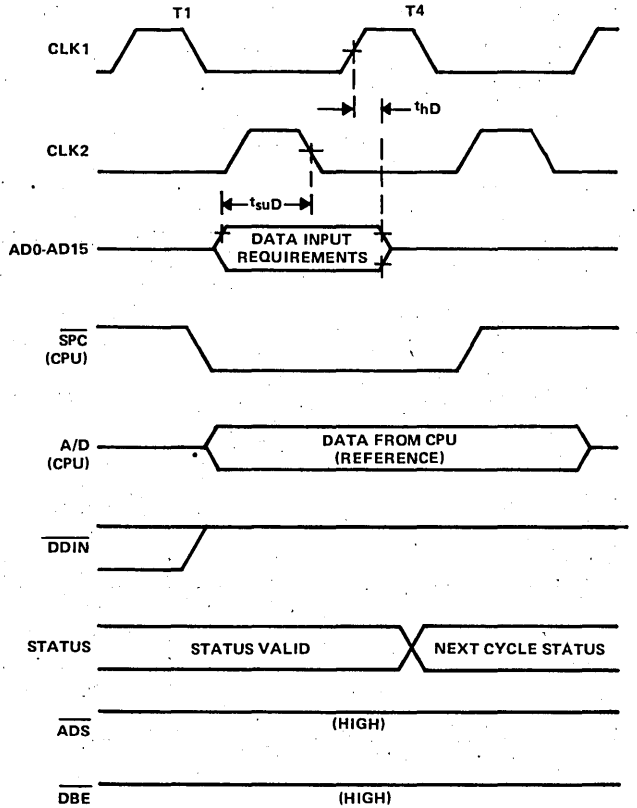
PARAMETER MEASUREMENT INFORMATION



NOTE: CPU timing shown for reference, for actual CPU setup and hold times see CPU data sheets.

FIGURE 10. MMU DATA OUT TIMING (COPROCESOR INSTRUCTIONS)

TYPICAL APPLICATION DATA



NOTE: CPU timing shown for reference, for actual CPU setup and hold times see CPU data sheets.

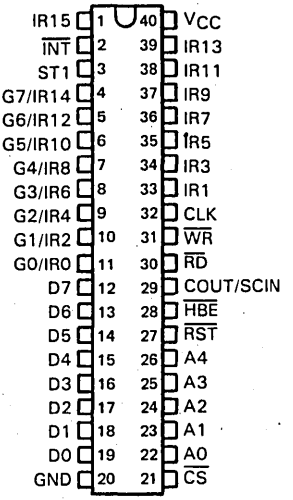
FIGURE 11. MMU DATA IN TIMING (COPROCESSOR INSTRUCTIONS)

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

D2874, APRIL 1985

- High-Speed NMOS Technology
- 16 Maskable Interrupt Sources (can be Cascaded to 256)
- Programmable 8- or 16-Bit Data Bus
- Edge or Level Triggering for Each Hardware Interrupt with Individually Selectable Polarities
- 8 Software Interrupts
- Fixed or Rotating Modes
- Dual DC to 10 MHz, 16-Bit Counters that may be Concatenated to a Single 32-Bit Counter
- Optional 8-Bit I/O Port Available in 8-Bit Data Bus Mode
- Single 5-V Supply Operation
- Designed to be Interchangeable with National Semiconductor NS16202

JD DUAL-IN-LINE PACKAGE
(TOP VIEW)



description

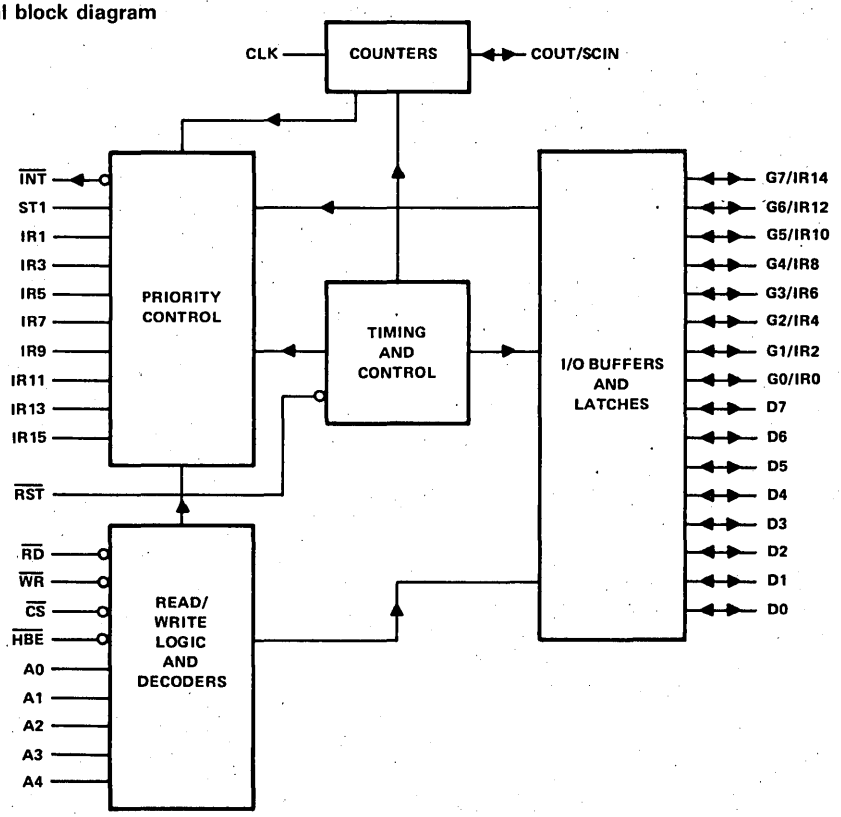
The TI32202 Interrupt Control Unit (ICU) is designed for use with the TI32000 microprocessor family. Its usage minimizes the software and real-time overhead needed to handle multilevel, prioritized interrupts. One TI32202 can manage up to 16 interrupt sources, resolve interrupt priorities, and supply a single-byte interrupt vector to the CPU.

The TI32202 operates in either the 8-bit or 16-bit data bus modes. In the 8-bit mode, up to 16 hardware interrupts with programmable priorities are handled. In the 16-bit mode, 8 hardware and 8 software interrupts can be handled. In either mode, up to 16 additional ICUs may be cascaded to handle a maximum of 256 interrupts.

Two 16-bit counters, which may be concatenated under program control into a single counter, are also available for real-time applications.

The TI3202 is characterized for operation from 0°C to 70°C.

functional block diagram



PIN FUNCTIONAL DESCRIPTION TABLE

PIN NO.	PIN NAME	I/O	DESCRIPTION
1	IR15	IN	Interrupt Request — One of eight inputs used for hardware interrupts. May be individually triggered in one of four modes (see explanation for pins 33 thru 39).
2	$\overline{INT}$	OUT	Interrupt Output — This signal, when low, indicates that an interrupt is pending.
3	ST1	IN	Status — Status signal from the CPU. When the Hardware Vector Register is read, this signal differentiates an INTA (Interrupt Acknowledge) cycle from an RETI (Return-from-Interrupt) cycle. If ST1 is low, the ICU initiates an INTA cycle. If ST1 is high, an RETI cycle will result.
4 5 6 7 8 9 10 11	G7/IR14 G6/IR12 G5/IR10 G4/IR8 G3/IR6 G2/IR4 G1/IR2 G0/IR0	I/O	General Purpose I/O Lines — These pins are the high-order data bits when the ICU is in the 16-bit bus mode. When the ICU is in the 8-bit bus mode, each pin may be individually assigned one of the following functions: 1. Additional Hardware Interrupt Input (IR0 through IR14) 2. General Purpose Data Input 3. General Purpose Data Output 4. Clock Output from H-Counter (pins G0/IR0 through G3/IR6 only)

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

PIN FUNCTIONAL DESCRIPTION TABLE (Continued)

PIN NO.	PIN NAME	I/O	DESCRIPTION
12	D7	I/O	Data Bus Lines — Eight low-order data bus lines used in both 8-bit and 16-bit bus modes
13	D6	I/O	
14	D5	I/O	
15	D4	I/O	
16	D3	I/O	
17	D2	I/O	
18	D1	I/O	
19	D0	I/O	
20	GND		Ground — Power supply return
21	$\overline{CS}$	IN	Chip Select — When $\overline{CS}$ is high (inactive), it disables the ICU from responding to address, data, and control signals.
22	A0	IN	Address Lines — These five address lines are used to select the ICU internal registers for read and write operations
23	A1	IN	
24	A2	IN	
25	A3	IN	
26	A4	IN	
27	$\overline{RST}$	IN	Reset — When low, this signal initializes the ICU. The ICU initializes to the 8-bit bus mode.
28	HBE	IN	High-Byte Enable — When low, this input data transfers on the most significant byte of the Data Bus. If the ICU is in the 8-bit bus mode, this signal is not used and the terminal should be connected to either GND or V_{CC}.
29	COUT/SCIN	I/O	Counter or Oscillator Output/Sampling Clock Input — An output signal from this pin provides either a clock signal generated by the ICU internal oscillator, or a zero-detect signal from one or both of the ICU counters. An input to this pin is used for an external clock to override the ICU internal oscillator. The internal oscillator is overridden during sampling. This is done only for testing purposes.
30	$\overline{RD}$	IN	Read — When low, this signal enables data to be read from internal registers of the ICU.
31	$\overline{WR}$	IN	Write — When low, this signal enables data to be written into internal registers of the ICU.
32	CLK	IN	Counter Clock — External clock signal used to drive internal counters of the ICU.
33	IR1	IN	Interrupt Request Lines — These seven lines along with IR15 (pin 1) constitute eight inputs used for hardware interrupts. Each of the eight inputs may be triggered in one of the following modes: Rising Edge Falling Edge Low Level High Level
34	IR3	IN	
35	IR5	IN	
36	IR7	IN	
37	IR9	IN	
38	IR11	IN	
39	IR13	IN	
40	V_{CC}		Power Supply, 5 V dc

absolute maximum ratings over recommended operating conditions (unless otherwise noted)

Supply voltage, V_{CC} (see Note 1) . . . . . 7 V
Input voltage range . . . . . −0.5 V to 7 V
Output voltage range . . . . . −0.5 V to 7 V
Continuous total dissipation . . . . . 1.5 W
Operating free-air temperature range . . . . . 0°C to 70°C
Storage temperature range . . . . . −65°C to 150°C
Lead temperature 1,6 mm (1/16 inch) from case for 60 seconds . . . . . 300°C

NOTE 1: All voltage values are with respect to the GND terminal.

recommended operating conditions

	MIN	NOM	MAX	UNIT
Supply voltage, V_{CC}	4.75	5	5.25	V
High-level input voltage, V_{IH}	2			V
Low-level input voltage, V_{IL}			0.8	V
Operating free-air temperature, T_A	0		70	°C

electrical characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
V_{OH}	High-level output voltage	I_{OH} = −400 μA	2.4			V
V_{OL}	Low-level output voltage	I_{OL} = 2 mA			0.45	V
I_I	Input current	V_I = 0 to V_{CC}			±20	μA
$I_{O(off)}$	Off-state output current	V_O = 0.4 V to V_{CC}			±20	μA
I_{CC}	Supply current	I_O = 0, T_A = 0°C			300	mA

Timing parameter symbols have been created in accordance with JEDEC Standard 100. In order to shorten the symbols, the pin names have been further abbreviated as follows:

Pin Name abbreviations are as follows:

CLK	C or CL
COUT	CO
$\overline{CS}$	CS
$\overline{INT}$	INT
$\overline{IR}$	I
$\overline{RD}$	RD
$\overline{RST}$	RST
ST1	S
$\overline{WR}$	WR

Lowercase subscripts and their meanings are:

a	access time
c	cycle time (period)
d	delay time interval
dis	disable time (H or L to Z)
en	enable time (Z to H or L)
f	fall time
h	hold time
p	propagation delay time
r	rise time
su	setup time
w	pulse duration

Parentheses have been omitted except when required for separation of a final subscript representing special conditions. The columns titled NSC SYMBOL show the symbols used by National Semiconductor for the parameters shown in the timing requirements and switching characteristics tables.

timing requirements over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

PARAMETER		FROM	TO	NSC SYMBOL	FIGURE NO.	MIN	MAX	UNIT
$t_{hA(R)}$	Read address hold time	2 V $\overline{RD}\uparrow$	0.8/2 V Address	t_{AhRDia}	1	80		ns
$t_{suA(R)}$	Read address setup time	0.8/2 V Address	2 V $\overline{RD}\downarrow$	t_{AsRDa}	1	50		ns
$t_{hCS(R)}$	Read $\overline{CS}$ hold time	2 V $\overline{RD}\uparrow$	0.8 V $\overline{CS}\downarrow$	$t_{CShRDia}$	1	80		ns
$t_{suCS(R)}$	Read $\overline{CS}$ setup time	0.8 V $\overline{CS}\downarrow$	2 V $\overline{RD}\downarrow$	t_{CSsRDa}	1	50		ns
$t_{hD(R)}$	Read data hold time	2 V $\overline{RD}\uparrow$	0.8/2 V Data	t_{DhRDia}	1	0	50	ns
t_{wRD}	$\overline{RD}$ low pulse duration	0.8 V $\overline{RD}\downarrow$	0.8 V $\overline{RD}\uparrow$	t_{RDw}	1	160		ns
t_{suS}	ST1 setup time	0.8/2 V ST1	2 V $\overline{RD}\downarrow$	t_{SsRDa}	1	50		ns
t_{hS}	ST1 hold time	2 V $\overline{RD}\uparrow$	0.8/2 V ST1	t_{ShRDia}	1	−30		ns
$t_{hA(W)}$	Write address hold time	2 V $\overline{WR}\uparrow$	0.8/2 V Address	t_{AhWRia}	2	80		ns
$t_{suA(W)}$	Write address setup time	0.8/2 V Address	2 V $\overline{WR}\downarrow$	t_{AsWRa}	2	50		ns
$t_{hCS(W)}$	Write $\overline{CS}$ hold time	2 V $\overline{WR}\uparrow$	0.8 V CS↑	$t_{CShWRia}$	2	80		ns
$t_{suCS(W)}$	Write $\overline{CS}$ setup time	0.8 V CS↓	2 V $\overline{WR}\downarrow$	t_{CSsWRa}	2	50		ns
$t_{hD(W)}$	Write data hold time	2 V $\overline{WR}\uparrow$	0.8/2 V Data	t_{DhWRia}	2	50		ns
$t_{suD(W)}$	Write data setup time	0.8/2 V Data	0.8 V $\overline{WR}\uparrow$	t_{DsWRia}	2	100		ns
t_{wWR}	$\overline{WR}$ low pulse duration	0.8 V $\overline{WR}\downarrow$	0.8 V $\overline{WR}\uparrow$	t_{WRw}	2	160		ns
t_{dIAI}	Interrupt acknowledge to IR delay	2 V $\overline{RD}$ (INTA)↓	0.8/2 V IR↓	t_{ACKiR}	3	500		ns
$t_{wI(ET)}$	IR low pulse duration in edge-trigger mode	0.8/2 V IR↓	0.8/2 V IR↑	t_{IRw}	3	50		ns
t_{wSCH}	Sampling clock high pulse duration	2 V SCIN↑	2 V SCIN↓	t_{SCINh}	5	100		ns
t_{wSCL}	Sampling clock low pulse duration	0.8 V SCIN↓	0.8 V SCIN↑	t_{SCINl}	5	100		ns
t_{cSC}	Sampling clock period	0.8 V SCIN↓	0.8 V SCIN↓	t_{SCINp}	5	800		ns
t_{cC}	Clock period	0.8 V CLK↓	0.8 V CLK↓	t_{Cy}	7	400		ns
$t_{cC(P)}$	Clock period with prescaler	0.8 V CLK↓	0.8 V CLK↓	t_{Cyp}	7	100		ns
t_{wCH}	CLK high pulse duration	2 V CLK↑	2 V CLK↓	t_{Ch}	7	100		ns
$t_{wCH(P)}$	CLK high pulse duration (with prescaler)	2 V CLK↑	2 V CLK↓	t_{Chp}	7	50		ns
t_{wCL}	CLK low pulse duration	0.8 V CLK↓	0.8 V CLK↑	t_{Cl}	7	100		ns
$t_{wCL(P)}$	CLK low pulse duration (with prescaler)	0.8 V CLK↓	0.8 V CLK↑	t_{Clp}	7	50		ns
t_{wRST}	$\overline{RST}$ low pulse duration	0.8 V $\overline{RST}\downarrow$	0.8 V $\overline{RST}\uparrow$	t_{RSTw}		400		ns

switching characteristics over recommended supply voltage and operating free-air temperature, C_L = 100 pF (unless otherwise noted)

