National Semiconductor

May 1991

NS32532-20/NS32532-25/NS32532-30
High-Performance 32-Bit Microprocessor

General Description

The NS32532 is a high-performance 32-bit microprocessor in the Series 32000® family. It is software compatible with the previous microprocessors in the family but with a greatly enhanced internal implementation.

The high-performance specifications are the result of a four-stage instruction pipeline, on-chip instruction and data caches, on-chip memory management unit and a significantly increased clock frequency. In addition, the system interface provides optimal support for applications spanning a wide range, from low-cost, real-time controllers to highly sophisticated, general purpose multiprocessor systems.

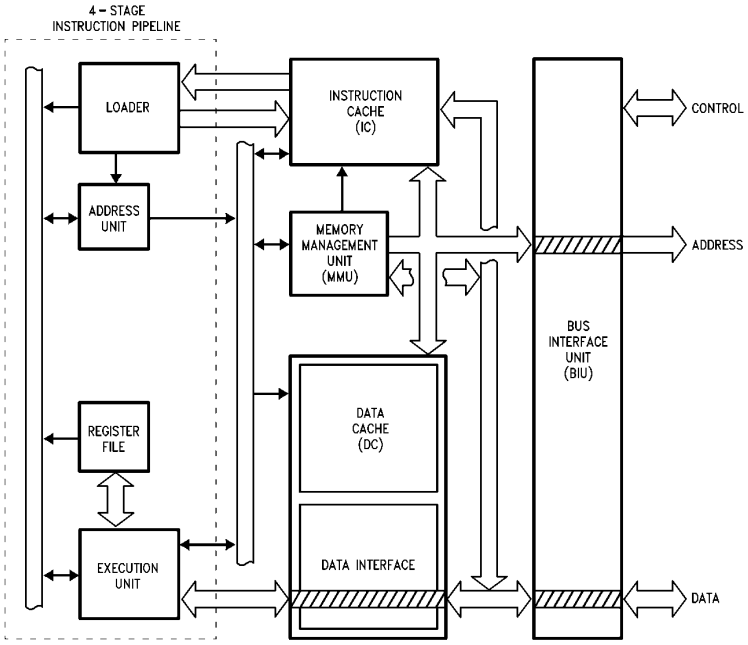
The NS32532 integrates more than 370,000 transistors fabricated in a 1.25 μm double-metal CMOS technology. The advanced technology and mainframe-like design of the device enable it to achieve more than 10 times the throughput of the NS32032 in typical applications.

In addition to generally improved performance, the NS32532 offers much faster interrupt service and task switching for real-time applications.

Features

- Software compatible with the Series 32000 family
- 32-bit architecture and implementation
- 4-GByte uniform addressing space
- On-chip memory management unit with 64-entry translation look-aside buffer
- 4-Stage instruction pipeline
- 512-Byte on-chip instruction cache
- 1024-Byte on-chip data cache
- High-performance bus
  - — Separate 32-bit address and data lines
  - — Burst mode memory accessing
  - — Dynamic bus sizing
- Extensive multiprocessing support
- Floating-point support via the NS32381 or NS32580
- 1.25 μm double-metal CMOS technology
- 175-pin PGA package

Block Diagram



TL/EE/9354–1

FIGURE 1

Series 32000® and TRI-STATE® are registered trademarks of National Semiconductor Corporation.

©1995 National Semiconductor Corporation TL/EE/9354

RRD-B30M105/Printed in U. S. A.

NS32532-20/NS32532-25/NS32532-30 High-Performance 32-Bit Microprocessor

Table of Contents

Table of Contents (Continued)

List of Illustrations

List of Illustrations (Continued)

List of Tables

1.0 Product Introduction

The NS32532 is an extremely sophisticated microprocessor in the Series 32000 family with a full 32-bit architecture and implementation optimized for high-performance applications.

By employing a number of mainframe-like features, the device can deliver 15 MIPS peaks performance with no wait states at a frequency of 30 MHz.

The NS32532 is fully software compatible will all the other Series 32000 CPUs. The architectural features of the Series 32000 family and particularly the NS32532 CPU, are described briefly below.

Powerful Addressing Modes. Nine addressing modes available to all instructions are included to access data structures efficiently.

Data Types. The architecture provides for numerous data types, such as byte, word, doubleword, and BCD, which may be arranged into a wide variety of data structures.

Symmetric Instruction Set. While avoiding special case instructions that compilers can't use, the Series 32000 architecture incorporates powerful instructions for control operations, such as array indexing and external procedure calls, which save considerable space and time for compiled code.

Memory-to-Memory Operations. The Series 32000 CPUs represent two-address machines. This means that each operand can be referenced by any one of the addressing modes provided.

This powerful memory-to-memory architecture permits memory locations to be treated as registers for all usefull operations. This is important for temporary operands as well as for context switching.

Memory Management. The NS32532 on-chip memory management unit provides advanced operating system support functions, including dynamic address translation, virtual memory management, and memory protection.

Large, Uniform Addressing. The NS32532 has 32-bit address pointers that can address up to 4 gigabytes without requiring any segmentation; this addressing scheme provides flexible memory management without added-on expense.

Modular Software Support. Any software package for the Series 32000 family can be developed independent of all other packages, without regard to individual addressing. In addition, ROM code is totally relocatable and easy to access, which allows a significant reduction in hardware and software costs.

Software Processor Concept. The Series 32000 architecture allows future expansions of the instruction set that can be executed by special slave processors, acting as extensions to the CPU. This concept of slave processors is unique to the Series 32000 family. It allows software compatibility even for future components because the slave hardware is transparent to the software. With future advances in semiconductor technology, the slaves can be physically integrated on the CPU chip itself.

To summarize, the architectural features cited above provide three primary performance advantages and characteristics:

- High-level language support
- Easy future growth path
- Application flexibility

2.0 Architectural Description

2.1 REGISTER SET

The NS32532 CPU has 28 internal registers grouped according to functions as follows: 8 general purpose, 7 address, 1 processor status, 1 configuration, 7 memory management and 4 debug. All registers are 32 bits wide except for the module and processor status, which are each 16 bits wide. *Figure 2-1* shows the NS32532 internal registers.

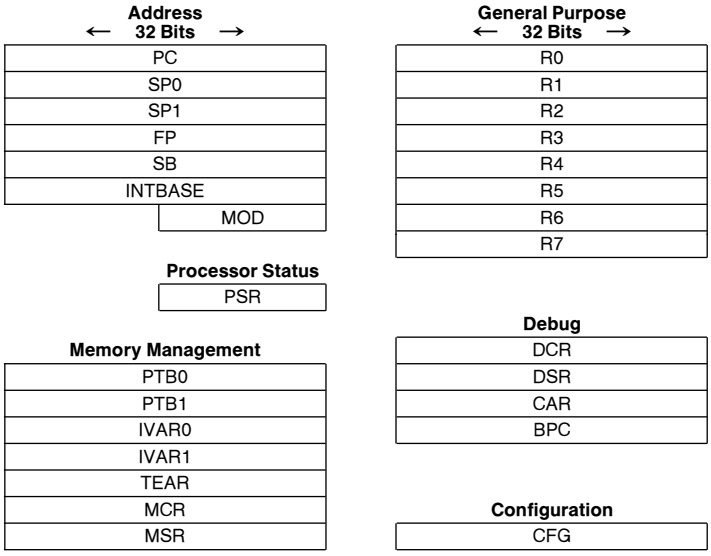


FIGURE 2-1. NS32532 Internal Registers

2.0 Architectural Description (Continued)

2.1.1 General Purpose Registers

There are eight registers (R0–R7) used for satisfying the high speed general storage requirements, such as holding temporary variables and addresses. The general purpose registers are free for any use by the programmer. They are 32 bits in length. If a general purpose register is specified for an operand that is eight or 16 bits long, only the low part of the register is used; the high part is not referenced or modified.

2.1.2 Address Registers

The seven address registers are used by the processor to implement specific address functions. A description of them follows.

PC—Program Counter. The PC register is a pointer to the first byte of the instruction currently being executed. The PC is used to reference memory in the program section.

SP0, SP1—Stack Pointers. 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. It is used primarily for storing temporary data, and 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 is used by normal user programs to hold temporary data and subroutine return information.

When a reference is made to the selected Stack Pointer (see PSR S-bit), the terms 'SP Register' or 'SP' are used. SP refers to either SP0 or SP1, depending on the setting of the S bit in the PSR register. If the S bit in the PSR is 0, SP refers to SP0. If the S bit in the PSR is 1 then SP refers to SP1.

The NS32532 also allows the SP1 register to be directly loaded and stored using privileged forms of the LPRi and SPRi instructions, regardless of the setting of the PSR S-bit. When SP1 is accessed in this manner, it is referred to as 'USP Register' or simply 'USP'.

Stacks in the Series 32000 family grow downward in memory. A Push operation pre-decrements the Stack Pointer by the operand length. A Pop operation post-increments the Stack Pointer by the operand length.

FP—Frame Pointer. The FP 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 restored on procedure termination with the EXIT instruction.

The frame pointer holds the address in memory occupied by the old contents of the frame pointer.

SB—Static Base. The SB register points to the global variables of a 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.

INTBASE—Interrupt Base. The INTBASE register holds the address of the dispatch table for interrupts and traps (Section 3.2.1).

MOD—Module. The MOD 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 64 kbytes of memory.

2.1.3 Processor Status Register

The Processor Status Register (PSR) holds status information for the microprocessor.

The PSR is sixteen bits long, divided into two eight-bit halves. The low order eight bits are accessible to all programs, but the high order eight bits are accessible only to programs executing in Supervisor Mode.

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 set to 1, a TRC trap is executed after every instruction (Section 3.3.1).

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, when both operands are interpreted as unsigned integers. Otherwise, it is set to ''0''. In Floating-Point comparisons, this bit is always cleared.

V The V-bit enables generation of a trap (OVF) when an integer arithmetic operation overflows.

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 the second operand is less than the first operand, when both operands are interpreted as signed integers. 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 processor is said to be in Supervisor Mode; when U = 1 the processor is said to

15							8	7							0
				I	P	S	U	N	Z	F	V		L	T	C

FIGURE 2-2. Processor Status Register (PSR)

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. A Supervisor 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 bit is automatically cleared on interrupts and traps. It may have a setting of 0 (use the SP0 register) or 1 (use the SP1 register).

P The P bit prevents a TRC trap from occuring more than once for an instruction (Section 3.3.1). It may have a setting of 0 (no trace pending) or 1 (trace pending).

I If I = 1, then all interrupts will be accepted. If I = 0, only the NMI interrupt is accepted. Trap enables are not affected by this bit.

2.1.4 Configuration Register

The Configuration Register (CFG) is 32 bits wide, of which ten bits are implemented. The implemented bits enable various operating modes for the CPU, including vectoring of interrupts, execution of slave instructions, and control of the on-chip caches. In the NS32332 bits 4 through 7 of the CFG register selected between the 16-bit and 32-bit slave protocols and between 512-byte and 4-Kbyte page sizes. The NS32532 supports only the 32-bit slave protocol and 4-Kbyte page size: consequently these bits are forced to 1.

When the CFG register is loaded using the LPRi instruction, bits 14 through 31 should be set to 0. Bits 4 through 7 are ignored during loading, and are always returned as 1's when CFG is stored via the SPRi instruction. When the SETCFG instruction is executed, the contents of the CFG register bits 0 through 3 are loaded from the instruction's short field, bits 4 through 7 are ignored and bits 8 through 13 are forced to 0.

The format of the CFG register is shown in *Figure 2-3*. The various control bits are described below.

I Interrupt vectoring. This bit controls whether maskable interrupts are handled in nonvectored (I=0) or vectored (I=1) mode. Refer to Section 3.2.3 for more information.

F Floating-point instruction set. This bit indicates whether a floating-point unit (FPU) is present to execute floating-point instructions. If this bit is 0 when the CPU executes a floating-point instruction, a Trap (UND) occurs. If this bit is 1, then the CPU transfers the instruction and any necessary operands to the FPU using the slave-processor protocol described in Section 3.1.4.1.

M Memory management instruction set. This bit enables the execution of memory management instructions. If this bit is 0 when the CPU executes an LMR, SMR, RDVAL, or WRVAL instruction, a Trap (UND) occurs. If this bit is 1, the CPU executes LMR, SMR, RDVAL, and WRVAL instructions using the on-chip MMU.

C Custom instruction set. This bit indicates whether a custom slave processor is present to execute custom instructions. If this bit is 0 when the CPU executes a custom instruction, a Trap (UND) occurs. If this bit is 1, the CPU transfers the instruction and any necessary operands to the custom slave processor using the slave-processor protocol described in Section 3.1.4.1.

DE Direct-Exception mode enable. This bit enables the Direct-Exception mode for processing exceptions. When this mode is selected, the CPU response time to interrupts and other exceptions is significantly improved. Refer to Section 3.2.1 for more information.

DC Data Cache enable. This bit enables the on-chip Data Cache to be accessed for data reads and writes. Refer to Section 3.4.2 for more information.

LDC Lock Data Cache. This bit controls whether the contents of the on-chip Data Cache are locked to fixed memory locations (LDC=1), or updated when a data read is missing from the cache (LDC=0).

IC Instruction Cache enable. This bit enables the on-chip Instruction Cache to be accessed for instruction fetches. Refer to Section 3.4.1 for more information.

LIC Lock Instruction Cache. This bit controls whether the contents of the on-chip Instruction Cache are locked to fixed memory locations (LIC=1), or updated when an instruction fetch is missing from the cache (LIC=0).

PF Pipelined Floating-point execution. This bit indicates whether the floating-point unit uses the pipelined slave protocol. When PF is 1 the pipelined protocol is selected. PF is ignored if the F bit is 0. Refer to Section 3.1.4.2 for more information.