PARAMETER		FROM	TO	NSC SYMBOL	FIGURE NO.	MIN	MAX	UNIT
t_{aRD}	Data access time	0.8 V $\overline{RD}$↓	0.8/2 V Data	t_{RDaDv}	1		150	ns
t_{disG}	G0...G7 disable time	2 V $\overline{WR}$↓	Hi-Z G0...G7	t_{WRiaPf}			200	ns
t_{pG}	G0...G7 output delay time	2 V $\overline{WR}$↑	0.8/2 V G0...G7	t_{WRiaPv}	2		200	ns
t_{pINT}	$\overline{INT}$ low delay after IR	0.8/2 V IR↓	0.8 V INT↓	t_{IRid}	3, 4		800	ns
t_{wCOL}	COUT low pulse duration	0.8 V COUT↓	0.8 V COUT↑	t_{COUTl} t_{COUTw}	6, 7	50		ns
t_{cCO}	COUT clock period	0.8 V COUT↓	0.8 V COUT↓	t_{COUTp}	6	400		ns
t_{CLCO}	CLK↓ to COUT delay	0.8 V CLK↓	0.8 V COUT↓	t_{GCOUTl}	7		300	ns

PARAMETER MEASUREMENT INFORMATION

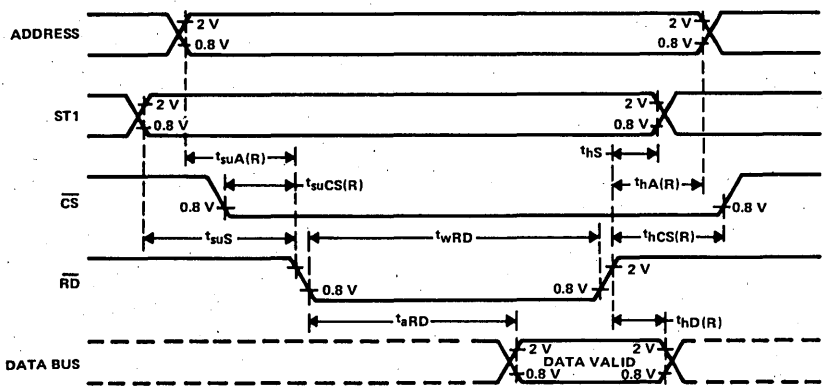


Figure 1. READ/INTA Cycle

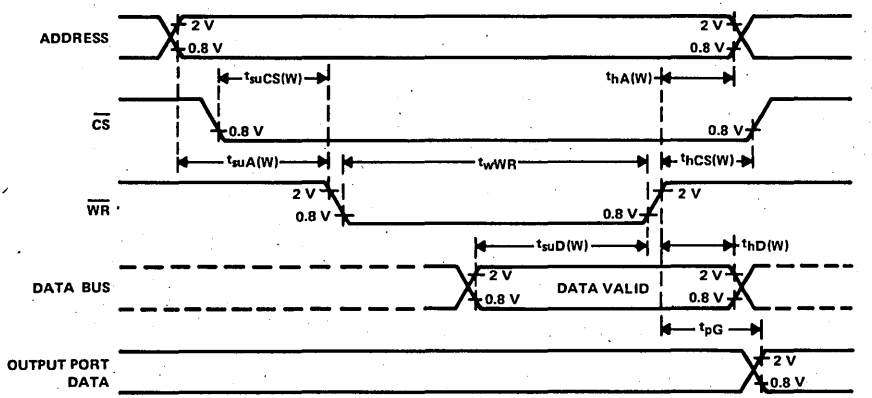


Figure 2. Write Cycle

PARAMETER MEASUREMENT INFORMATION

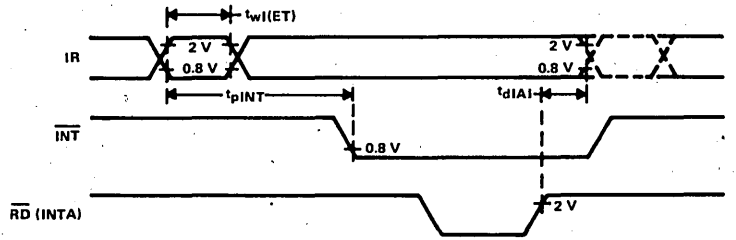


Figure 3. Interrupt Timing in Edge-Triggering Mode

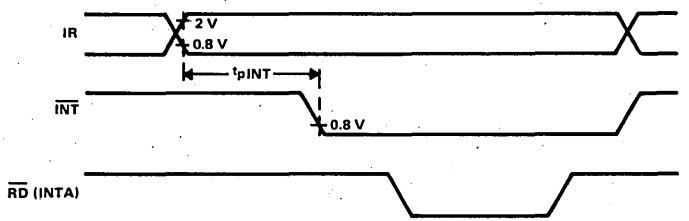


Figure 4. Interrupt Timing In Level Triggering Mode

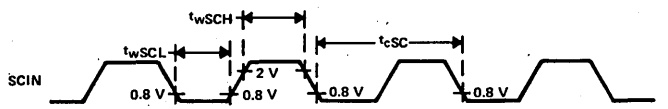


Figure 5. External Interrupt Sampling Clock to be Provided at COUT/SCIN When in Test Mode

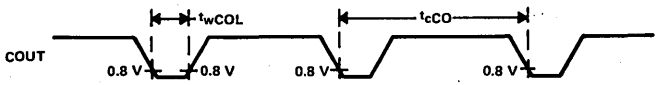


Figure 6. Internal Interrupt Sampling Clock Provided at COUT/SCIN

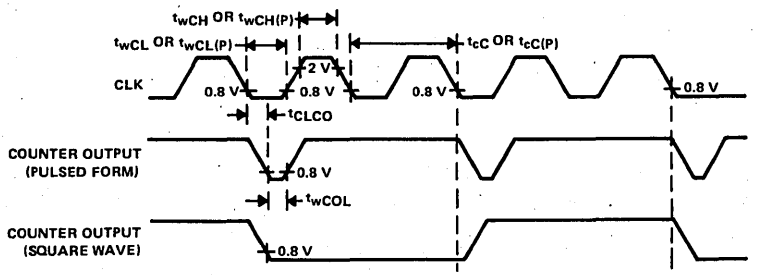
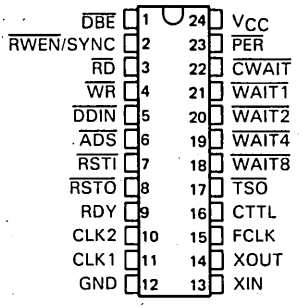


Figure 7. Relationslhip Between Clock Input at CLK and Counter Output Signals at COUT/SCIN or G0/R0,...G3/R6, Pulsed Form and Square Wave

D2870, APRIL 1985

- 2-Phase Full V_{CC} Swing High-Capacitance Drivers
- 4-Bit Input ($\overline{WAITn}$) Allowing Precise Specification of 0 to 15 Wait States
- Cycle Hold for System Arbitration and/or Memory Refresh
- System Timing (CTTL and FCLK) and Control ($\overline{RD}$, $\overline{WR}$, and $\overline{DBE}$) Outputs
- General-Purpose Timing-State Output to Identify Internal States
- Support of Slow MOS Peripheral Interface ICs (e.g., 8080 Series)
- Provides Ready (RDY) Output for TI32000 MPUs
- Synchronous System Reset Generation from Schmitt Trigger Input
- Single 5-V Power Supply

JD DUAL-IN-LINE PACKAGE
(TOP VIEW)



description

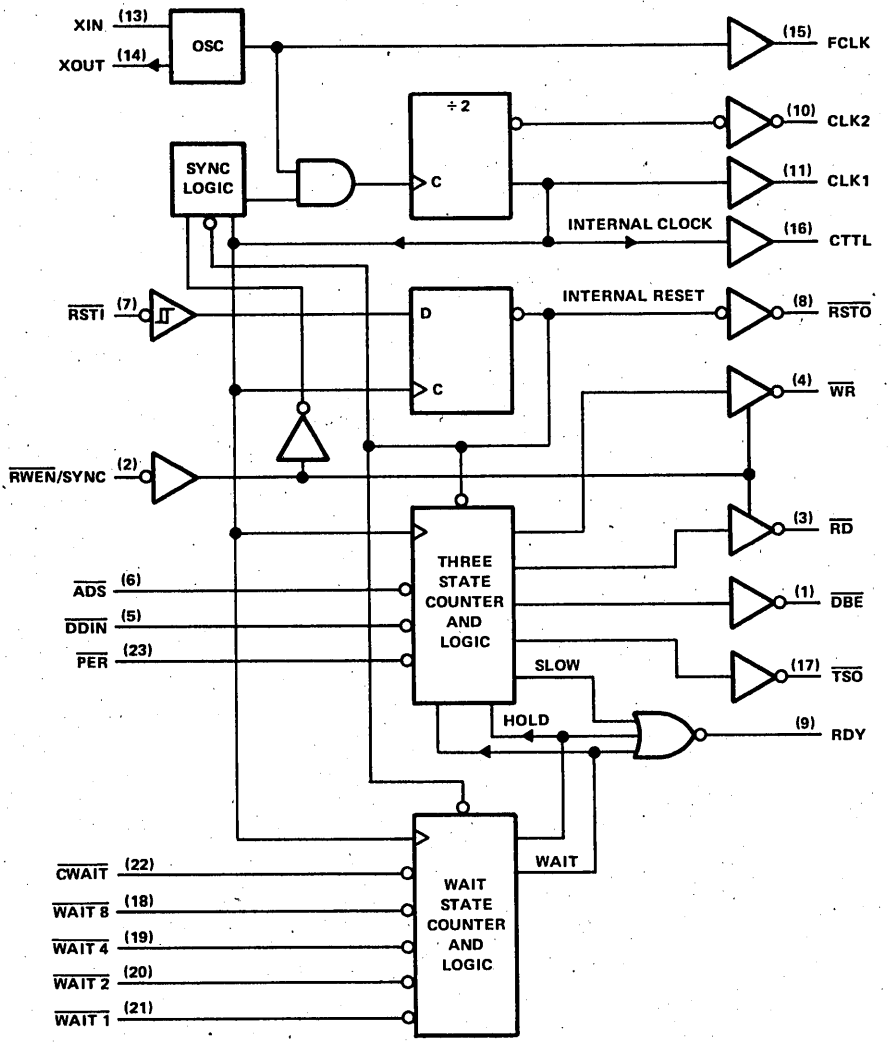
The TI32201 Timing Control Unit (TCU) is a 24-pin device fabricated by a Schottky bipolar process. It provides the 2-phase MOS clock drivers, system control logic (read, write, and data buffer enable), and cycle extension logic for the TI32000 microprocessor family.

A crystal or external signal is used as the frequency source. In addition to the two-phase MPU clock outputs (CLK1 and CLK2), two other clock outputs (TTL-compatible) are available for system timing. One of these is a fast clock (FCLK) providing twice the MPU clock frequency (i.e., at the crystal frequency). Clock CTTL is a TTL version of CLK1.

The TCU also features cycle extension that includes digitally programmable wait state inputs, a peripheral (slow) cycle to accommodate slower MOS peripheral interface, and Cycle Hold between the first cycle time (T1) and second cycle time (T2) to allow for arbitration prior to generating control signals.

The TI32201 is characterized for operation from 0°C to 70°C.

functional block diagram



PIN FUNCTIONAL DESCRIPTION

PIN NO.	PIN NAME	I/O	DESCRIPTION
1	$\overline{DBE}$	OUT	Data Buffer Enable Output — A low signal at this pin enables the 3-state buffers on the data lines of the TI32000 CPU system.
2	$\overline{RWEN}$/SYNC	IN	Read-Write Enable/Synchronizing Input — This input when low enables the $\overline{RD}$ and $\overline{WR}$ outputs. When high, it synchronizes the CLK1 and FCLK through the sync logic circuitry and disables the $\overline{RD}$ and $\overline{WR}$ outputs placing the lines in the high-impedance state.
3	$\overline{RD}$	OUT	Read Output Strobe (3-state) — When low, this signal identifies a READ cycle. It is decoded from $\overline{DDIN}$ and placed in the high-impedance state by $\overline{RWEN}$ going high.
4	$\overline{WR}$	OUT	Write Output Strobe (3-state) — When low, this signal identifies a WRITE cycle. It is decoded from $\overline{DDIN}$ and placed in the high-impedance state by $\overline{RWEN}$ going high.
5	$\overline{DDIN}$	IN	Data Direction Input — When low, this signal determines that a READ cycle is performed. When high, a WRITE cycle is performed.
6	$\overline{ADS}$	IN	Address Strobe Input from CPU/MMU — This signal (going low) identifies the first timing state (T1) of a bus cycle for the TCU.
7	$\overline{RSTI}$	IN	Reset Input — This Schmitt trigger input generates the synchronous reset signal $\overline{RSTO}$. The rising edge of $\overline{RSTO}$ is synchronized to CLK1. $\overline{RSTI}$ going low causes $\overline{RSTO}$ to go low, and this signals a system reset.
8	$\overline{RSTO}$	OUT	Reset Output — The signal $\overline{RSTI}$ going low causes $\overline{RSTO}$ to bo low. When the slow rising edge of $\overline{RSTI}$ reaches the threshold of the internal Schmitt trigger, $\overline{RSTO}$ goes high on the next rising edge of CLK1.
9	RDY	OUT	Ready Output — This signal remains low as long as WAIT states are to be inserted in a bus cycle (cycle extension). This pin is normally connected to the RDY input of the CPU.
10 11	CLK1 CLK2	OUT OUT	CPU Clocks — These outputs provide 2-phase nonoverlapping clock signals. Their frequency is half that of the crystal source or external source. They are intended to drive high-capacitance MOS inputs with a full V_{CC} swing.
12	GND		Common reference for power supply and all signals.
13	XIN	IN	Crystal or External Frequency Source (XIN) — The desired CPU clock (CLK1 or CLK2) frequency will be half that of the crystal or external source. A Schottky series gate (V_{OH}min = 2.7 V) is recommended to drive XIN ($V_{threshold}$ = 2.5 V) as the external frequency source.
14	XOUT	OUT	Crystal Feedback Output — This output is used in crystal operation only. This pin must be left open when driving XIN with an external frequency source.
15	FCLK	OUT	Fast Clock Output — This is a TTL-level clock output having the same frequency as the crystal or external frequency source. Therefore, its frequency is twice that of the CPU clocks.
16	CTTL	OUT	TTL System Clock — This output is a TTL version of CLK1; therefore, it operates at the CPU clock frequency.
17	$\overline{TSO}$	OUT	Timing State Output — The falling edge signals the beginning of state T2. The rising edge signals the start of state T4 of the bus cycle.
18 19 20 21	$\overline{WAIT1}$ $\overline{WAIT2}$ $\overline{WAIT4}$ $\overline{WAIT8}$	IN IN IN IN	Wait-State Inputs — These inputs, collectively called $\overline{WAITn}$, allow from 0 through 15 wait states to be specified. They have binary weights of 1, 2, 4, and 8, respectively. The number of wait states will be equal to the sum of the weights of the wait-state inputs standing at the low level.
22	$\overline{CWAIT}$	IN	Continuous Wait Input — This input is used to initiate the Cycle Hold (sampled at end of T1) or (continuous) Wait-State (sampled in the middle of T2) modes. In the Wait-State mode, this input causes the $\overline{WAITn}$ inputs to be sampled and possibly to be overridden.
23	$\overline{PER}$	IN	Peripheral Cycle Input — When this signal is low, the TCU performs a slow cycle. This input causes the TCU to insert 5 WAIT states into a normal bus cycle and reshape the $\overline{RD}$ and $\overline{WR}$ signals. This satisfies the $\overline{RD}$ and $\overline{WR}$ setup and hold times required by slower peripherals.
24	V_{CC}	IN	+5-Volt power supply input

absolute maximum ratings over recommended operating conditions (unless otherwise noted)

Supply voltage, V_{CC} (see Note 1) . . . . . 7 V
Input voltage range . . . . . −1 V to 5.5 V
Output voltage range . . . . . −1 V to 5.5 V
Continuous total dissipation . . . . . 1.5 W
Operating free-air temperature range . . . . . 0°C to 70°C
Storage temperature range . . . . . −65°C to 150°C

NOTE 1: All voltage values are with respect to the common ground.

recommended operating conditions

	MIN	NOM	MAX	UNIT
Supply voltage, V_{CC}	4.75	5	5.25	V
High-level input voltage, V_{IH}, any input except $\overline{RSTI}$ and XIN	2			V
Low-level input voltage, V_{IL}, any input except $\overline{RSTI}$ and XIN			0.8	V
Operating free-air temperature, T_A	0		70	°C

electrical characteristics over recommended ranges of supply voltages and operating free-air temperature (unless otherwise noted)

PARAMETER			TEST CONDITIONS	MIN	TYP†	MAX	UNIT
V_{IK}	Input clamp voltage	Any input except XIN	I_I = −18 mA		−0.7	−1.2	V
V_{hys}	Hysteresis ($V_{T+} - V_{T-}$)	$\overline{RSTI}$ input		0.1V_{CC}	0.2V_{CC}	0.25V_{CC}	
V_{T+}	Positive-going input threshold voltage	$\overline{RSTI}$ input		0.5V_{CC}	0.6V_{CC}	0.7V_{CC}	
V_T	Threshold voltage	XIN input		0.4V_{CC}	0.5V_{CC}	0.6V_{CC}	
V_{OH}	High-level output voltage	CLK1 and CLK2	I_{OH} = −1 mA	4.3			V
		Any other output except XOUT		2.4			
V_{OL}	Low-level output voltage	CLK1 and CLK2	I_{OL} = 1 mA			0.5	V
		Any other output except XOUT	I_{OL} = 20 mA			0.5	
I_{IH}	High-level input current	Any input except XIN	V_I = 5.25 V			50	μA
I_{IL}	Low-level input current	Any input except XIN	V_I = 0.5 V			−500	μA
I_{CC}	Supply current		All outputs high		180	260	mA

†All typical values are at V_{CC} = 5 V, T_A = 25°C.