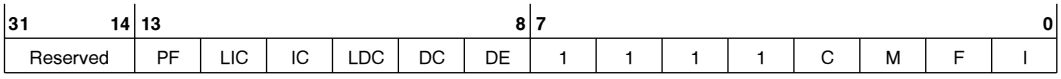


FIGURE 2-3. Configuration Register (CFG) Bits
13 to 31 are Reserved; Bits 4 to 7 are Forced to 1.

2.1.5 Memory Management Registers

The NS32532 provides 7 registers to support memory management functions. They are accessed by means of the LMR and SMR instructions. All of them can be read and written except IVAR0 and IVAR1 that are write-only. A description of the memory management registers is given in the following sections.

PTB0, PTB1—Page Table Base Pointers. The PTBn registers hold the physical addresses of the level-1 page tables used in address translation. The least significant 12 bits are permanently zero, so that each register always points to a 4-Kbyte boundary in memory.

When either PTB0 or PTB1 is loaded by executing an LMR instruction, the MMU automatically invalidates all entries in the TLB that had been translated using the old value in the selected PTBn register.

The format of the PTBn registers is shown in *Figure 2-4*.

31 12	11 0
Base Address	000000000000

FIGURE 2-4. Page Table Base Registers (PTBn)

IVAR0, IVAR1—Invalidate Virtual Address. The Invalidate Virtual Address registers are write-only registers. When a virtual address is written to IVAR0 or IVAR1 using the LMR instruction, the translation for that virtual address is purged, if present, from the TLB. This must be done whenever a Page Table Entry has been changed in memory, since the TLB might otherwise contain an incorrect translation value.

Another technique for purging TLB entries is to load a PTBn register. Turning off translation (clearing the MCR TU and/or TS bits) does not purge any entries from the TLB.

TEAR—Translation Exception Address Register. The TEAR register is loaded by the on-chip MMU when a translation exception occurs. It contains the 32-bit virtual address that caused the translation exception.

TEAR is not updated if a page fault is detected while prefetching an instruction that is not executed because the previous instruction caused a trap.

MCR—Memory Management Control. The MCR register controls the operation of the MMU. Only four bits are implemented. Bits 4 to 31 are reserved for future use and must be loaded with zeroes.

When MCR is read as a 32-bit word, bits 4 to 31 are returned as zeroes. The format of MCR is shown in *Figure 2-5*. Details on the control bits are given below.

TU Translate User. While this bit is 1, address translation is enabled for User-Mode memory references. While this bit is 0, address translations is disabled for User-Mode memory references.

TS Translate Supervisor. While this bit is 1, address translation is enabled for Supervisor Mode memory references. While this bit is 0, address translation is disabled for Supervisor-Mode memory references.

DS Dual Space. While this bit is 1, then PTB1 contains the level-1 page table base address of all addresses specified in User-Mode, and PTB0 contains the level-1 page table base address of all addresses specified in Supervisor Mode. While this bit is 0, then PTB0 contains the level-1 page table base address of all addresses specified in both User and Supervisor Modes.

AO Access Level Override. When this bit is set to 1, User-Mode accesses are given Supervisor Mode privilege.

31 4	3			0
Reserved	AO	DS	TS	TU

FIGURE 2-5. Memory Management Control Register (MCR)

MSR—Memory Management Status. The MSR register provides status information related to the occurrence of a translation exception. Only eight bits are implemented. Bits 8 to 31 are ignored when MSR is loaded and are returned as zeroes when it is read as a 32-bit word. MSR is only updated by the MMU when a protection violation or page fault is detected while translating an address for a reference required to execute an instruction. It is not updated if a page fault is detected during either an operand or an instruction prefetch, if the data being prefetched is not needed due to a change in the instruction execution sequence. The format of MSR is shown in *Figure 2-6*. Details on the function of each bit are given below.

TEX Translation Exception. This two-bit field specifies the cause of the current address translation exception. (Trap(ABT)). Combinations appearing in this field are summarized below.

00 No Translation Exception

01 First Level PTE Invalid

10 Second Level PTE Invalid

11 Protection Violation

During address translation, if a protection violation and an invalid PTE are detected at the same time, the TEX field is set to indicate a protection violation.

DDT Data Direction. This bit indicates the direction of the transfer that the CPU was attempting when the translation exception occurred.

DDT = 0 => Read Cycle

DDT = 1 => Write Cycle

UST User/Supervisor. This bit indicates whether the Translation Exception was caused by a User-Mode or Supervisor Mode reference. If UST is 1, then the exception was caused by a User-Mode reference; otherwise it was caused by a Supervisor Mode reference.

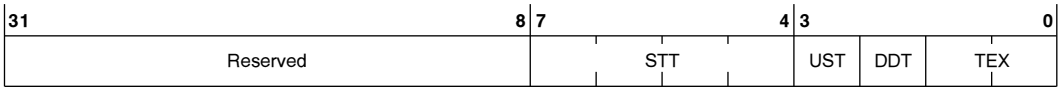


FIGURE 2-6. Memory Management Status Register (MSR)

STT CPU Status. This four bit field is set on an address translation exception according to the following encodings.

1000 Sequential Instruction Fetch

1001 Non-Sequential Instruction Fetch

1010 Data Transfer

1011 Read Read-Modify-Write Operand

1100 Read for Effective Address

If a reference for an Interrupt-Acknowledge or End-of-Interrupt bus cycle (either Master of Cascaded) causes a Translation Exception, then the value of the STT-field is undefined.

2.1.6 Debug Registers

The NS32532 contains 4 registers dedicated for debugging functions.

These registers are accessed using privileged forms of the LPRi and SPRi instructions.

DCR—Debug Condition Register. The DCR Register enables detection of debug conditions. The format of the DCR is shown in *Figure 2-7;* the various bits are described below. A debug condition is enabled when the related bit is set to 1.

CBE0 Compare Byte Enable 0; when set, BYTE0 of an aligned double-word is included in the address comparison

CBE1 Compare Byte Enable 1; when set, BYTE1 of an aligned double-word is included in the address comparison

CBE2 Compare Byte Enable 2; when set, BYTE2 of an aligned double-word is included in the address comparison

CBE3 Compare Byte Enable 3; when set, BYTE3 of an aligned double-word is included in the address comparison

VNP Compare virtual address (VNP = 1) or physical address (VNP = 0)

CWR Address-compare enable for write references

CRD Address-compare enable for read references

CAE Address-compare enable

TR Enable Trap (DBG) when a debug condition is detected

PCE PC-match enable

UD Enable debug conditions in User-Mode

SD Enable debug conditions in Supervisor Mode

DEN Enable debug conditions

The following 2 bits control testing features that can be used during initial system debugging. These features are unique to the NS32532 implementation of the Series 32000 architecture; as such, they may not be supported in future implementations. For normal operation these 2 bits should be set to 0.

SI Single-Instruction mode enable. This bit, when set to 1, inhibits the overlapping of instruction's execution.

BCP Branch Condition Prediction disable. When this bit is 1, the branch prediction mechanism is disabled. See Section 3.1.3.1.

DSR—Debug Status Register. The DSR Register indicates debug conditions that have been detected. When the CPU detects an enabled debug condition, it sets the corresponding bit (BC, BEX, BCA) in the DSR to 1. When an address-compare condition is detected, then the RD-bit is loaded to indicate whether a read or write reference was performed. Software must clear all the bits in the DSR when appropriate. The format of the DSR is shown in *Figure 2-8;* the various fields are described below.

RD Indicates whether the last address-compare condition was for a read (RD = 1) or write (RD = 0) reference

BPC PC-match condition detected

BEX External condition detected

BCA Address-compare condition detected

Note 1: The content of the DSR register is not defined if a debug condition was detected on a floating-point instruction in pipelined mode and a trap was generated by a previous floating-point instruction.

Note 2: If an address compare is detected on a read and a write for the same instruction then the RD-bit will remain clear.

CAR—Compare Address Register. The CAR Register contains the address that is compared to operand reference addresses to detect an address-compare condition. The address must be double-word aligned; that is, the two least-significant bits must be 0. The CAR is 32 bits wide.

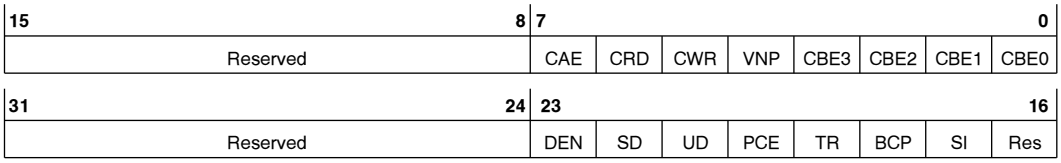


FIGURE 2-7. Debug Condition Register (DCR)

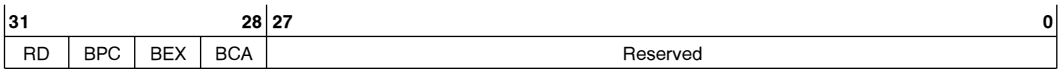


FIGURE 2-8. Debug Status Register (DSR)

BPC—Breakpoint Program Counter. The BPC Register contains the address that is compared with the PC contents to detect a PC-match condition. The BPC Register is 32 bits wide.

2.2 MEMORY ORGANIZATION

The NS32532 implements full 32-bit virtual addresses. This allows the CPU to access up to 4 Gbytes of virtual memory. The memory is a uniform linear address space. Memory locations are numbered sequentially starting at zero and ending at $2^{32}-1$. The number specifying a memory location is called an address. The contents of each memory location is a byte consisting of eight 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. Also, when data is shown vertically, the lowest address is at the top of a diagram and the highest address at the bottom of the diagram. When bits are numbered in a diagram, the least significant bit is given the number zero, and is shown at the right of the diagram. Bits are numbered in increasing significance and toward the left.

7	0
A	

Byte at Address A

Two contiguous bytes are called a word. Except where noted, 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.

15	8	7	0
A+1		A	
MSB		LSB	

Word at Address A

Two contiguous words are called a double-word. Except where noted, 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 higher. 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.

31	24	23	16	15	8	7	0
A+3		A+2		A+1		A	
MSB						LSB	

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.

2.2.1 Address Mapping

Figure 2-9 shows the NS32532 address mapping.

The NS32532 supports the use of memory-mapped peripheral devices and coprocessors. Such memory-mapped devices can be located at arbitrary locations in the address space except for the upper 8 Mbytes of virtual memory (addresses between FF800000 (hex) and FFFFFFFF (hex), inclusive), which are reserved by National Semiconductor Corporation. Nevertheless, it is recommended that high-performance peripheral devices and coprocessors be located in a specific 8 Mbyte region of virtual memory (addresses between FF000000 (hex) and FF7FFFFF (hex), inclusive), that is dedicated for memory-mapped I/O. This is because the NS32532 detects references to the dedicated locations and serializes reads and writes. See Section 3.1.3.3. When making I/O references to addresses outside the dedicated region, external hardware must indicate to the NS32532 that special handling is required.

In this case a small performance degradation will also result. Refer to Section 3.1.3.2 for more information on memory-mapped I/O.

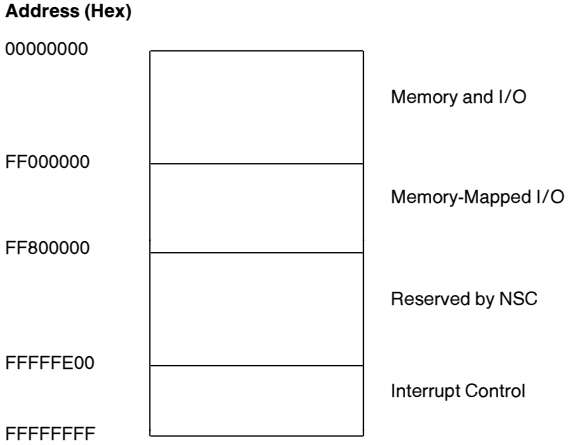


FIGURE 2-9. NS32532 Address Mapping

2.0 Architectural Description (Continued)

2.3 MODULAR SOFTWARE SUPPORT

The NS32532 provides special support for software modules and modular programs.

Each module in a NS32532 software environment consists of three components:

1. Program Code Segment.

   This segment contains the module's code and constant data.

2. Static Data Segment.

   Used to store variables and data that may be accessed by all procedures within the module.

3. Link Table.

   This component contains two types of entries: Absolute Addresses and Procedure Descriptors.

   An Absolute Address is used in the external addressing mode, in conjunction with a displacement and the current MOD Register contents to compute the effective address of an external variable belonging to another module.

   The Procedure Descriptor is used in the call external procedure (CXP) instruction to compute the address of an external procedure.

Normally, the linker program specifies the locations of the three components. The Static Data and Link Table typically reside in RAM; the code component can be either in RAM or in ROM. The three components can be mapped into non-contiguous 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 handle the transfer of control from one module to another, the NS32532 uses a module table in memory and two registers in the CPU.