Timing parameter symbols have been created in accordance with JEDEC Standard 100. In order to shorten the symbols, the pin names have been further abbreviated as follows:

CLK1	C or C1	$\overline{RD}$	R
CLK2	C2	$\overline{RSTO}$	RS
$\overline{CTTL}$	CT	SYNC	SN
$\overline{CWAIT}$	CW	$\overline{TSO}$	T
$\overline{DBE}$	DB	$\overline{WAITn}$	Wn
$\overline{DDIN}$	DD	$\overline{WR}$	W
FCLK	F	XIN	X

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

Lowercase subscripts and their meanings are:

c	cycle time (period)
dis	disable time (H or L to Z)
en	enable time (Z to H or L)
f	fall time
p	propagation delay time
r	rise time
su	setup time
w	pulse duration

Parentheses have been omitted except when required for separation of a final subscript representing special conditions. The columns titled NSC SYMBOL show the symbols used by National Semiconductor for the parameters shown in the timing requirements and switching characteristics tables.

timing requirements over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

PARAMETER		FROM	TO	NSC SYMBOL	FIGURE NO.	MIN	MAX	UNIT
t_{cC}	CLK1 clock period	50% CLK1↑	50% CLK1↑	t_{Cp}	4	100		ns
t_{wXH}	XIN applied high pulse duration	2.5 V XIN↑	2.5 V XIN↓	t_{Xh}	4	16		ns
t_{wXL}	XIN applied low pulse duration	2.5 V XIN↓	2.5 V XIN↑	t_{Xl}	4	16		ns
t_{suADS}	$\overline{ADS}$ setup time	0.8 V $\overline{ADS}$↓	50% CLK1↑	t_{ADs}	6	25		ns
t_{wADS}	$\overline{ADS}$ low pulse duration	0.8 V $\overline{ADS}$↓	0.8 V $\overline{ADS}$↑	t_{ADw}	6	25		ns
t_{suDD}	$\overline{DDIN}$ setup time	0.8/2 V $\overline{DDIN}$	50% CLK1↑	t_{DDs}	6	10		ns
$t_{suCW(CH)}$	$\overline{CWAIT}$ setup time (cycle hold)	0.8 V $\overline{CWAIT}$↓	50% CLK1↑	$t_{CWs(H)}$	10	25		ns
$t_{hCW(CH)}$	$\overline{CWAIT}$ hold time (cycle hold)	50% CLK1↑	0.8 V $\overline{CWAIT}$↑ or 2 V $\overline{CWAIT}$↓	$t_{CWh(H)}$	10	0		ns
$t_{suCW(WS)}$	$\overline{CWAIT}$ setup time (wait state)	0.8 V $\overline{CWAIT}$↓	50% CLK2↑	$t_{CWs(W)}$	10,11	13		ns
$t_{hCW(WS)}$	$\overline{CWAIT}$ hold time (wait state)	50% CLK2↑	0.8 V $\overline{CWAIT}$↑	$t_{CWh(W)}$	11	8		ns
t_{suWn}	$\overline{WAITn}$ setup time	0.8 V $\overline{WAITn}$↓	50% CLK2↑	t_{Ws}	11,12	5		ns
t_{hWn}	$\overline{WAITn}$ hold time	50% CLK2↑	0.8 V $\overline{WAITn}$↑	t_{Wh}	11,12	15		ns
t_{suPER}	$\overline{PER}$ setup time	0.8 V $\overline{PER}$↓	50% CLK1↑	t_{Ps}	12	0		ns
t_{hPER}	$\overline{PER}$ hold time	50% CLK1↑	0.8 V $\overline{PER}$↑	t_{Ph}	12	20		ns
t_{suSN}	SYNC setup time	2 V SYNC↑	0.8 V FCLK↑	t_{Sys}	13	18		ns
t_{hSN}	SYNC hold time	2 V FCLK↑	2 V SYNC↓	t_{Syh}	13	0		ns

switching characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted) (see Note 2)

clocks (XIN, FCLK, CLK1, and CLK2)

PARAMETER		FROM	TO	NSC SYMBOL	FIGURE NO.	MIN	MAX	UNIT
t_{CH}	CLK1 high time	90% CLK1↑	90% CLK1↓	t_{CLh}	4	0.5 t_{cC} − 15	0.5 t_{cC} − 7	ns
t_{CL}	CLK1 low time	10% CLK1↓	10% CLK1↑	t_{CLl}	4	0.5 t_{cC}	0.5 t_{cC} + 10	ns
t_{wCH}	CLK1 high pulse duration	50% CLK1↑	50% CLK1↓	$t_{CLw(1)}$	4	0.5 t_{cC} − 10	0.5 t_{cC} − 4	ns
t_{wC2H}	CLK2 high pulse duration	50% CLK2↑	50% CLK2↓	$t_{CLw(2)}$	4	0.5 t_{cC} − 10	0.5 t_{cC} − 4	ns
	Clock pulse asymmetry ($t_{wC1H} - t_{wC2H}$)			t_{CLwas}	4		±5	ns
t_{rC}	CLK1 rise time	10% CLK1↑	90% CLK1↑	t_{CLR}	4	2	7	ns
t_{fC}	CLK1 fall time	90% CLK1↓	10% CLK1↓	t_{CLF}	4	2	7	ns
t_{C1LC2H}	CLK1 low to CLK2 high nonoverlap	10% CLK1↓	10% CLK2↑	$t_{nOVL(1)}$	4		±5	ns
t_{C2LC1H}	CLK2 low to CLK1 high nonoverlap	10% CLK2↓	10% CLK1↑	$t_{nOVL(2)}$	4		±5	ns
	Nonoverlap asymmetry ($t_{C1LC2H} - t_{C1LC1H}$)			t_{nOVLas}	4		±4	ns
t_{XFH}	XIN high to FCLK high delay	2.5 V XIN↑	2 V FCLK↑	t_{XFr}	4	15	27	ns
t_{XFL}	XIN low to FCLK low delay	2.5 V XIN↓	0.8 V FCLK↓	t_{XFf}	4	15	27	ns
t_{XCTH}	XIN high to CTTL high delay	2.5 V XIN↑	2 V CTTL↑	t_{XCr}	4	24	35	ns
t_{XCH}	XIN high to CLK1 high delay	2.5 V XIN↑	50% CLK1↑	t_{XPr}	4	21	32	ns
t_{FCTH}	FCLK to CTTL high differential delay	2 V FCLK↑	2 V CTTL↑	t_{FCr}	4	5	15	ns
t_{FCTL}	FCLK to CTTL low differential delay	2 V FCLK↑	0.8 V CTTL↓	t_{FCf}	4	5	15	ns
t_{FCH}	FCLK to CLK1 high differential delay	2 V FCLK↑	50% CLK1↑	t_{FPr}	5	2	10	ns
t_{FCL}	FCLK to CLK1 low differential delay	2 V FCLK↑	50% CLK1↓	t_{FPf}	5		±4	ns
t_{FH}	FCLK high time	2 V FCLK↑	2 V FCLK↓	t_{Fw}	5	0.25 t_{cC} − 5	0.25 t_{cC} + 5	ns
t_{C2CT}	CLK2 to CTTL differential delay	50% CLK2↑	2 V CTTL↓	t_{PCf}	5		+10 −6	ns
t_{CTH}	CTTL high time	2 V CTTL↑	2 V CTTL↓	t_{CTw}	5	0.5 t_{cC} − 7	0.5 t_{cC} + 7	ns

NOTE 2: Unless otherwise noted, parameter measurements are taken with the output pins in the following conditions:
CLK1 and CLK2 — Figure 1.
All TTL outputs (except CTTL) C_L = 50 pF — Figure 2.
CTTL C_L = 100 pF — Figure 2.
$\overline{RD}$ and $\overline{WR}$ — Figure 3.

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

switching characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted) (see Note 2)

clock (CTTL) timing, C_L = 50 pF

PARAMETER		FROM	TO	NSC SYMBOL	FIGURE NO.	MIN	MAX	UNIT
t_{CCT}	CLK1 to CTTL differential delay	50% CLK1↑	2 V CTTL↑	t_{PCr}	5		+5 −2	ns
t_{rCT}	CTTL rise time	0.8 V CTTL↑	2 V CTTL↑	t_{CTR}	5		5	ns
t_{fCT}	CTTL fall time	2 V CTTL↓	0.8 V CTTL↓	t_{CTF}	5		4	ns

clock (CTTL) timing, C_L = 100 pF

PARAMETER		FROM	TO	NSC SYMBOL	FIGURE NO.	MIN	MAX	UNIT
t_{CCT}	CLK1 to CTTL differential delay	50% CLK1↑	2 V CTTL↑	t_{PCr}	5		+6 −2	ns
t_{rCT}	CTTL rise time	0.8 V CTTL↑	2 V CTTL↑	t_{CTR}	5		7	ns
t_{fCT}	CTTL fall time	2 V CTTL↓	0.8 V CTTL↓	t_{CTF}	5		5	ns

output controls ($\overline{RSTO}$, $\overline{TSO}$, $\overline{WR}$, $\overline{RD}$, RDY, and $\overline{RWEN}$/SYNC)

PARAMETER		FROM	TO	NSC SYMBOL	FIGURE NO.	MIN	MAX	UNIT
t_{pRS}	CLK1 to $\overline{RSTO}$ high delay	50% CLK1↑	2 V $\overline{RSTO}$↑	t_{RSTr}	6		15	ns
t_{pTL}	CLK1 to $\overline{TSO}$ low delay	50% CLK1↑	0.8 V $\overline{TSO}$↓	t_{Tf}	7,8		10	ns
t_{pTH}	CLK1 to $\overline{TSO}$ high delay	50% CLK1↑	2 V $\overline{TSO}$↑	t_{Tr}	7,8	5	15	ns
$t_{pWL(F)}$	CLK1 to $\overline{WR}$ low delay (Fast cycle)	50% CLK1↑	0.8 V $\overline{WR}$↓	$t_{RWf(F)}$	7	14	30	ns
$t_{pRL(F)}$	CLK1 to $\overline{RD}$ low delay	50% CLK1↑	0.8 V $\overline{RD}$↓	$t_{RWf(F)}$	7	14	30	ns
$t_{pWL(S)}$	CLK1 to $\overline{WR}$ low delay (Slow cycle)	50% CLK1↑	0.8 V $\overline{WR}$↓	$t_{RWf(S)}$	8	3	15	ns
$t_{pRL(S)}$	CLK1 to $\overline{RD}$ low delay	50% CLK1↑	0.8 V $\overline{RD}$↓	$t_{RWf(S)}$	8	3	15	ns
t_{pWH}	CLK1 to $\overline{WR}$ delay	50% CLK1↑	2 V $\overline{WR}$↑	t_{RWr}	7,8	7	20	ns
t_{pRH}	CLK1 to $\overline{RD}$ delay	50% CLK1↑	2 V $\overline{RD}$↑	t_{RWr}	7,8	7	20	ns
$t_{pDBL(W)}$	CLK1 to $\overline{DBE}$ delay (Write cycle)	50% CLK1↑	0.8 V $\overline{DBE}$↓	$t_{DBf(W)}$	7,8	8	24	ns
$t_{pDBL(R)}$	CLK1 to $\overline{DBE}$ delay (Read cycle)	50% CLK2↑	0.8 V $\overline{DBE}$↓	$t_{DBf(R)}$	7,8	3	15	ns
t_{pDBH}	CLK2 to $\overline{DBE}$ delay	50% CLK2↑	2 V $\overline{DBE}$↑	t_{DBr}	7,8	5	20	ns
$t_{dis(W/R)}$	$\overline{WR}$ or $\overline{RD}$ output disable time	2 V $\overline{RWEN}$↑	0.5 V change $\overline{WR}$ or $\overline{RD}$	t_{pLZ}, t_{pHZ}	9		20	ns
$t_{en(W/R)}$	$\overline{WR}$ or $\overline{RD}$ output enable time	0.8 V $\overline{RWEN}$↓	0.8 / 2 V $\overline{WR}$ or $\overline{RD}$	t_{pZL}, t_{pZH}	9		20	ns
t_{C2RDY}	CLK2 to RDY delay	50% CLK2↓	0.8/2 V RDY	t_{Rd}	10,11,12		23	ns
t_{CTSN}	CTTL to SYNC delay	0.8 V CTTL↓ (MASTER)	2 V SYNC↑ (SLAVE)	t_{CS}	13		15	ns

NOTE 2: Unless otherwise noted, parameter measurements are taken with the output pins in the following conditions:
CLK1 and CLK2 — Figure 1.
All TTL outputs (except CTTL) C_L = 50 pF — Figure 2.
CTTL C_L = 100 pF — Figure 2.
$\overline{RD}$ and $\overline{WR}$ — Figure 3.

PARAMETER MEASUREMENT INFORMATION

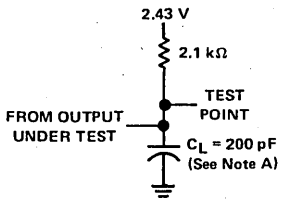


FIGURE 1. CLK1 AND CLK2 OUTPUT LOAD CIRCUIT

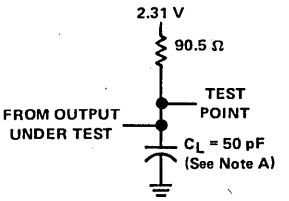


FIGURE 2. TTL OUTPUT LOAD CIRCUIT

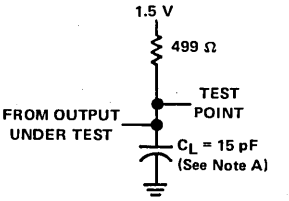


FIGURE 3. 3-STATE OUTPUT LOAD CIRCUIT

NOTE A: C_L includes probe and jig capacitance.