The Module Table is located within the first 64 kbytes of virtual memory. This table contains a Module Descriptor (also called a Module Table Entry) 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:

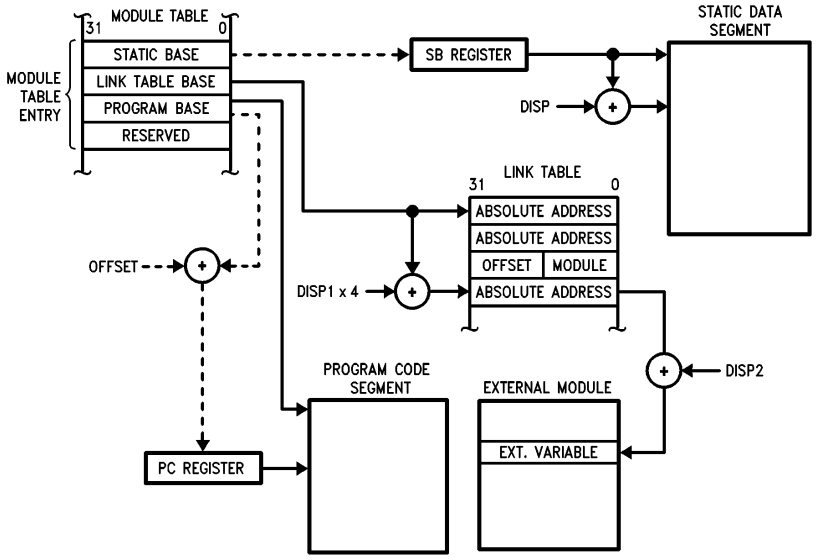
- The Static Base entry contains the address of the beginning of the module's static data segment.
- The Link Table Base points to the beginning of the module's Link Table.
- The Program Base is the address of the beginning of the code and constant data for the module.
- A fourth entry is currently unused but reserved.

The MOD Register in the CPU contains the address of the Module Descriptor for the currently executing module.

The Static Base Register (SB) contains a copy of the Static Base entry in the Module Descriptor of the currently executing module, i.e., it points to the beginning of the current module's static data area.

This register is implemented in the CPU for efficiency purposes. By having a copy of the static base entry or chip, the CPU can avoid reading it from memory each time a data item in the static data segment is accessed.

In an NS32532 software environment modules need not be linked together prior to loading. As modules are loaded, a linking loader simply updates the Module Table and fills the Link 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. *Figure 2-10* shows a typical NS32532 run-time environment.



TL/EE/9354–2

Note: Dashed lines indicate information copied to registers during transfer of control between modules.

FIGURE 2-10. NS32532 Run-Time Environment

2.0 Architectural Description (Continued)

2.4 MEMORY MANAGEMENT

The Memory Mangement Unit of the NS32532 provides support for demand-paged virtual memory. The MMU translates 32-bit virtual addresses into 32-bit physical addresses. The page size is 4096 bytes.

The mapping from virtual to physical addresses is defined by means of sets of tables in physical memory. These tables are found by the MMU using one of its two Page Table Base registers: PTB0 or PTB1. Which register is used depends on the currently selected address space. See Section 2.4.2.

Translation efficiency is improved by means of an on-chip 64-entry translation look-aside buffer (TLB). Refer to Section 3.4.4 for details.

If the MMU detects a protection violation or page fault while translating an address for a reference required to execute an instruction, a translation exception (Trap (ABT)) will result.

2.4.1 Page Tables Structure

The page tables are arranged in a two-level structure, as shown in *Figure 2-11*. Each of the MMU's PTBn registers may point to a Level-1 page table. Each entry of the Level-1 page table may in turn point to a Level-2 page table. Each Level-2 page table entry contains translation information for one page of the virtual space.

The Level-1 page table must remain in physical memory while the PTBn register contains its address and translation is enabled. Level-2 Page Tables need not reside in physical memory permanently, but may be swapped into physical memory on demand as is done with the pages of the virtual space.

The Level-1 Page Table contains 1024 32-bit Page Table Entries (PTE's) and therefore occupies 4 Kbytes. Each entry of the Level-1 Page Table contains a field used to construct the physical base address of a Level-2 Page Table. This field is a 20-bit PFN field, providing bits 12–31 of the physical address. The remaining bits (0–11) are assumed zero, placing a Level-2 Page Table always on a 4-Kbyte (page) boundary.

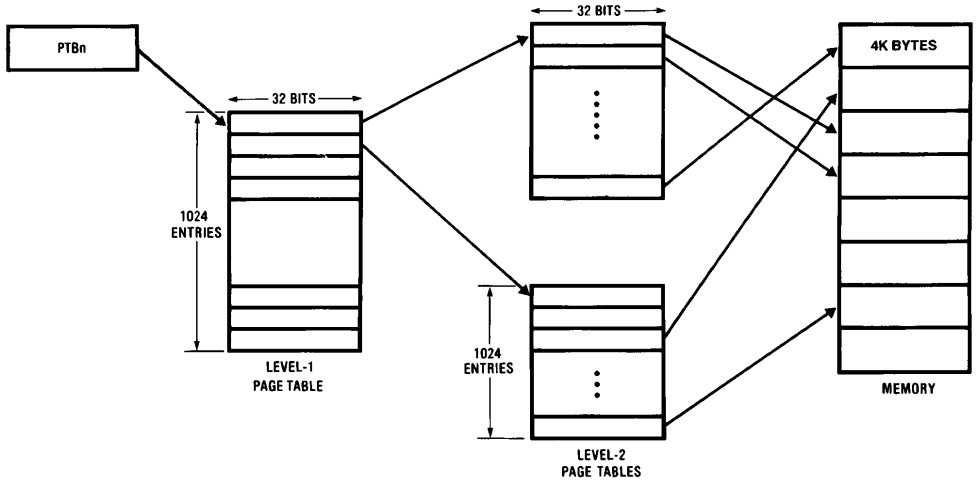
Level-2 Page Tables contain 1024 32-bit Page Table entries, and so occupy 4 Kbytes (1 page). Each Level-2 Page Table Entry points to a final 4-Kbyte physical page frame. In other words, its PFN provides the Page Frame Number portion (bits 12–31) of the translated address (*Figure 2-13*). The OFFSET field of the translated address is taken directly from the corresponding field of the virtual address.

2.4.2 Virtual Address Spaces

When the Dual Space option is selected for address translation in the MCR (Section 2.1.5) the on-chip MMU uses two maps: one for translating addresses presented to it in Supervisor Mode and another for User Mode addresses. Each map is referenced by the MMU using one of the two Page Table Base registers: PTB0 or PTB1. The MMU determines the map to be used by applying the following rules.

1) While the CPU is in Supervisor Mode (U/$\overline{S}$ pin = 0), the CPU is said to be generating virtual addresses belonging to Address Space 0, and the MMU uses the PTB0 register as its reference for looking up translations from memory.
2) While the CPU is in User Mode (U/$\overline{S}$ pin = 1), and the MCR DS bit is set to enable Dual Space translation, the CPU is said to be generating virtual addresses belonging to Address Space 1, and the MMU uses the PTB1 register to look up translations.
3) If Dual Space translation is not selected in the MCR, there is no Adress Space 1, and all virtual addresses generated in both Supervisor and User modes are considered by the MMU to be in Address Space 0. The privilege level of the CPU is used then only for access level checking.

Note: When the CPU executes a Dual-Space Move instruction (MOVUSi or MOVSUi), it temporarily enters User Mode by switching the state of the U/$\overline{S}$ pin. Accesses made by the CPU during this time are treated by the MMU as User-Mode accesses for both mapping and access level checking. It is possible, however, to force the MMU to assume Supervisor Mode privilege on such accesses by setting the Access Override (AO) bit in the MCR (Section 2.1.5).