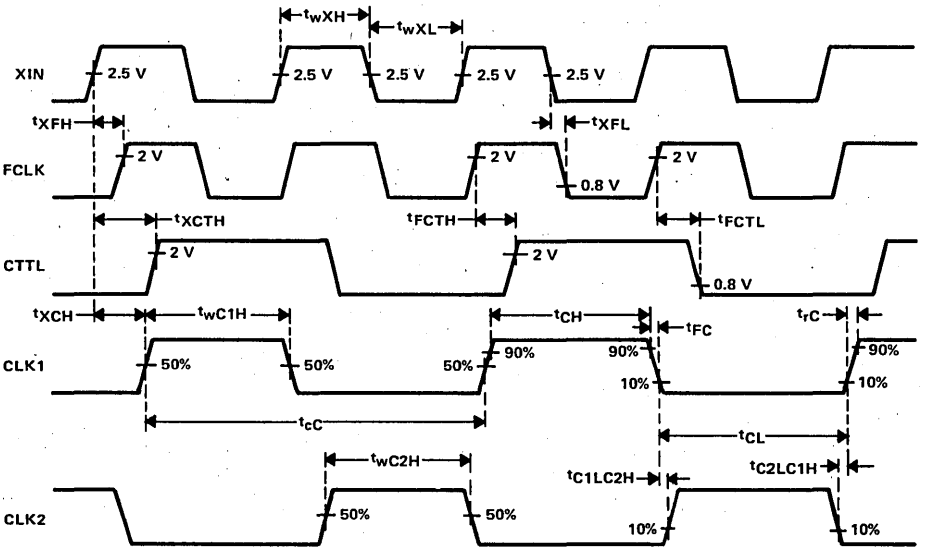


FIGURE 4. CLOCK SIGNALS (A)

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

PARAMETER MEASUREMENT INFORMATION

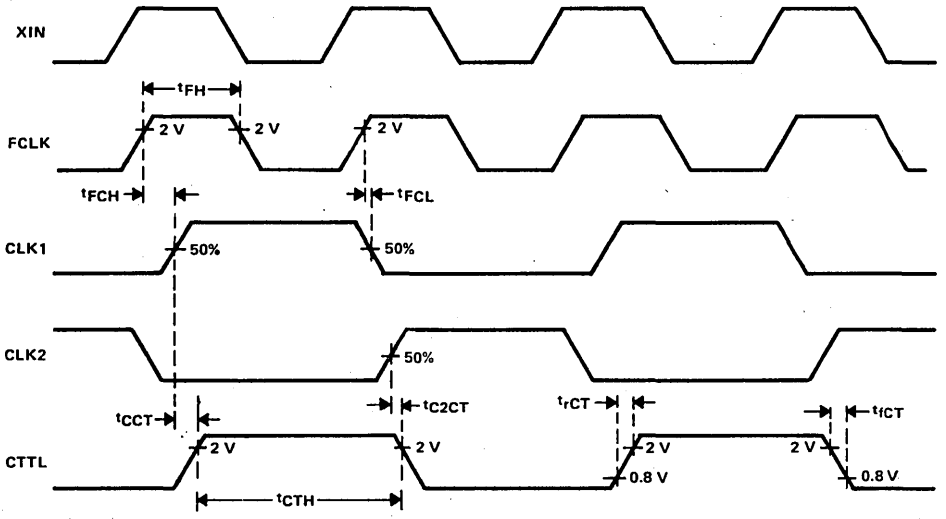


FIGURE 5. CLOCK SIGNALS (B)

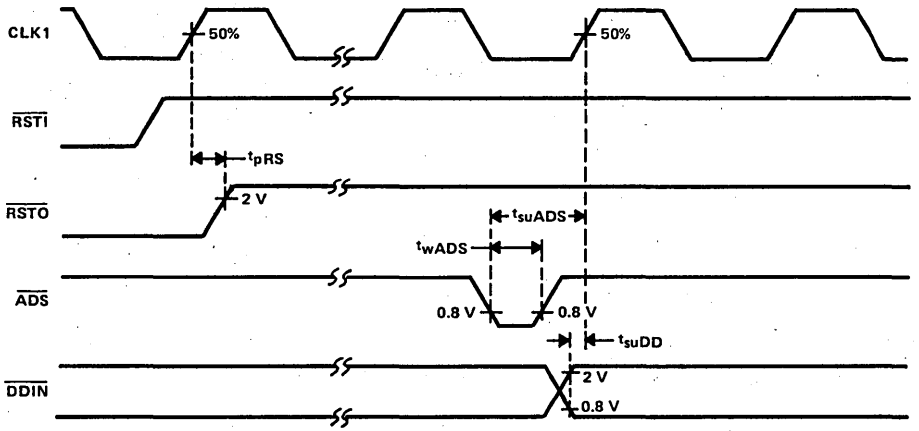


FIGURE 6. CONTROL INPUTS

PARAMETER MEASUREMENT INFORMATION

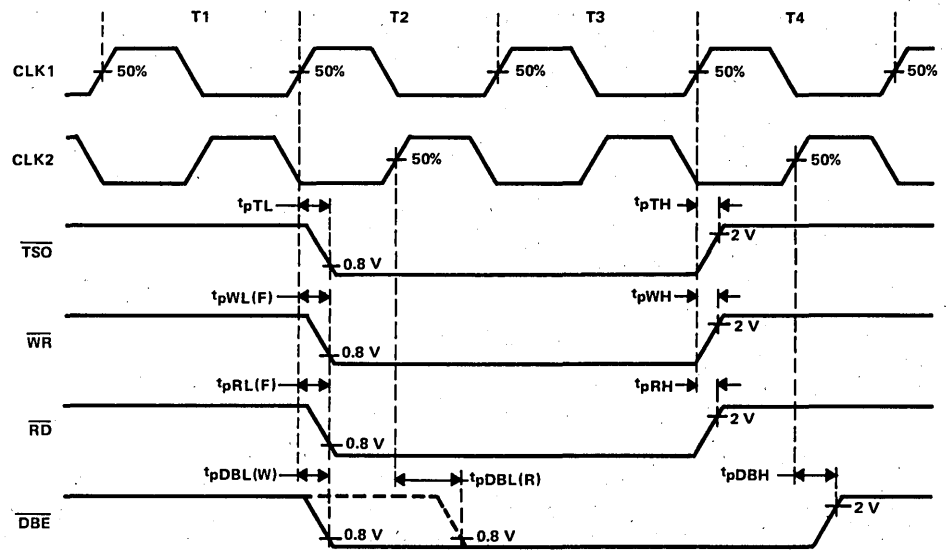


FIGURE 7. CONTROL OUTPUTS (FAST CYCLE)

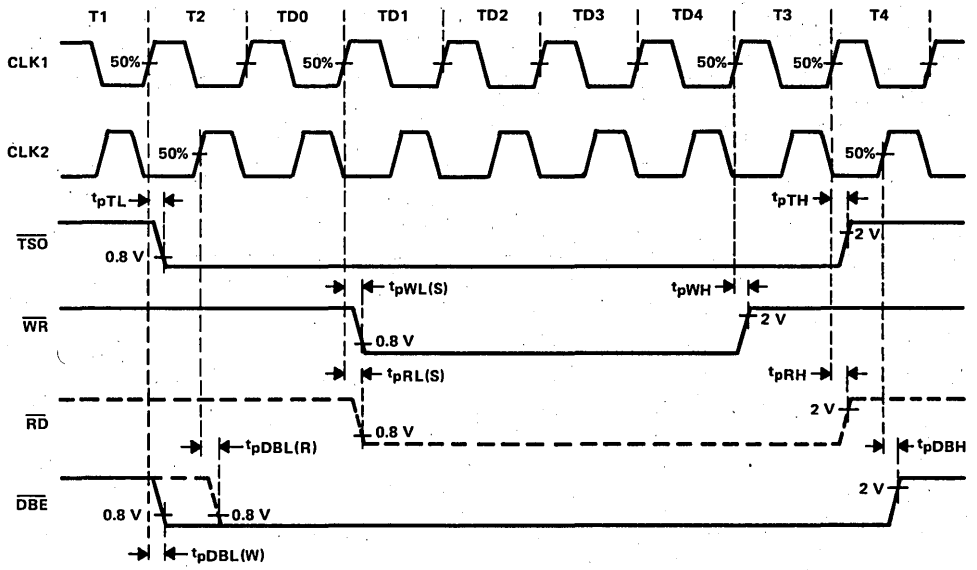


FIGURE 8. CONTROL OUTPUTS (PERIPHERAL CYCLE)

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

PARAMETER MEASUREMENT INFORMATION

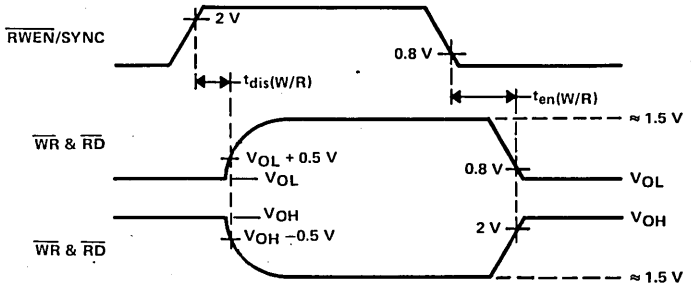


FIGURE 9. CONTROL OUTPUTS (THREE-STATE TIMING)

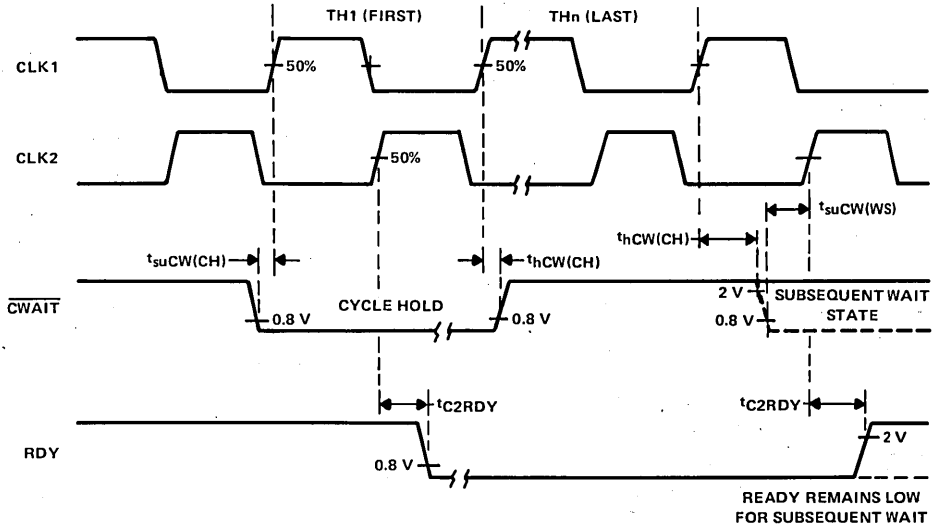


FIGURE 10. CYCLE HOLD

PARAMETER MEASUREMENT INFORMATION

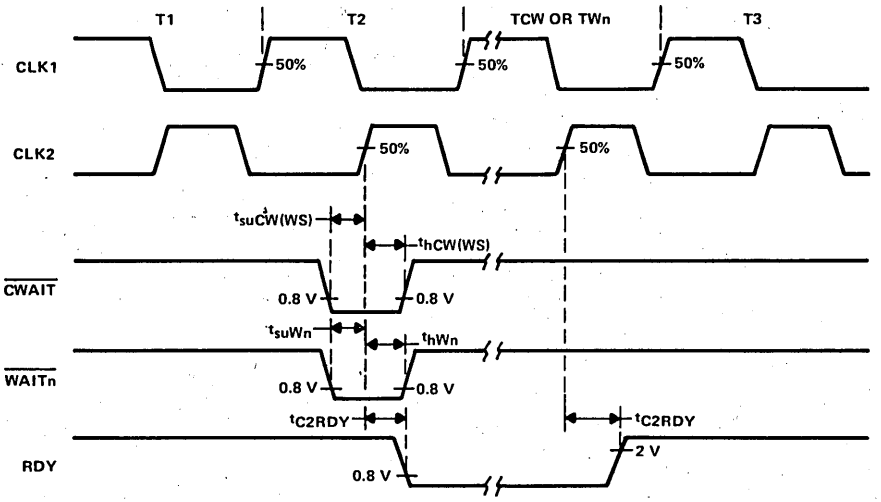


FIGURE 11. WAIT STATE (FAST CYCLE)

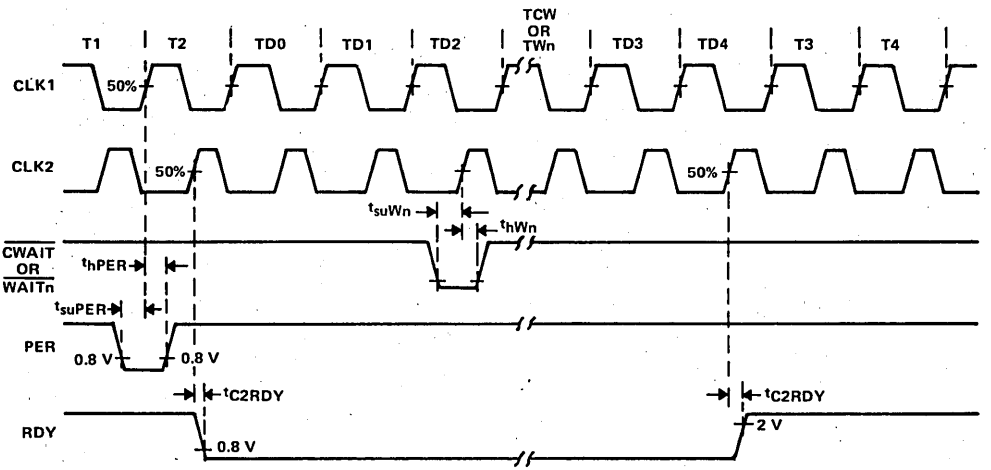


FIGURE 12. WAIT-STATE (PERIPHERAL CYCLE)

TEXAS INSTRUMENTS
POST OFFICE BOX 225012 • DALLAS, TEXAS 75265

PARAMETER MEASUREMENT INFORMATION

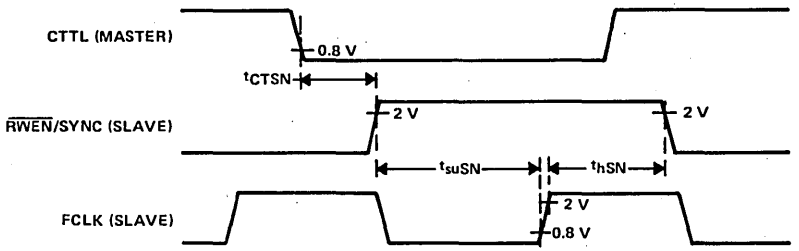


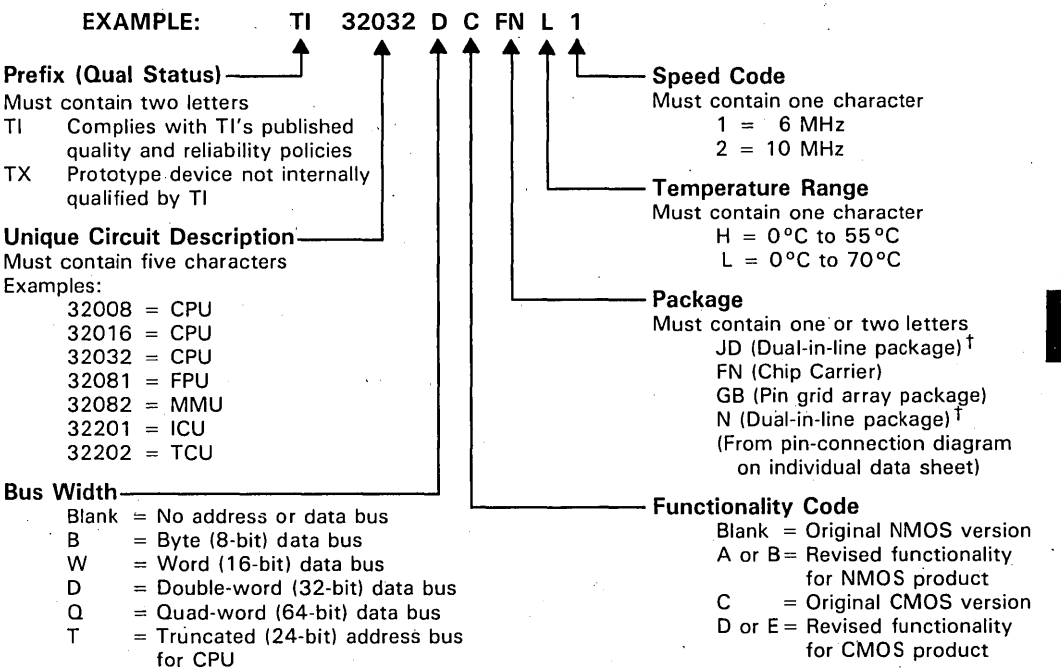
FIGURE 13. SYNCHRONIZATION TIMING

5 Ordering Information and Packaging Mechanical Data

5.1 Ordering Instructions

Electrical characteristics presented in this data manual, unless otherwise noted, apply for circuit type(s) listed in the page heading regardless of package. The availability of a circuit function in a particular package is denoted by an alphabetical reference above the pin-connection diagram(s). These alphabetical references refer to mechanical outline drawings shown in this section.

Factory orders for circuits described in this catalog should include a seven-part type number as explained in the following example.



†These circuits in dual-in-line packages are shipped in one of the carriers listed below. Unless a specific method of shipment is specified by the customer (with possible additional costs), circuits will be shipped in the most practical carrier. Please contact your TI sales representative for the method that will best suit your particular needs.

Slide Magazines	Sectioned Cardboard Box
A-Channel Plastic Tubing	Individual Plastic Box

5.2 Packaging Mechanical Data

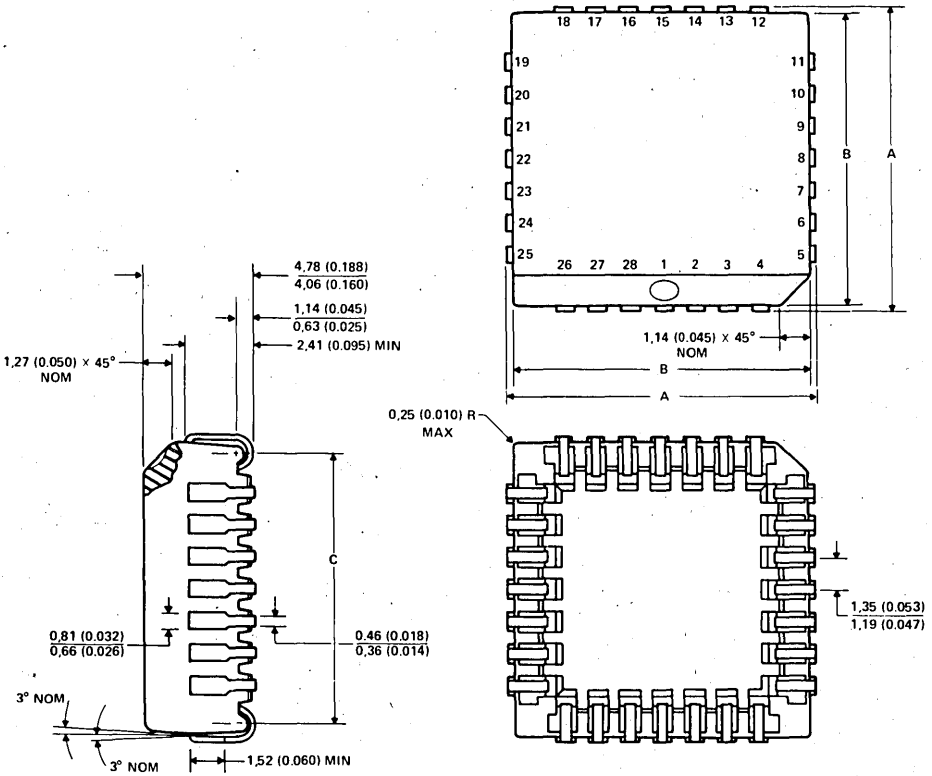
5.2.1 FN Plastic Chip Carrier Package

This chip carrier package consists of a circuit mounted on a lead frame and encapsulated within an electrically nonconductive plastic compound. The compound withstands soldering temperatures with no deformation, and circuit performance characteristics remain stable when the devices are operated in high-humidity conditions. The package is intended for surface mounting on solder lands on 1,27 (0.050-inch) centers, but can also be used in a socketed configuration that is pin-compatible with the 68-Pin Grid Array Package.

FN PLASTIC CHIP CARRIER PACKAGE

(28-terminal package used for illustration)

NO. OF TERMINALS	A		B		C	
	MIN	MAX	MIN	MAX	MIN	MAX
28	12,24 (0.482)	12,57 (0.495)	11,43 (0.450)	11,58 (0.456)	10,41 (0.410)	10,92 (0.430)
68	24,94 (0.982)	25,27 (0.995)	24,13 (0.950)	24,28 (0.956)	23,11 (0.910)	23,62 (0.930)

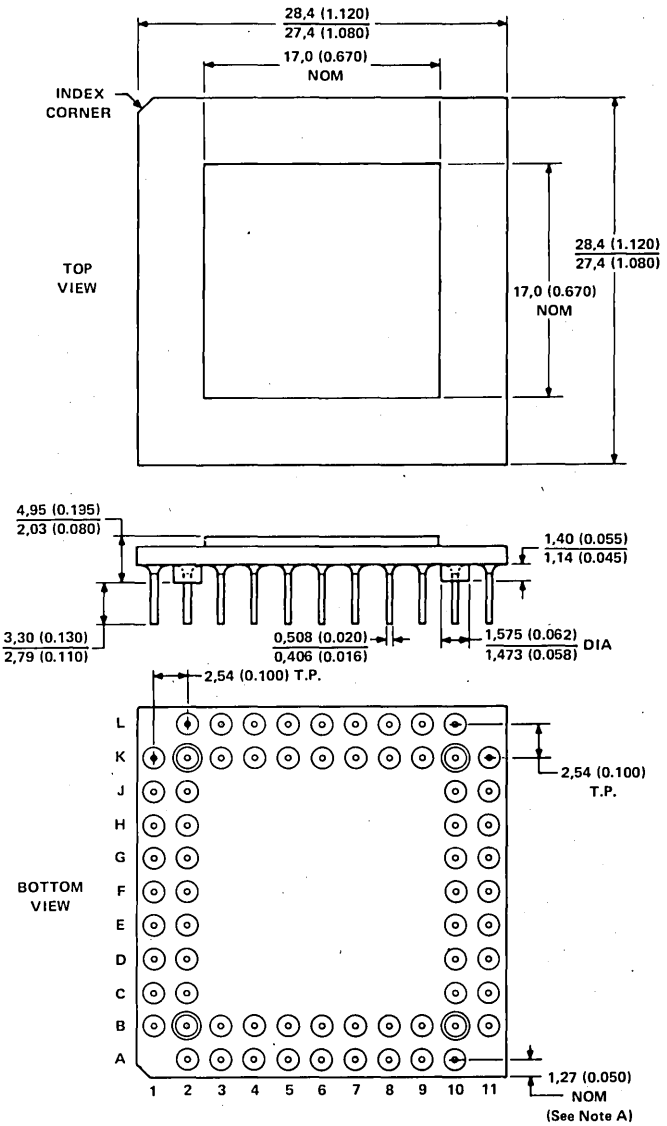


ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES.

5.2.2 GB Ceramic Pin Grid Array Package

This is a hermetically sealed ceramic package with metal cap and gold-plated pins.

GB CERAMIC 68-PIN GRID ARRAY PACKAGE
(CAVITY UP)



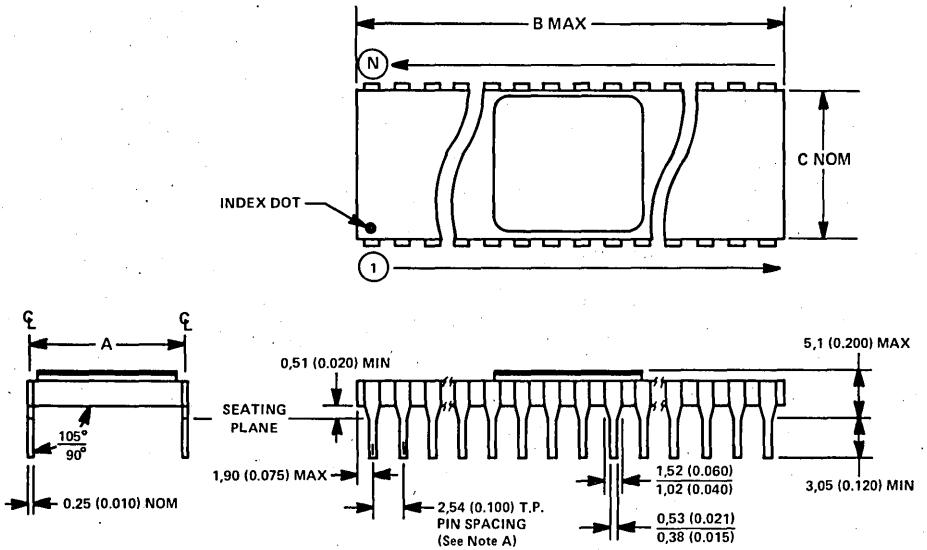
NOTE A: Pins are located within 0,127 (0.005) radius of true position relative to each other at maximum material condition and within 0,254 (0.010) radius relative to the center of the ceramic.

ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES

5.2.3 JD Ceramic Dual-in-Line Package—Side Braze

This is a hermetically sealed ceramic package with a metal cap and side-brazed gold-plated leads.

DIM. \ PINS	24	40	48
A ± 0.025 (0.010)	15,24 (0.600)	15,24 (0.600)	15,24 (0.600)
B (MAX)	32,8 (1.290)	52,1 (2.050)	62,2 (2.45)
C (NOM)	15,1 (0.595)	15,1 (0.59)5	15,1 (0.595)

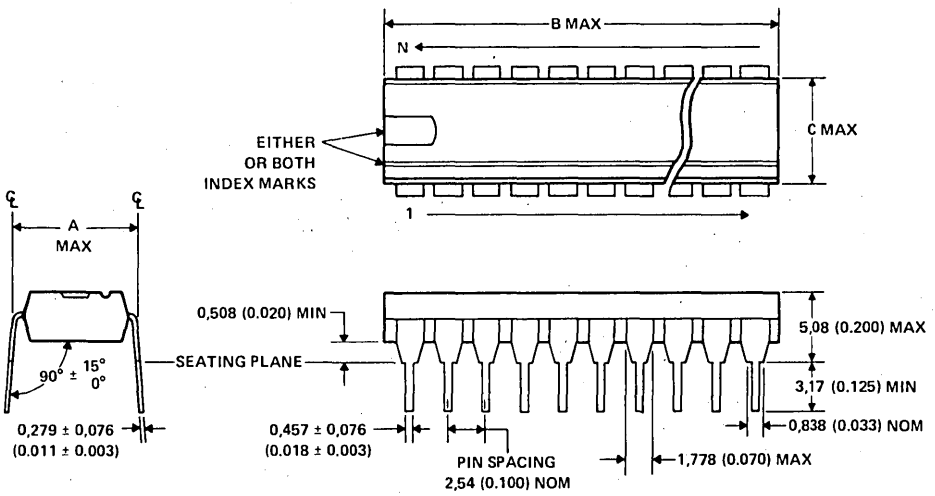


NOTE A: Each pin center line is located within 0,25 (0.010) of its true longitudinal position.

ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES.

5.2.4 N Plastic Packages

DIM. \ PINS	24	40	48
A (MAX)	15,88 (0.625	15,88 (0.625)	15,88 (0.625)
B (MAX)	31,8 (1.250)	53,1 (2.090)	62,2 (2.450)
C (MAX)	13,97 (0.550)	13,97 (0.550)	13,97 (0.550)



ALL LINEAR DIMENSIONS ARE IN MILLIMETERS AND PARENTHETICALLY IN INCHES.

6 Glossary

Abort — The first step of recovery when an instruction or its operand(s) is not available in main memory. An Abort is initiated by the Memory Management Unit (MMU) and handled by the CPU.

Absolute Address — An address that is permanently assigned to a fixed location in main memory. In assembly code, a pattern of characters that identifies a fixed storage location.

Access Time — The time interval between when a request for information is made and the instant this information is available.

Access Class — The five TI32000 family access classes are memory read, memory write, memory read-modify-write, memory address, and register address. The access class informs the TI32000 family CPU how to interpret a reference to a general operand. Each instruction assigns an access class to each of its two operands, which in turn fully defines the action of any addressing mode in referencing that operand.

Accumulator — A device which stores the result of an ALU operation.

ACU — Application-specific Coprocessor Unit. This is a user-designed device which utilizes the same coprocessor interface as the dedicated coprocessors (FPU and MMU). The TI32000 family instruction set contains 20 instructions designed specifically for use with an ACU, but the exact interpretation of these instructions is left to the system designer.

Address — An expression, usually numerical, which designates a specific location in a storage or memory device.

Address-Data Register — A register which may contain either address or data, sometimes referred to as a general-purpose register.

Address Strobe — Control signal used to tell external devices when the address is valid on the external address bus.

Address Translation — The process by which a logical address emanating from the CPU is transformed into a physical address to main memory. This is performed by the Memory Management Unit (MMU) in TI32000 family systems. Logical address to Physical address mapping is established by the operating system when it brings pages into main memory.

Addressing Mode — The manner in which an operand is accessed. TI32000 family CPUs have nine addressing modes: Register, Register Relative, Memory Relative, Immediate, Absolute, External, Top-of-Stack, Memory Space, and Scaled Indexing.

Algorithm — A set of procedures by which a given result is obtained.

Alignment — The issue of whether an instruction must begin on a byte, double byte, or quad byte address boundary.

ALU — Arithmetic Logic Unit. A computational subsystem which performs the arithmetic and logical operations of a digital system.

Array — A structured data type consisting of a number of elements, all of the same data type, such that each data element can be individually identified by an integer index. Arrays represent a basic storage data type used in all high-level languages.

ASCII — (*American National Standard Code for Information Interchange*, 1968). This standard code uses a character set coded as 7-bit characters (8-bits when using parity check). It is used for information interchange among data processing systems, communication systems, and associated equipment. The ASCII set consists of alpha, numeric, control and graphic characters. Synonymous with USASCII.

Assemble — To prepare a machine language program (also called machine code or object code) from a symbolic language program by substituting absolute operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic addresses. Machine code is a series of ones and zeros which a computer understands.

Assembler — A computer program that assembles.

Assembly Language — A machine-specific language whose symbolic instructions are usually in one-to-one correspondence to machine language instructions.

Associative Cache — A storage area where each data entry has an associated "tag". The tags are simultaneously compared to some input value (a logical address) in the case of the MMU, and if a matching tag is resident, the associated data entry is emitted. An associative cache is present within the MMU in TI32000 family systems to provide logical-to-physical address translation.

Asynchronous Device — A device in which the speed of operation is not related to any frequency in the system to which it is connected.

Baud Rate — Data transfer rate. For most serial transmission protocols, this is synonymous with bits-per-second (bps).

BCD — Binary Coded Decimal. A binary numbering system for coding decimal numbers. A 4-bit grouping provides a binary value range from 0000 to 1001, and codes the decimal digits "0" through "9". To count to 9 requires a single 4-bit grouping; to count to 99 takes two groupings of 4 bits; to count to 999 takes three groupings of 4 bits, etc.

Benchmark — In terms of computers, this refers to a software program designed to perform some task which will demonstrate the relative processing speed of one computer versus another.

Bit — An abbreviation of "binary digit". It is a unit of information represented by either a one or a zero.

Bit Field — A group of bits addressable as a single entity. A bit field is fully specified by the location of its least significant bit and its length in bits. In TI32000 family systems, bit fields may be from one to 32 bits in length.

Branch — A nonsequential flow in a software instruction stream.

Breakpoint — A place in a routine specified by an instruction, instruction digit, or other condition, where the software program flow will be interrupted by external intervention or by a monitor routine.

Buffer — An isolating circuit used to avoid reaction of a driven circuit on the corresponding driver circuit. Buffers also supply increased current drive capacity.

Bus — A group of conductors used for transmitting signals or power.

Bus Cycle — The time necessary to complete one transfer of information requiring the use of external address, data and control buses.

Byte — eight bits.

Byte Enable — $\overline{BE0}$ to $\overline{BE3}$. CPU control signals which activate memory banks, each bank providing one byte of data per address.