TL/EE/9354–3

FIGURE 2-11. Two-Level Page Tables

2.4.3 Page Table Entry Formats

Figure 2-12 shows the formats of Level-1 and Level-2 Page Table Entries (PTE's).

The bits are defined as follows:

V Valid. The V bit is set and cleared only by software.

V = 1 => The PTE is valid and may be used for translation by the MMU.

V = 0 => The PTE does not represent a valid translation. Any attempt to use this PTE to translate and address will cause the MMU to generate an Abort trap.

PL Protection Level. This two-bit field establishes the types of accesses permitted for the page in both User Mode and Supervisor Mode, as shown in Table 2-1.

The PL field is modified only by software. In a Level-1 PTE, it limits the maximum access level allowed for all pages mapped through that PTE.

TABLE 2-1. Access Protection Levels

Mode	U/$\overline{S}$	Protection Level Bits (PL)			
		00	01	10	11
User	1	no access	no access	read only	full access
Supervisor	0	read only	full access	full access	full access

NU Not Used. These bits are reserved by National for future enhancements. Their values should be set to zero.

CI Cache Inhibit. This bit appears only in Level-2 PTE's. It is used to specify non-cacheable pages.

R Referenced. This is a status bit, set by the MMU and cleared by the operating system, that indicates whether the page mapped by this PTE has been referenced within a period of time determined by the operating system. It is intended to assist in implementing memory allocation strategies. In a Level-1 PTE, the R bit indicates only that the Level-2 Page Table has been referenced for a translation, without necessarily implying that the translation was successful. In a Level-2 PTE, it indicates that the page mapped by the PTE has been sucessfully referenced.

R = 1 => The page has been referenced since the R bit was last cleared.

R = 0 => The page has not been referenced since the R bit was last cleared.

M Modified. This is a status bit, set by the MMU whenever a write cycle is successfully performed to the page mapped by this PTE. It is initialized to zero by the operating system when the page is brought into physical memory.

M = 1 => The page has been modified since it was last brought into physical memory.

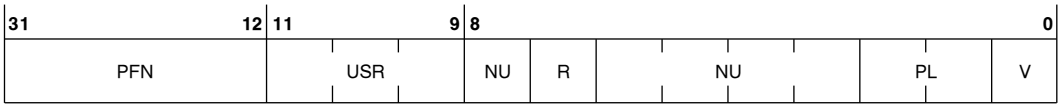
M = 0 => The page has not been modified since it was last brought into physical memory.

In Level-1 Page Table Entries, this bit position is undefined, and is unaltered.

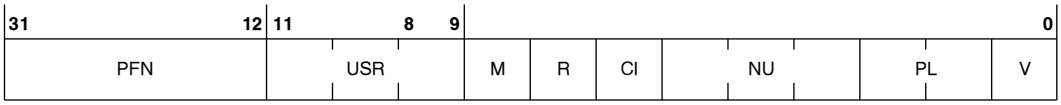
USR User bits. These bits are ignored by the MMU and their values are not changed.

They can be used by the user software.

PFN Page Frame Number. This 20-bit field provides bits 12–31 of the physical address. See *Figure 2-13*.

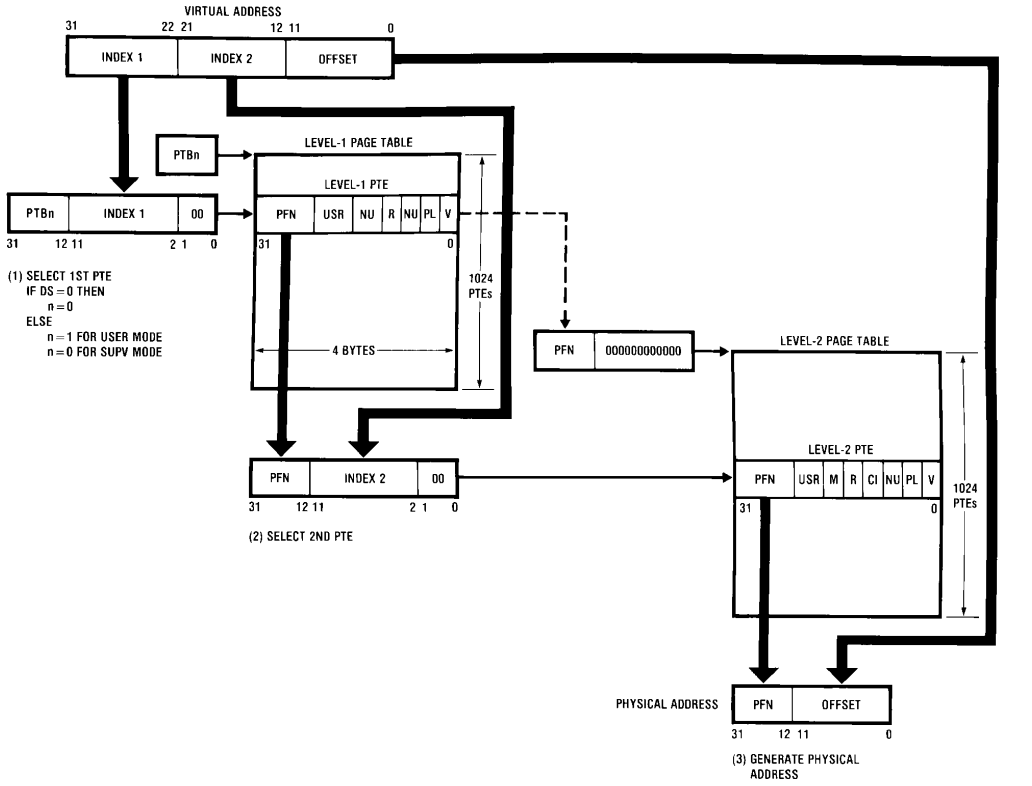


First Level PTE



Second Level PTE

FIGURE 2-12. Page Table Entries (PTE's)



TL/EE/9354–4

FIGURE 2-13. Virtual to Physical Address Translation

2.4.4 Physical Address Generation

When a virtual address is presented to the MMU and the translation information is not in the TLB, the MMU performs a page table lookup in order to generate the physical address.

The Page Table structure is traversed by the MMU using fields taken from the virtual address. This sequence is diagrammed in *Figure 2-13*.

Bits 12–31 of the virtual address hold the 20-bit Page Number, which in the course of the translation is replaced with the 20-bit Page Frame Number of the physical address. The virtual Page Number field is further divided into two fields, INDEX 1 and INDEX 2.

Bits 0–11 constitute the OFFSET field, which identifies a byte's position within the accessed page. Since the byte position within a page does not change with translation, this value is not used, and is simply echoed by the MMU as bits 0–11 of the final physical address.

The 10-bit INDEX 1 field of the virtual address is used as an index into the Level-1 Page Table, selecting one of its 1024 entries. The address of the entry is computed by adding INDEX 1 (scaled by 4) to the contents of the current Page Table Base register. The PFN field of that entry gives the base address of the selected Level-2 Page Table.