Cache — See Associative Cache.

Cache Hit — In the MMU, logical-to-physical address translation takes place via the associative cache. For this to happen, the addressed page must be resident in physical memory such that a logical address tag is present in the MMU's translation cache.

Cache Miss — When a logical address is presented to the MMU, and no physical address translation entry is found in the MMU's associative cache.

Cascaded — Interrupt Control Units present in a TI32000 family system which are in addition to the Master ICU are referred to as "cascaded" ICUs; i.e., interrupts cascade from a second-level ICU through the master ICU to the CPU.

Clock — A device that generates a periodic signal used for synchronization.

Clock Cycle — After making a low-to-high transition, the clock will have completed one cycle when it is about to make another low-to-high transition. This time is equal to 1/f where f = the clock frequency.

Compile — To take a program written in a High-Level Language such as C, Pascal, or FORTRAN and convert it into an object-code format which can be loaded into a computer's main memory by a linker/loader program. During compilation, symbolic HLL statements, called source code, are converted into one or more machine instructions which the CPU understands. A compiler also performs the assemble function.

Compiler — A program that compiles.

Coprocessor — A processor which cooperates with the main microprocessor in executing certain instructions from the instruction stream. A coprocessor generally accelerates certain functions which increases overall system throughput. Examples of coprocessors are floating-point accelerators and memory-management units.

CPU — The portion of a computer system that contains the arithmetic logic unit, register file, and other control oriented subsystems. It performs arithmetic operations, controls instruction processing, and provides timing signals and other housekeeping operations.

Demand-Paged Virtual Memory — A virtual memory method in which memory is divided into blocks of equal size which are referred to as pages. These pages are then moved back and forth between main memory and secondary storage as required by the CPU. Demand paging reduces the problem of memory fragmentation which results in unused memory space.

Dispatch Table — In TI32000 family systems, this is an area of memory which contains interrupt descriptors for all possible hardware interrupts and software traps. The interrupt descriptor directs the CPU to the module descriptor for the procedure which is designed to handle that particular interrupt.

Displacement — A numerical offset from a known point of reference. Displacements are used in programming to facilitate position independent code, such that a given program can be loaded anywhere in memory. In TI32000 family processors, a displacement is contained in the instruction itself, as opposed to being stored in a register by the programmer.

DMA — Direct Memory Access. A method that uses a small processor whose sole task is that of controlling input-output or data movement. With DMA, data is moved into or out of the system without program intervention once the DMA controller has been initialized by the CPU.

Double-Precision — With reference to floating-point arithmetic, a double-precision number has a 52-bit fraction field, 11-bit exponent field and a sign bit.

Double Word — Two words, i.e., 32 bits.

Emulate — To imitate one system with another, such that the imitating system accepts the same data, executes the same programs, and achieves the same results as the imitated system.

Exception — An occurrence which must be resolved through CPU intervention. An exception results in the suspension of normal program flow. In TI32000 family systems, exceptions occur as a result of a hardware reset, interrupt or software traps. Execution of floating-point instructions may also result in occurrences which must be resolved through CPU intervention.

Exponent — In floating-point representation, a numeral that indicates the power to which the base is raised.

FIFO — First-in first-out memory. A memory from which data can be read out in the same order as it was entered, but not necessarily at the same rate.

Firmware — Computer programs and data loaded in a class of memory that cannot be dynamically modified by the computer during processing.

Floating-Point — A method by which computers deal with numbers having a fractional component. In general, it pertains to a system in which the location of the decimal/binary point does not remain fixed with respect to one end of numerical expressions, but is regularly recalculated. The location of the point is usually given by expressing a power of the base.

FP — Frame Pointer. CPU register which points to a dynamically allocated data area created at the beginning of a procedure by the ENTER instruction.

FPU — Floating-Point Unit. Coprocessor in TI32000 family systems which implements in hardware all calculations needed to support floating-point arithmetic, which otherwise would have to be implemented in software.

Fragmented — The term used to describe the presence of small, unused blocks of memory. The problem is especially common in segmented memory systems, and results in inefficient use of memory storage.

Frame — A block of memory on the stack that provides local storage for parameters in the current procedure.

Hardware — Physical equipment, as opposed to the software programs or method in which the hardware is used, e.g., mechanical, magnetic, electrical, or electronic devices.

HLL — High-Level Language. A programming language in which the statements represent procedures which are multiple machine instructions. C, Pascal, FORTRAN, LISP and BASIC are some common high-level languages. A HLL requires a compiler or interpreter that translates each HLL statement into a series of machine language instructions.

ICE — In-Circuit-Emulator. A computer system which imitates the operation of another in terms of software execution. In microprocessor system development, the ICE takes the place of the microprocessor by means of a connector at the end of an umbilical cable. Not only does the ICE perform all the functions of the microprocessor, but it also allows the engineer to debug his system by setting breakpoints on various conditions, permits tracing of program flow, and provides substitution memory which may be used in place of actual target system memory.

ICU — Interrupt Control Unit. A memory-mapped microprocessor support chip in TI32000 family systems which handles external interrupts as well as additional software traps. The ICU provides a vector to the CPU to identify the servicing software procedure.

Indexing — In computers, a method of address modification that is by means of index registers.

Index Register — A register whose contents may be added to or subtracted from the operand address.

Indirect Addressing — Programming method where the initial address is the storage location of a word which is the actual address. This indirect address is the location of the data to be operated upon.

Instruction — A statement that specifies an operation and the values or locations of its operands, i.e., it tells the CPU what to do and to what.

Instruction Cycle — The period of time during which a programmed system obeys a particular instruction.

Instruction Fetch — The action of accessing the next instruction from memory, often overlapped by its partial execution.

Instruction Queue — In TI32000 family CPUs, this is a small area of RAM organized as a FIFO buffer which stores prefetched instructions until the CPU is ready to execute them.

Interpreter — A program which translates HLL statements into machine instructions in real-time, i.e., while the program is executing.

Interrupt — To stop a software program in such a way that it can be resumed. Interrupts can be caused by events external or internal to the CPU, and by either software or hardware.

INTBASE — Interrupt Base Register. A 32-bit CPU register which holds the address of the dispatch table used for interrupts and traps.

Linear Address Space — An address space where addresses start at location zero and proceed in a linear fashion (i.e., with no holes or breaks) to the upper limit imposed by the total number of bits in a logical address.

Link Base — Module Descriptor entry which points to a table in memory containing entries which reference variables or entry points in Modules external to the one presently executing.

Linker — A computer program used to create one load module from one or more independent translated object modules or load modules by resolving cross-references among the object modules, and possibly by relocating the elements.

Logical Address Space — The range of addresses which a programmer can assign in a software program. This range is determined by the length of the computer's address registers.

LSB — Least Significant Bit. The bit in a string of bits representing the lowest value.

Machine Code — The code that a computer recognizes. Specifies internal register files and operations that directly control the computer's internal hardware.

Main Memory — The program and data storage area in a computer system which is physically addressed by the microprocessor or MMU address lines.

Mantissa — In a floating-point number, this is the fractional component.

Mapping — The process whereby the operating system assigns physical addresses in main memory to the logical addresses assigned by the software.

Memory-Mapped — Referring to peripheral hardware devices which are addressed as if they were part of the computer's memory space. They are accessed in the same manner as main memory, i.e., through memory read/write operations.

Microcode — A sequence of primitive instructions that control the internal hardware of a computer. Their execution is initiated by the decoding of a software instruction. Microcode is maintained in special storage and often used in place of hard-wired logic.

Microcomputer — A computer system whose Central Processing Unit is a Microprocessor. Generally refers to a board-level product.

Minicomputer — A "box-level" computer with system capabilities generally between that of a microcomputer and a mainframe.

MMU — Memory Management Unit. This is a coprocessor in the TI32000 family which aids in the implementation of demand-paged virtual memory. It provides logical to physical address translation and initiates an instruction abort to the CPU when a desired memory location is not in main memory.

MOD — Mod Register. A 16-bit CPU register which holds the address of the Module Descriptor of the currently executing software module.

Module — An independent subprogram that performs a specific function and is usually part of a task, i.e., part of a larger program.

Module Descriptor — In the TI32000 family, a set of four 32-bit entries found in main memory which point to the static data area, link table, and first instruction of the module it describes.

Modularity — A software concept which provides a means of overcoming natural human limitations for dealing with programming complexity by specifying the subdivision of large and complex programming tasks into smaller and simpler subprograms, or modules, each of which performs some well-defined portion of the complete processing task.

MSB — Most Significant Bit. The bit in a string of bits representing the highest value.

NMI — Nonmaskable Interrupt. A hardware interrupt which cannot be disabled by software. It is generally the highest priority interrupt.

Object Code — Output from a compiler or assembler which is itself executable machine code or is suitable for processing to produce executable machine code.

Operand — In a computer, a datum which is processed by the CPU. It is referenced by the address part of an instruction.

Operating System — Software which controls the execution of computer programs and which may provide storage assignment, input/output control, scheduling, data management, accounting, debugging, editing, and related services.

Operating System Mode — In this mode, the CPU can execute all instructions in the instruction set, access all bits in the Processor Status Register, and access any memory location available to the processor.

Operator — In the description of an instruction, it is the action to be performed on operands.

Page Fault — A hardware generated trap used to tell the operating system to bring the missing page in from secondary storage.

Page Swap — The exchange of a page of software in secondary storage with another page located in main memory. The operating system supervises this operation, which is executed by the CPU and involves external devices such as disk and DMA controllers.

Page Table — A 1K-byte area in main memory containing 256 entries which describe the location and attributes of all pointer tables, i.e., a list of pointer table addresses.

Peripheral — A device which is part of the computer system and operates under the supervision of the CPU. Peripheral devices are often physically separated from the CPU.

Physical Address — The address presented to main memory, either by the CPU or MMU.

Pointer Table — A 512-byte page located either in main memory or secondary storage containing 128 entries. Each entry describes an individual page of the software program. Each page of the software program may reside in main memory or in secondary storage.

Pop — To read a datum from the top of a stack.

Primitive Data Type — A data type which can be directly manipulated by the hardware. In the TI32000 family, these are integers, floating-point numbers, Booleans, BCD digits, and bit fields.

Procedure — A subprogram which performs a particular function required by a module, i.e., by a larger program; an ordered set of instructions that have a general or frequent use.

Process — A task.

Program Base — Module Descriptor entry which points to the first instruction in the module being described.

Program Counter — CPU register which specifies the logical address of the currently executing instruction.

Protection — The process of restricting a software program's access to certain portions of memory using hardware mechanisms. Typically done at the operating system and page level.

PSR — Processor Status Register. A 16-bit register on TI32000 family CPU's which contains bits used by the software to make decisions and determine program flow.

Push — To write a datum to the top of a stack.

Quad word — Four words, i.e., 64 bits.

Queue — A First-In-First-Out data storage area, in which the data may be removed at a rate different from that at which it was stored.

Real Time — Pertaining to the performance of a computation during the actual physical time that relates to the time in which the physical process transpires, in order that the results of the computation can be used in guiding the process in the next unit of physical time.

Record — A structured data type with multiple elements, each of which may be of a different data type, e.g., strings, arrays, bytes, etc.

Register — A temporary storage location, usually in the CPU, which holds digital data.

Relative Address — The number that specifies the difference between the base address and the absolute address.

Relocatable — In reference to software programs, this is code which can be loaded into any location in main memory without affecting the operation of the program.

Return Address — The address to which a subroutine call, interrupt or trap subroutine will return after it is finished executing.

Routine — A procedure.

SB — Static Base Register. Points to the start of the static data area for the currently executing module.

Secondary Storage — This is generally slow-access, nonvolatile memory such as a hard-disk which is used to store the pages of software programs not currently needed by the CPU.

Segmented Address Space — Term used to describe the division of allocatable memory space into blocks or segments of variable size.

Setup Time — The minimum amount of time that data must be present at an input to ensure data acceptance when the device is clocked.

Software — A user program consisting of tasks, modules and procedures; the rules and possibly associated documentation and data pertaining to the operation of a computer system.

SP1 — User Stack Pointer. Points to the top of the User Stack and is selected for all stack operations while in User Mode.

SP0 — Interrupt Stack Pointer. Points to the top of the interrupt stack. It is used by the operating system and whenever an interrupt or trap occurs.

Stack — A one-dimensional data structure in which values are entered and removed one datum at a time from a location called the Top-of-Stack. To the programmer, it appears as a block of memory and a variable called the Stack Pointer (which points to the top of the stack).

Stack Pointer — CPU register which points to the top of a stack.

Static Base Register — A 32-bit CPU register which points to the beginning of the static data area for the currently executing module.

String — An array of integers, all of the same length. The integers may be bytes, words, or double words. The integers may be interpreted in various ways (see ASCII).

Subroutine — A self-contained program which is part of a procedure.

Symmetry — A computer architecture is said to be symmetrical when any instruction can specify any operand length (byte, word or double word) and make use of any address-data register or memory location while using any addressing mode.

Synchronous — Refers to two or more things made to happen in a system at the same time, by means of a common clock signal.

Tag — A label appended to some data entry used in a look-up process whereby the desired datum can be identified by its tag.

Task — The highest-level subdivision of a user software program. The largest program entity that a computer's hardware directly deals with.

TCU — Timing Control Unit. A device used to provide system clocks, bus control signals and bus cycle extension capability in TI32000 family systems.

Trap – An internally generated interrupt request caused as a direct and immediate result of the execution of an instruction.

T-State – One clock period. If the system clock frequency is 10 MHz, one T-State will take 100 ns to complete. Operations internal and external to the CPU are synchronized to the beginning and middle of the T-States. There are four T-States in a normal TI32000 family CPU bus cycle.

UNIX™ – An operating system developed at Bell Laboratories in the early 1970s. Software programs that run under UNIX are written in the high-level language C, making them highly portable. UNIX systems do not distinguish user programs from operating system programs in either capability or usage, and they allow users to route the output of one program directly into the input of another.

User – A software program. The total set of tasks (instructions) that accomplish a desired result. Tasks are managed by the operating system.

User Mode – Machine state in which the executing procedure has limited use of the instruction set and limited access to memory and the PSR.

Variable – A parameter that can assume any of a given set of values.

Vector – Byte provided by the ICU (Interrupt Control Unit) which tells the CPU where within the Descriptor table the descriptor is located for the interrupt it has just requested.

Virtual Address – An address generated by a user's logical address (working in conjunction with the operating system) which maps into an address in main or secondary storage.

Virtual Memory – The storage space that may be regarded as addressable main storage by the system. The operating system maps Virtual addresses into physical (main memory) addresses. The size of virtual memory is limited by the method of memory management employed and by the amount of secondary storage available, not by the actual number of main storage locations.

Wait-State – An additional clock period added to a CPU memory cycle which gives an external memory device additional time to provide the CPU with data. Also used by bus arbitration circuitry to hold the CPU in an idle state until access to a shared resource is gained.

Winchester – Small, hard-disk media commonly found in personal computers.

Word – A character string or bit string considered as the primary data entity. For historical reasons, a word is a group of 16 bits in TI32000 family systems.

UNIX is a trademark of AT&T Bell Laboratories

Appendix
Instruction Formats

NOTATIONS

i = Integer Type Field
- B = 00 (Byte)
- W = 01 (Word)
- D = 11 (Double Word)

f = Floating-Point Type Field
- F = 1 (Std. Floating: 32 bits)
- L = 0 (Long Floating: 64 bits)

c = ACU Type Field
- D = 1 (Double Word)
- Q = 0 (Quad Word)

op = Operation Code
Valid encodings shown with each format.

gen, gen 1, gen 2 = General Addressing Mode Field

reg = Address-Data Register Number

cond = Condition Code Field
- 0000 = Equal: Z = 1
- 0001 = Not Equal: Z = 0
- 0010 = Carry Set: C = 1
- 0011 = Carry Clear: C = 0
- 0100 = Higher: L = 1
- 0101 = Lower or Same: L = 0
- 0110 = Greater Than: N = 1
- 0111 = Less or Equal: N = 0
- 1000 = Flag Set: F = 1
- 1001 = Flag Clear: F = 0
- 1010 = Lower: L = 0 and Z = 0
- 1011 = Higher or Same: L = 1 or Z = 1
- 1100 = Less Than: N = 0 and Z = 0
- 1101 = Greater or Equal: N = 1 or Z = 1
- 1110 = (Unconditionally True)
- 1111 = (Unconditionally False)

short = Short Immediate value. May contain
quick: Signed 4-bit value, in MOVQ, ADDQ, CMPQ, ACB
cond: Condition Code (above), in Scond.
areg: CPU Dedicated Register, in LPR, SPR.
0000 = US
0001 = 0111 = (Reserved)
1000 = FP
1001 = SP
1010 = SB
1011 = (Reserved)
1100 = (Reserved)
1101 = PSR
1110 = INTBASE
1111 = MOD
Options: in String Instructions

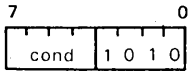
U/W	B	T

T = Translated
B = Backward
U/W = 00: None
01: While Match
11: Until Match

Configuration bits, in SETCFG:

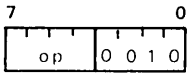
C	M	F	I

mreg: MMU Register number, in LMR, SMR.
0000 = BPR0
0001 = BPR1
0010 = (Reserved)
0011 = (Reserved)
0100 = (Reserved)
0101 = (Reserved)
0110 = (Reserved)
0111 = (Reserved)
1000 = (Reserved)
1001 = (Reserved)
1010 = MSR
1011 = BCNT
1100 = PTB0
1101 = PTB1
1110 = (Reserved)
1111 = EIA



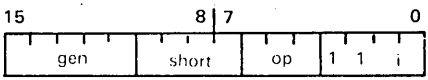
Format 0

Bcond (BR)



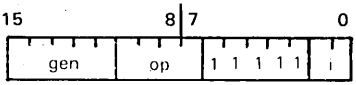
Format 1

BSR	-0000	ENTER	-1000
RET	-0001	EXIT	-1001
CXP	-0010	NOP	-1010
RXP	-0011	WAIT	-1011
RETT	-0100	DIA	-1100
RETI	-0101	FLAG	-1101
SAVE	-0110	SVC	-1110
RESTORE	-0111	BPT	-1111



Format 2

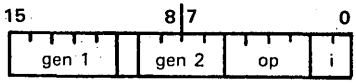
ADDQ	-000	ACB	-100
CMPQ	-001	MOVQ	-101
SPR	-010	LPR	-110
Scond	-011		



Format 3

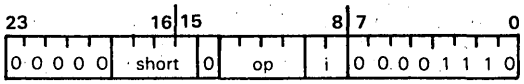
CXPD	-0000	ADJSP	-1010
BICPSR	-0010	JSR	-1100
JUMP	-0100	CASE	-1110
BISPSR	-0110		

Trap (UND) on XXX1, 1000



Format 4

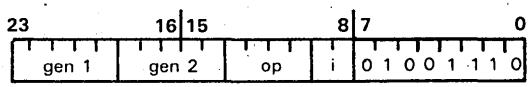
ADD	-0000	SUB	-1000
CMP	-0001	ADDR	-1001
BIC	-0010	AND	-1010
ADDC	-0100	SUBC	-1100
MOV	-0101	TBIT	-1101
OR	-0110	XOR	-1110



Format 5

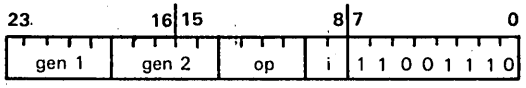
MOVS	-0000	SETCFG	-0010
CMPS	-0001	SKPS	-0011

Trap (UND) on 1XXX, 01XX



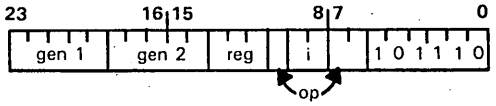
Format 6

ROT	-0000	NEG	-1000
ASH	-0001	NOT	-1001
CBIT	-0010	Trap (UND)	-1010
CBITI	-0011	SUBP	-1011
Trap (UND)	-0100	ABS	-1100
LSH	-0101	COM	-1101
SBIT	-0110	IBIT	-1110
SBITI	-0111	ADDP	-1111



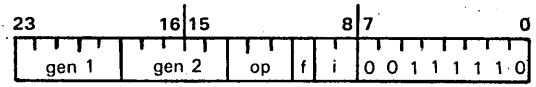
Format 7

MOVM	-0000	MUL	-1000
CMPM	-0001	MEI	-1001
INSS	-0010	Trap (UND)	-1010
EXTS	-0011	DEI	-1011
MOVXBW	-0100	QUO	-1100
MOVZBW	-0101	REM	-1101
MOVZiD	-0110	MOD	-1110
MOVXiD	-0111	DIV	-1111



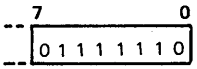
Format 8

EXT	-0 00	INDEX	-1 00
CVTP	-0 01	FFS	-1 01
INS	-0 10		
CHECK	-0 11		
MOVSU	-110, reg = 001		
MOVUS	-110, reg = 011		



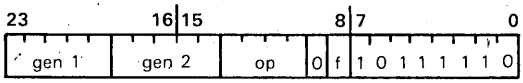
Format 9

MOVif	-000	ROUND	-100
LFSR	-001	TRUNC	-101
MOVLF	-010	SFSR	-110
MOVFL	-011	FLOOR	-111



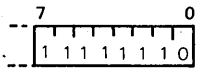
Format 10

Trap (UND) Always



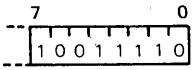
Format 11

ADDf	-0000	DIVf	-1000
MOVf	-0001	Trap (UND)	-1010
CMPf	-0010	Trap (UND)	-1011
SUBf	-0100	MULf	-1100
NEGf	-0101	ABSf	-1101
Trap (UND)	-0110	Trap (UND)	-1110
Trap (UND)	-0111	Trap (UND)	-1111



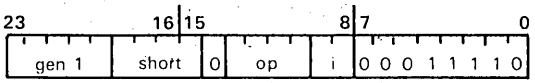
Format 12

Trap (UND) Always



Format 13

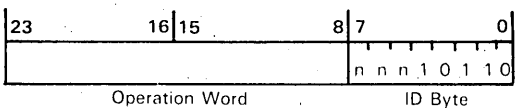
Trap (UND) Always



Format 14

RDVAL	-0000	LMR	1010
WRVAL	-0001	SMR	1011

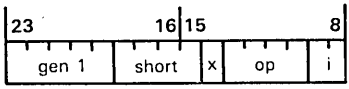
Trap (UND) on 01XX, 1XXX



Format 15

(ACU)

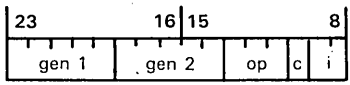
nnn **Operation Word Format**



Format 15.0

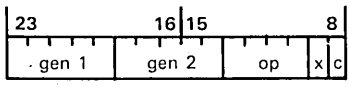
CATST0	-0000	LCR	-1010
CATST1	-0001	SCR	-1011

Trap (UND) on all others



Format 15.1

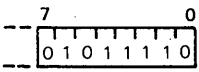
CCV3	-000	CCV2	-100
LCSR	-001	CCV1	-110
CCV5	-010	SCSR	-110
CCV4	-011	CCV0	-111



Format 15.5

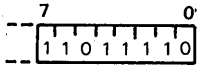
CCAL0	-0000	CCAL3	-1000
CMOV0	-0001	Trap (UND)	-1010
CCMP	-0010	Trap (UND)	-1011
CCAL1	-0100	CCAL2	-1100
CMOV2	-0101	CMOV1	-1101
Trap (UND)	-0110	Trap (UND)	-1110
Trap (UND)	-0111	Trap (UND)	-1111

If nnn = 010, 011, 100, 110, 111 then Trap (UND) Always



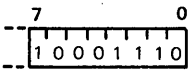
Format 16

Trap (UND) Always



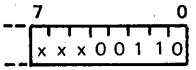
Format 17

Trap (UND) Always



Format 18

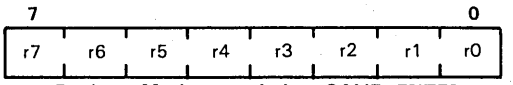
Trap (UND) Always



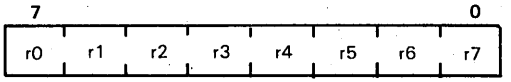
Format 19

Trap (UND) Always

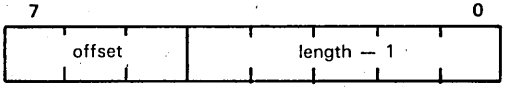
Implied Immediate Encodings:



Register Mark, apended to SAVE, ENTER



Register Mark, appended to RESTORE, EXIT



Offset/Length Modifier appended to INSS, EXTS

TI Sales Offices

ALABAMA: Huntsville (205) 837-7530.

ARIZONA: Phoenix (602) 995-1007.

CALIFORNIA: Irvine (714) 660-1200;
Sacramento (916) 929-1521;
San Diego (619) 278-9601;
Santa Clara (408) 980-9000;
Torrance (213) 217-7010;
Woodland Hills (213) 704-7759.

COLORADO: Aurora (303) 368-8000.

CONNECTICUT: Wallingford (203) 269-0074.

FLORIDA: Ft. Lauderdale (305) 973-8502;
Maitland (305) 660-4600; Tampa (813) 870-6420.

GEORGIA: Norcross (404) 662-7900.

ILLINOIS: Arlington Heights (312) 640-2925.

INDIANA: Ft. Wayne (219) 424-5174;
Indianapolis (317) 248-8555.

IOWA: Cedar Rapids (319) 395-9550.

MARYLAND: Baltimore (301) 944-8600.

MASSACHUSETTS: Waltham (617) 895-9100.

MICHIGAN: Farmington Hills (313) 553-1500.

MINNESOTA: Eden Prairie (612) 828-9300.

MISSOURI: Kansas City (816) 523-2500;
St. Louis (314) 569-7600.

NEW JERSEY: Iselin (201) 750-1050.

NEW MEXICO: Albuquerque (505) 345-2555.

NEW YORK: East Syracuse (315) 463-9291;
Endicott (607) 754-3900; Melville (516) 454-6600;
Pittsford (716) 385-6770;
Poughkeepsie (914) 473-2900.

NORTH CAROLINA: Charlotte (704) 527-0930;
Raleigh (919) 876-2725.

OHIO: Beachwood (216) 464-6100;
Dayton (513) 258-3877.

OKLAHOMA: Tulsa (918) 250-0633.

OREGON: Beaverton (503) 643-6758.

PENNSYLVANIA: Ft. Washington (215) 643-6450;
Coraopolis (412) 771-8550.

PUERTO RICO: Hato Rey (809) 753-8700

TEXAS: Austin (512) 250-7655;
Houston (713) 778-6592; Richardson (214) 680-5082;
San Antonio (512) 496-1779.

UTAH: Murray (801) 266-8972.

VIRGINIA: Fairfax (703) 849-1400.

WASHINGTON: Redmond (206) 881-3080.

WISCONSIN: Brookfield (414) 785-7140.

CANADA: Nepean, Ontario (613) 726-1970;
Richmond Hill, Ontario (416) 884-9181;
St. Laurent, Quebec (514) 334-3635.

TI Regional Technology Centers

CALIFORNIA: Irvine (714) 660-8140,
Santa Clara (408) 748-2220.

GEORGIA: Norcross (404) 662-7945.

ILLINOIS: Arlington Heights (312) 640-2909.

MASSACHUSETTS: Waltham (617) 890-6671.

TEXAS: Richardson (214) 680-5066.

Technical Support Center

(214) 995-2171

For further literature information:
(800) 232-3200

TI Distributors

TI AUTHORIZED DISTRIBUTORS IN USA

Arrow Electronics
Diplomat Electronics
ESCO Electronics
General Radio Supply Company
Graham Electronics
Harrison Equipment Co.
International Electronics
JACO Electronics
Kierulff Electronics
LCOMP, Incorporated
Marshall Industries
Milgray Electronics
Newark Electronics
Rochester Radio Supply
Time Electronics
R.V. Weatherford Co.
Wyle Laboratories

TI AUTHORIZED DISTRIBUTORS IN CANADA

CESCO Electronics, Inc.
Future Electronics
ITT Components
L.A. Varah, Ltd.

ALABAMA: Arrow (205) 882-2730;
Kierulff (205) 883-6070; Marshall (205) 881-9235.

ARIZONA: Arrow (602) 968-4800;
Kierulff (602) 243-4101; Marshall (602) 968-6181;
Wyle (602) 249-2232.

CALIFORNIA: Los Angeles/Orange County:
Arrow (818) 701-7500, (714) 838-5422;
Kierulff (213) 725-0325, (714) 731-5711, (714) 220-6300;
Marshall (213) 999-5001, (818) 442-7204,
(714) 660-0951; R.V. Weatherford (714) 634-9600,
(213) 849-3451, (714) 623-1261; Wyle (213) 322-8100,
(714) 863-9953; **Sacramento:** Arrow (916) 925-7456;
Wyle (916) 638-5282; **San Diego:**
Arrow (619) 565-4800; Kierulff (619) 278-2112;
Marshall (619) 578-9600; Wyle (619) 565-9171;
San Francisco Bay Area: Arrow (408) 745-6600;
(415) 487-4600; Kierulff (408) 971-2600;
Marshall (408) 732-1100; Wyle (408) 727-2500;
Santa Barbara: R.V. Weatherford (805) 965-8551.

COLORADO: Arrow (303) 696-1111;
Kierulff (303) 790-4444; Wyle (303) 457-9953.

CONNECTICUT: Arrow (203) 265-7741;
Diplomat (203) 797-9674; Kierulff (203) 265-1115;
Marshall (203) 265-3822; Milgray (203) 795-0714.

FLORIDA: Ft. Lauderdale: Arrow (305) 429-8200;
Diplomat (305) 974-8700; Kierulff (305) 486-4004;
Orlando: Arrow (305) 725-1480;
Milgray (305) 647-5747; **Tampa:**
Arrow (813) 576-8995; Diplomat (813) 443-4514;
Kierulff (813) 576-1966.

GEORGIA: Arrow (404) 449-8252;
Kierulff (404) 447-5252; Marshall (404) 923-5750.

TEXAS INSTRUMENTS

Creating useful products
and services for you.

ILLINOIS: Arrow (312) 397-3440;
Diplomat (312) 595-1000; Kierulff (312) 640-0200;
Marshall (312) 490-0155; Newark (312) 638-4411.

INDIANA: Indianapolis: Arrow (317) 243-9353;
Graham (317) 634-8202;
Ft. Wayne: Graham (219) 423-3422.

IOWA: Arrow (319) 395-7230.

KANSAS: Kansas City: Marshall (913) 492-3121;
Wichita: LCOMP (316) 265-9507.

MARYLAND: Arrow (301) 247-5200;
Diplomat (301) 995-1226; Kierulff (301) 636-5800;
Milgray (301) 793-3993.

MASSACHUSETTS: Arrow (617) 933-8130;
Diplomat (617) 935-6611; Kierulff (617) 667-8331;
Marshall (617) 272-8200; Time (617) 935-8080.

MICHIGAN: Detroit: Arrow (313) 971-8220;
Marshall (313) 525-5850; Newark (313) 967-0600;
Grand Rapids: Arrow (616) 243-0912.

MINNESOTA: Arrow (612) 830-1800;
Kierulff (612) 941-7500; Marshall (612) 559-2211.

MISSOURI: Kansas City: LCOMP (816) 221-2400;
St. Louis: Arrow (314) 567-6888;
Kierulff (314) 739-0855.

NEW HAMPSHIRE: Arrow (603) 668-6968.

NEW JERSEY: Arrow (201) 575-5300, (609) 596-8000;
Diplomat (201) 785-1830;
General Radio (609) 964-8560; Kierulff (201) 575-6750;
(609) 235-1444; Marshall (201) 882-0320,
(609) 234-9100; Milgray (609) 983-5010.

NEW MEXICO: Arrow (505) 243-4566;
International Electronics (505) 345-8127.

NEW YORK: Long Island: Arrow (516) 231-1000;
Diplomat (516) 454-6400; JACO (516) 273-5500;
Marshall (516) 273-2053; Milgray (516) 420-9800;
Rochester: Arrow (716) 275-0300;
Marshall (716) 235-7620;
Rochester Radio Supply (716) 454-7800;
Syracuse: Arrow (315) 652-1000;
Diplomat (315) 652-5000; Marshall (607) 754-1570.