The INDEX 2 field of the virtual address (10 bits) is used as the index into the Level-2 Page Table, by adding it (scaled by 4) to the base address taken from the Level-1 Page Table Entry. The PFN field of the selected entry provides the entire Page Frame Number of the translated address.

The offset field of the virtual address is then appended to this frame number to generate the final physical address.

2.4.5. Address Translation Algorithm

The MMU either translates the 32-bit virtual address to a 32-bit physical address or generates an abort trap to report a translation error. The algorithm used by the MMU to perform the translation is compatible with that of the NS32382. Refer to Appendix C for differences between the two MMUs.

In the description that follows, the symbol 'U' takes the value 1 for a User-Mode memory reference. A reference is a User-Mode reference in the following cases:

1. The reference is performed while executing in User-Mode.
2. The reference is for the source operand of a MOVUS instruction.
3. The reference is for the destination operand of a MOVSU instruction.

The following notations are used in the algorithm.

- A||B ⟶ A concatenated with B
- A.B ⟶ B is a field inside register A
- (A) ⟶ object pointed to by address A
- (A).B ⟶ B field of the object pointed to by address A

2.0 Architectural Description (Continued)

Each access is associated with one of two Address Spaces (AS), defined as follows:

AS = U AND MCR.DS

If AS = 1, Page Table Base Register 1 (PTB1) is used to select the first-level page table. If AS = 0, PTB0 is used to select the first-level page table.

The access-level is a 2-bit value used to specify the privilege level of an access. It is determined as follows:

- BIT1 = U AND (NOT(MCR.A0))
- BIT0 = 1 for write, or read with 'RMW' status
  0 otherwise

START TRANSLATION:

If (U = 0 AND MCR.TS = 0 OR U = 1 AND MCR.TU = 0)
then
/* address translation disabled */
(physical address ← virtual address; CIOUT pin = 0);
/* Note: CIOUT = 0 in all MMU generated accesses */
else BEGIN /* (see also *Figure 2-13*) */
1. Select PTB:
- **If** (MCR.DS = 1 AND U = 1) **then**
— PTB = PTB1,
— AS = 1;
- **else** (PTB = PTB0, AS = 0);

2. Fetch first level PTE:
- PTE Pointer = PTB.BASE ADDRESS||INDEX1||00;
- PTE ← (PTE Pointer); /* Fetch PTE1 */
- Effective PL ← PTE.PL

3. Validate First Level PTE:
- **If** (PTE.PL < access level) **then**
- /* Protection Exception */
— TEAR ← virtual address,
— clock MSR with MSR.TEX = 11,
— terminate translation;
- **If** (PTE.V = 0) **then**
- /* PTE1 Invalid */
— TEAR ← virtual address,
— clock MSR with MSR.TEX = 01,
— terminate translation;
- **If** (PTE.R = 0) **then**
— Write a Byte (PTE Pointer) .R = 1;
- Effective PL ← PTE.PL

4. Fetch second level PTE:
- PTE Pointer = PTE.PFN||INDEX2||00;
- PTE ← (PTE Pointer); /* Fetch PTE2 */
- **If** (PTE.PL < effective PL) **then**
— Effective PL ← PTE.PL;

5. Validate Second Level PTE:
- **If** (PTE.PL < access level) **then**
- /* Protection Exception */
— TEAR ← virtual address,
— clock MSR with MSR.TEX = 11,
— terminate translation;
- **If** (PTE.V = 0) **then**
- /* PTE2 Invalid */
— TEAR ← virtual address,
— clock MSR with MSR.TEX = 10,
— terminate translation;
- **If** ((read AND NOT interlocked) AND PTE.R = 0) **then** Read-Modify-Write a double-word interlocked (PTE Pointer).R = 1;
- **If** ((write OR interlocked read) AND (PTE.R = 0 OR PTE.M = 0) **then** Read-Modify-Write a double-word interlocked (PTE Pointer).R = 1, (PTE Pointer).M = 1;

6. Generate Physical address:
- physical address ← PTE.PFN||OFFSET
- CIOUT pin ← PTE.CI

7. Update Translation Buffer:
- Select entry for replacement;
- TLB. Virtual Page Number ← INDEX1|| INDEX2;
- TLB.AS ← AS;
- TLB. Physical Frame Number ← PTE.PFN
- TLB.PL ← Effective PL
- TLB.CI ← PTE.CI
- TLB.M ← (PTE Pointer) .M
- Enable entry

END

Note 1: The TEAR and MSR are only updated when a Trap (ABT) occurs. It is possible that the MMU detects a page fault or protection violation on a reference for an instruction that is not executed, for example on a prefetch. In that event, Trap (ABT) does not occur, and the TEAR and MSR are not updated.

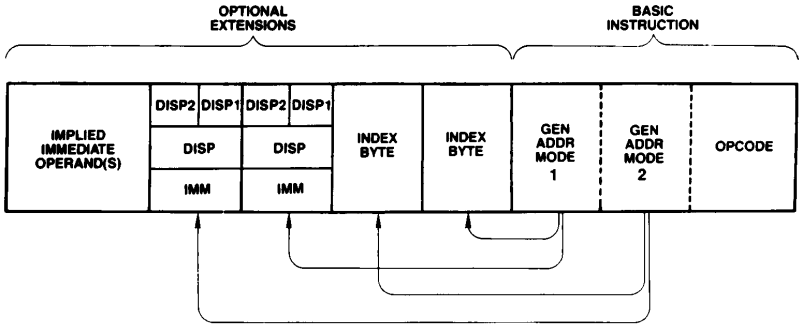
Note 2: If the MMU is translating a virtual address to check protection while executing a RDVAL or WRVAL instruction, then Trap (ABT) occurs only if the level-1 PTE is invalid and the access is permitted by the PL-field. These instructions will not generate an abort if the F bit value can be determined from Level-1 PTE.

2.5 INSTRUCTION SET

2.5.1 General Instruction Format

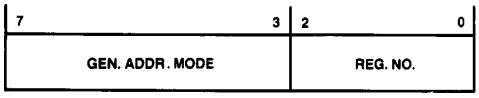
Figure 2-14 shows the general format of a Series 32000 instruction. The Basic Instruction is one to three bytes long and contains the Opcode and up to two 5-bit General Addressing Mode (''Gen'') fields. Following the Basic Instruction field is a set of optional extensions, which may appear depending on the instruction and the addressing modes selected.

Index 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 General Purpose Register to use as the index, and which addressing mode calculation to perform before indexing. See *Figure 2-15*.



TL/EE/9354–5

FIGURE 2-14. General Instruction Format



TL/EE/9354–6

FIGURE 2-15. Index Byte Format

Following Index Bytes come any displacements (addressing constants) or immediate values associated with the selected addressing modes. Each Disp/Imm field may contain one or two displacements, or one immediate value. The size of a Displacement field is encoded with the top bits of that field, as shown in *Figure 2-16*, 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 Displacement and Immediate fields are stored most significant byte first. Note that this is different from the memory representation of data (Section 2.2).