NORTH CAROLINA: Arrow (919) 876-3132,
(919) 725-8711; Kierulff (919) 872-8410.

OHIO: Cincinnati: Graham (513) 772-1661;
Cleveland: Arrow (216) 248-3990;
Kierulff (216) 587-6558; Marshall (216) 248-1788.
Columbus: Graham (614) 895-1590;
Dayton: Arrow (513) 435-5563;
ESCO (513) 226-1133; Kierulff (513) 439-0045;
Marshall (513) 236-8088.

OKLAHOMA: Arrow (918) 665-7700;
Kierulff (918) 252-7537.

OREGON: Arrow (503) 684-1690;
Wyle (503) 640-6000; Marshall (503) 644-5050.

PENNSYLVANIA: Arrow (412) 856-7000,
(215) 928-1800; General Radio (215) 922-7037.

TEXAS: Austin: Arrow (512) 835-4180;
Kierulff (512) 835-2090; Marshall (512) 458-5654;
Wyle (512) 834-9957; **Dallas:** Arrow (214) 380-6464;
International Electronics (214) 233-9323;
Kierulff (214) 343-2400; Marshall (214) 233-5200;
Wyle (214) 235-9953;
El Paso: International Electronics (915) 598-3406;
Houston: Arrow (713) 530-4700;
Marshall (713) 789-6600;
Harrison Equipment (713) 879-2600;
Kierulff (713) 530-7030; Wyle (713) 879-9953.

UTAH: Diplomat (801) 486-4134;
Kierulff (801) 973-6913; Wyle (801) 974-9953.

VIRGINIA: Arrow (804) 282-0413.

WASHINGTON: Arrow (206) 643-4800;
Kierulff (206) 575-4420; Wyle (206) 453-8300; Marshall
(206) 747-9100.

WISCONSIN: Arrow (414) 764-6600;
Kierulff (414) 784-8160.

CANADA: Calgary: Future (403) 259-6408; Varah
(403) 230-1235; **Edmonton:** Future (403) 486-0974;
Montreal: CESCO (514) 735-5511; Future
(514) 694-7710; ITT Components (514) 735-1177;
Ottawa: CESCO (613) 226-6903; Future
(613) 820-8313; ITT Components (613) 226-7406;
Varah (613) 726-8884; **Quebec City:** CESCO
(418) 687-4231; **Toronto:** CESCO (416) 661-0220;
Future (416) 638-4771; ITT Components
(416) 630-7971; Varah (416) 816-9311;
Vancouver: Future (604) 438-5545; Varah
(604) 873-3211; ITT Components (604) 270-7805;
Winnipeg: Varah (204) 633-6190

BK

TI Worldwide Sales Offices

ALABAMA: Huntsville: 500 Wynn Drive, Suite 514, Huntsville, AL 35805, (205) 837-7530.

ARIZONA: Phoenix: 8825 N. 23rd Ave., Phoenix, AZ 85021, (602) 995-1007.

CALIFORNIA: Irvine: 17891 Cartwright Rd., Irvine, CA 92714, (714) 660-1200; **Sacramento:** 1900 Point West Way, Suite 171, Sacramento, CA 95815, (916) 929-1521; **San Diego:** 4333 View Ridge Ave., Suite B., San Diego, CA 92123, (619) 278-9601; **Santa Clara:** 5353 Betsy Ross Dr., Santa Clara, CA 95054, (408) 980-9000; **Torrance:** 19505 Hamilton St., Bldg. A, Suite 1, Torrance, CA 90502, (213) 217-7010; **Woodland Hills:** 21220 Erwin St., Woodland Hills, CA 91367, (213) 704-7759.

COLORADO: Aurora: 1400 S. Potomac Ave., Suite 101, Aurora, CO 80012, (303) 368-8000.

CONNECTICUT: Wallingford: 9 Barnes Industrial Park Rd., Barnes Industrial Park, Wallingford, CT 06492, (203) 269-0074.

FLORIDA: Ft. Lauderdale: 2765 N.W. 62nd St., Ft. Lauderdale, FL 33309, (305) 973-8502; **Maitland:** 2601 Maitland Center Parkway, Maitland, FL 32751, (305) 660-4600; **Tampa:** 5010 W. Kennedy Blvd., Suite 101, Tampa, FL 33609, (813) 870-6420.

GEORGIA: Norcross: 5515 Spalding Drive, Norcross, GA 30092, (404) 662-7900

ILLINOIS: Arlington Heights: 515 W. Algonquin, Arlington Heights, IL 60005, (312) 640-2925.

INDIANA: Ft. Wayne: 2020 Inwood Dr., Ft. Wayne, IN 46815, (219) 424-5174; **Indianapolis:** 2346 S. Lynhurst, Suite J-400, Indianapolis, IN 46241, (317) 248-8555.

IOWA: Cedar Rapids: 373 Collins Rd. NE, Suite 200, Cedar Rapids, IA 52402, (319) 395-9550.

MARYLAND: Baltimore: 1 Rutherford Pl., 7133 Rutherford Rd., Baltimore, MD 21207, (301) 944-8600.

MASSACHUSETTS: Waltham: 504 Totten Pond Rd., Waltham, MA 02154, (617) 895-9100.

MICHIGAN: Farmington Hills: 33737 W. 12 Mile Rd., Farmington Hills, MI 48018, (313) 553-1500.

MINNESOTA: Eden Prairie: 11000 W. 78th St., Eden Prairie, MN 55344 (612) 828-9300.

MISSOURI: Kansas City: 8080 Ward Pkwy., Kansas City, MO 64114, (816) 523-2500; **St. Louis:** 11861 Westline Industrial Drive, St. Louis, MO 63141, (314) 569-7600.

NEW JERSEY: Iselin: 485E U.S. Route 1 South, Parkway Towers, Iselin, NJ 08830 (201) 750-1050

NEW MEXICO: Albuquerque: 2820-D Broadbent Pkwy NE, Albuquerque, NM 87107, (505) 345-2555.

NEW YORK: East Syracuse: 6365 Collamer Dr., East Syracuse, NY 13057, (315) 463-9291; **Endicott:** 112 Nanticoke Ave., P.O. Box 618, Endicott, NY 13760, (607) 754-3900; **Melville:** 1 Huntington Quadrangle, Suite 3C10, P.O. Box 2936, Melville, NY 11747, (516) 454-6600; **Pittsford:** 2851 Clover St., Pittsford, NY 14534, (716) 385-6770; **Poughkeepsie:** 385 South Rd., Poughkeepsie, NY 12601, (914) 473-2900.

NORTH CAROLINA: Charlotte: 8 Woodlawn Green, Woodlawn Rd., Charlotte, NC 28210, (704) 527-0930; **Raleigh:** 2809 Highwoods Blvd., Suite 100, Raleigh, NC 27625, (919) 876-2725.

OHIO: Beachwood: 23408 Commerce Park Rd., Beachwood, OH 44122, (216) 464-6100; **Dayton:** Kingsley Bldg., 4124 Linden Ave., Dayton, OH 45432, (513) 258-3877.

OKLAHOMA: Tulsa: 7615 East 63rd Place, 3 Memorial Place, Tulsa, OK 74133, (918) 250-0633.

OREGON: Beaverton: 6700 SW 105th St., Suite 110, Beaverton, OR 97005, (503) 643-6758.

PENNSYLVANIA: Ft. Washington: 260 New York Dr., Ft. Washington, PA 19034, (215) 643-6450; **Coraopolis:** 420 Rouser Rd., 3 Airport Office Park, Coraopolis, PA 15108, (412) 771-8550.

PUERTO RICO: Hato Rey: Mercantil Plaza Bldg., Suite 505, Hato Rey, PR 00919, (809) 753-8700.

TEXAS: Austin: 12501 Research Blvd., P.O. Box 2909, Austin, TX 78723, (512) 250-7655; **Richardson:** 1001 E. Campbell Rd., Richardson, TX 75080, (214) 680-5082; **Houston:** 9100 Southwest Frwy., Suite 237, Houston, TX 77036, (713) 778-6592; **San Antonio:** 1000 Central Parkway South, San Antonio, TX 78232, (512) 496-1779.

UTAH: Murray: 5201 South Green SE, Suite 200, Murray, UT 84107, (801) 266-8972.

VIRGINIA: Fairfax: 3001 Prosperity, Fairfax, VA 22031, (703) 849-1400.

WASHINGTON: Redmond: 5010 148th NE, Bldg B, Suite 107, Redmond, WA 98052, (206) 881-3080.

WISCONSIN: Brookfield: 450 N. Sunny Slope, Suite 150, Brookfield, WI 53005, (414) 785-7140.

CANADA: Nepean: 301 Moodie Drive, Mallorn Center, Nepean, Ontario, Canada, K2H9C4, (613) 726-1970. **Richmond Hill:** 280 Centre St. E., Richmond Hill L4C1B1, Ontario, Canada (416) 884-9181; **St. Laurent:** Ville St. Laurent Quebec, 9460 Trans Canada Hwy., St. Laurent, Quebec, Canada H4S1R7, (514) 334-3635.

ARGENTINA: Texas Instruments Argentina - S.A.I.C.F.: Esmeralda 130, 15th Floor, 1035 Buenos Aires, Argentina, 1 + 394-3008.

AUSTRALIA (& NEW ZEALAND): Texas Instruments Australia Ltd.: 6-10 Talavera Rd., North Ryde (Sydney), New South Wales, Australia 2113, 2 + 887-1122; 5th Floor, 418 St. Kilda Road, Melbourne, Victoria, Australia 3004, 3 + 267-4677; 171 Philip Highway, Elizabeth, South Australia 5112, 8 + 255-2066.

AUSTRIA: Texas Instruments Ges.m.b.H.: Industriestrabe B/16, A-2345 Brunn/Gebirge, 2236-846210.

BELGIUM: Texas Instruments N.V. Belgium S.A.: Mercure Centre, Raketstraat 100, Rue de la Fusee, 1130 Brussels, Belgium, 2/720.80.00.

BRAZIL: Texas Instruments Electronicos do Brasil Ltda.: Rua Paes Leme, 524-7 Andar Pinheiros, 05424 Sao Paulo, Brazil, 0815-6166.

DENMARK: Texas Instruments A/S, Mairelundvej 46E, DK-2730 Herlev, Denmark, 2 - 91 74 00.

FINLAND: Texas Instruments Finland OY: Teollisuuskatu 19D 00511 Helsinki 51, Finland, (90) 701-3133.

FRANCE: Texas Instruments France: Headquarters and Prod. Plant, BP 05, 06270 Villeneuve-Loubet, (93) 20-01-01; Paris Office, BP 67 8-10 Avenue Morane-Saulnier, 78141 Velizy-Villacoublay, (3) 946-97-12; Lyon Sales Office, L'Oree D'Ecully, Batiment B, Chemin de la Forestiere, 69130 Ecully, (7) 833-04-40; Strasbourg Sales Office, Le Sebastopol 3, Quai Kleber, 67055 Strasbourg Cedex, (88) 22-12-66; Rennes, 23-25 Rue du Puits Mauger, 35100 Rennes, (99) 31-54-86; Toulouse Sales Office, Le Peripole—2, Chemin du Pigeonnier de la Cepiere, 31100 Toulouse, (61) 44-18-19; Marseille Sales Office, Nolily Paradis—146 Rue Paradis, 13006 Marseille, (91) 37-25-30.

TEXAS INSTRUMENTS

Creating useful products and services for you

GERMANY (Fed. Republic of Germany): Texas Instruments Deutschland GmbH: Haggertystrasse 1, D-8050 Freising, 8161 + 80-4591; Kurfuerstendamm 195/196, D-1000 Berlin 15, 30 + 882-7365; III, Hagen 43/Kibbelstrasse, 19, D-4300 Essen, 201-24250; Frankfurter Allee 6-8, D-6236 Eschborn 1, 06196 + 8070; Hamburgerstrasse 11, D-2000 Hamburg 76, 040 + 220-1154, Kirchhorsterstrasse 2, D-3000 Hannover 51, 511 + 648021; Maybachstrabe 11, D-7302 Ostfildern 2-Nelingen, 711 + 547001; Mixikoring 19, D-2000 Hamburg 60, 40 + 637 + 0061; Postfach 1309, Roonstrasse 16, D-5400 Koblenz, 261 + 35044.

HONG KONG (+ PEOPLES REPUBLIC OF CHINA): Texas Instruments Asia Ltd., 8th Floor, World Shipping Ctr., Harbour City, 7 Canton Rd., Kowloon, Hong Kong, 3 + 722-1223.

IRELAND: Texas Instruments (Ireland) Limited: Brewery Rd., Stillorgan, County Dublin, Eire, 1 831311.

ITALY: Texas Instruments Semiconduttori Italia Spa: Viale Delle Scienze, 1, 02015 Cittaducale (Rieti), Italy, 746 694.1; Via Salaria KM 24 (Palazzo Cosma), Monterotondo Scalo (Rome), Italy, 6 + 9003241; Viale Europa, 38-44, 20093 Cologno Monzese (Milano), 2 2532541; Corso Svizzera, 185, 10100 Torino, Italy, 11 774545; Via J. Barozzi 6, 40100 Bologna, Italy, 51 355851.

JAPAN: Texas Instruments Asia Ltd.: 4F Aoyama Fuji Bldg., 6-12, Kita Aoyama 3-Chome, Minato-ku, Tokyo, Japan 107, 3-498-2111; Osaka Branch, 5F, Nissho Iwai Bldg., 30 Imabashi 3- Chome, Higashi-ku, Osaka, Japan 541, 06-204-1881; Nagoya Branch, 7F Daini Toyota West Bldg., 10-27, Meieki 4-Chome, Nakamura-ku Nagoya, Japan 450, 52-583-8691.

KOREA: Texas Instruments Supply Co.: 3rd Floor, Samon Bldg., Yuksam-Dong, Gangnam-ku, 135 Seoul, Korea, 2 + 462-8001.

MEXICO: Texas Instruments de Mexico S.A.: Mexico City, AV Reforma No. 450 — 10th Floor, Mexico, D.F., 06600, 5 + 514-3003.

MIDDLE EAST: Texas Instruments: No. 13, 1st Floor Mannai Bldg., Diplomatic Area, P.O. Box 26335, Manama Bahrain, Arabian Gulf, 973 + 274681.

NETHERLANDS: Texas Instruments Holland B.V., P.O. Box 12995, (Bullewijk) 1100 CB Amsterdam, Zuid-Oost, Holland 20 + 5602911.

NORWAY: Texas Instruments Norway A/S: PB106, Refstad 131, Oslo 1, Norway, (2) 155090.

PHILIPPINES: Texas Instruments Asia Ltd.: 14th Floor, Ba- Lepanto Bldg., 8747 Paseo de Roxas, Makati, Metro Manila, Philippines, 2 + 8188987.

PORTUGAL: Texas Instruments Equipamento Electronico (Portugal), Lda.: Rua Eng. Frederico Ulrich, 2650 Moreira Da Maia, 4470 Maia, Portugal, 2-948-1003.

SINGAPORE (+ INDIA, INDONESIA, MALAYSIA, THAILAND): Texas Instruments Asia Ltd.: 12 Lorong Bakar Batu, Unit 01-02, Kolam Ayer Industrial Estate, Republic of Singapore, 747-2255.

SPAIN: Texas Instruments Espana, S.A.: C/Jose Lazaro Galdiano No. 6, Madrid 16, 1/458.14.58.

SWEDEN: Texas Instruments International Trade Corporation (Sverigefilialen): Box 39103, 10054 Stockholm, Sweden, 8 - 235480.

SWITZERLAND: Texas Instruments, Inc., Reidstrasse 6, CH-8953 Dietikon (Zuerich) Switzerland, 1-740 2220.

TAIWAN: Texas Instruments Supply Co.: Room 903, 205 Tun Hwan Rd., 71 Sung-Kiang Road, Taipei, Taiwan, Republic of China, 2 + 521-9321.

UNITED KINGDOM: Texas Instruments Limited: Manton Lane, Bedford, MK41 7PA, England, 0234 67466; St. James House, Wellington Road North, Stockport, SK4 2RT, England, 61 + 442-7162. BK

TEXAS INSTRUMENTS

Creating useful products
and services for you.

April 1985
Printed in U.S.A.

SNYS001