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 2.5.3).

2.5.2 Addressing Modes

The 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 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.

Addressing Modes fall into nine basic types:

Register: The operand is available in one of the eight General Purpose Registers. In certain Slave Processor instructions, an auxiliary set of eight registers may be referenced instead.

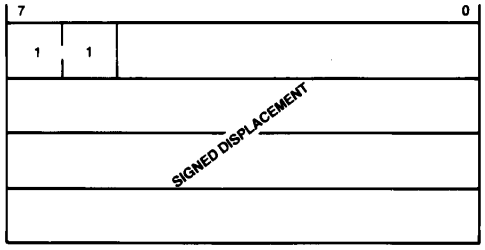
Register Relative: A General Purpose Register contains an address to which is added a displacement value from the instruction, yielding the Effective Address of the operand in memory.

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.

Byte Displacement: Range −64 to +63

Word Displacement: Range −8192 to +8191

Double Word Displacement:
Range $-(2^{29} - 2^{24})$ to $+ (2^{29} - 1)$*



TL/EE/9354–7

FIGURE 2-16. Displacement Encodings

***Note:** The pattern ''11100000'' for the most significant byte of the displacement is reserved by National for future enhancements. Therefore, it should never be used by the user program. This causes the lower limit of the displacement range to be $-(2^{29}-2^{24})$ instead of -2^{29}.

2.0 Architectural Description (Continued)

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.

Immediate: The operand is encoded within the instruction. This addressing mode is not allowed if the operand is to be written.

Absolute: The address of the operand is specified by a displacement field in the instruction.

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.

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.

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 General Purpose Register by 1, 2, 4 or 8 and adding it into the total, yielding the final Effective Address of the operand.

Table 2-2 is a brief summary of the addressing modes. For a complete description of their actions, see the Instruction Set Reference Manual.

2.5.3 Instruction Set Summary

Table 2-3 presents a brief description of the NS32532 instruction set. The Format column refers to the Instruction Format tables (Appendix A). 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 Instruction Set 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 A for encodings).

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 legal.

reg = Any General Purpose Register: R0–R7.

areg = Any Processor Register: Address, Debug, Status, Configuration.

mreg = Any Memory Management Register.

creg = A Custom Slave Processor Register (Implementation Dependent).

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

2.0 Architectural Description (Continued)

TABLE 2-2. NS32532 Addressing Modes

ENCODING	MODE	ASSEMBLER SYNTAX	EFFECTIVE ADDRESS
Register			
00000	Register 0	R0, F0, L0	None: Operand is in the specified register.
00001	Register 1	R1, F1, L1	
00010	Register 2	R2, F2, L2	
00011	Register 3	R3, F3, L3	
00100	Register 4	R4, F4, L4	
00101	Register 5	R5, F5, L5	
00110	Register 6	R6, F6, L6	
00111	Register 7	R7, F7, L7	
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))	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))	
Reserved			
10011	(Reserved for Future Use)		
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.
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.

2.0 Architectural Description (Continued)

TABLE 2-3. NS32532 Instruction 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.

SHIFTS

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

TABLE 2-3. NS32532 Instruction Set Summary (Continued)

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 General Purpose Registers:
R4 - Comparison Value
R3 - Translation Table Pointer
R2 - String 2 Pointer
R1 - String 1 Pointer
R0 - Limit Count

Options on all string 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.

Format	Operation	Operands	Description
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.

2.0 Architectural Description (Continued)

TABLE 2-3. NS32532 Instruction Set Summary (Continued)

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)

CPU REGISTER MANIPULATION

Format	Operation	Operands	Description
1	SAVE	[reg list]	Save General Purpose Registers.
1	RESTORE	[reg list]	Restore General Purpose Registers.
2	LPRi	areg,gen	Load Processor Register. (Privileged if PSR, INTBASE, USP, CFG or Debug Registers).
2	SPRi	areg,gen	Store Processor Register. (Privileged if PSR, INTBASE, USP, CFG or Debug Registers).
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	FLOORfi	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.
12	POLYf	gen,gen	Polynomial Step.
12	DOTf	gen,gen	Dot Product.
12	SCALBf	gen,gen	Binary Scale.
12	LOGBf	gen,gen	Binary Log.
12	SQRTf	gen,gen	Square Root
12	MACf	gen,gen	Multiply and Accumulate
9	LFSR	gen	Load FSR.
9	SFSR	gen	Store FSR.

TABLE 2-3. NS32532 Instruction Set Summary (Continued)

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 Supervisor Space to User Space. (Privileged)
8	MOVUSi	gen,gen	Move a value from User Space to Supervisor Space. (Privileged)

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.
14	CINV	[options],gen	Cache Invalidate. (Privileged)

CUSTOM SLAVE

Format	Operation	Operands	Description
15.5	CCAL0c	gen,gen	Custom Calculate.
15.5	CCAL1c	gen,gen	
15.5	CCAL2c	gen,gen	
15.5	CCAL3c	gen,gen	
15.5	CMOV0c	gen,gen	Custom Move.
15.5	CMOV1c	gen,gen	
15.5	CMOV2c	gen,gen	
15.5	CMOV3c	gen,gen	
15.5	CCMP0c	gen,gen	Custom Compare.
15.5	CCMP1c	gen,gen	
15.1	CCV0ci	gen,gen	Custom 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 Custom Status Register.
15.1	SCSR	gen	Store Custom Status Register.
15.0	LCR	creg,gen	Load Custom Register. (Privileged)
15.0	SCR	creg,gen	Store Custom Register. (Privileged)

3.0 Functional Description

This chapter provides details on the functional characteristics of the NS32532 microprocessor.

The chapter is divided into five main sections:

Instruction Execution, Exception Processing, Debugging, On-Chip Caches and System Interface.

3.1 INSTRUCTION EXECUTION

To execute an instruction, the NS32532 performs the following operations:

- Fetch the instruction
- Read source operands, if any (1)
- Calculate results
- Write result operands, if any
- Modify flags, if necessary
- Update the program counter

Under most circumstances, the CPU can be conceived to execute instructions by completing the operations above in strict sequence for one instruction and then beginning the sequence of operations for the next instruction. However, due to the internal instruction pipelining, as well as the occurrence of exceptions, the sequence of operations performed during the execution of an instruction may be altered. Furthermore, exceptions also break the sequentiality of the instructions executed by the CPU.

Details on the effects of the internal pipelining, as well as the occurrence of exceptions on the instruction execution, are provided in the following sections.

Note: 1 In this and following sections, memory locations read by the CPU to calculate effective addresses for Memory-Relative and External addressing modes are considered like source operands, even if the effective address is being calculated for an operand with access class of write.

3.1.1 Operating States

The CPU has five operating states regarding the execution of instructions and the processing of exceptions: Reset, Executing Instructions, Processing An Exception, Waiting-For-An-Interrupt, and Halted. The various states and transitions between them are shown in *Figure 3-1*.

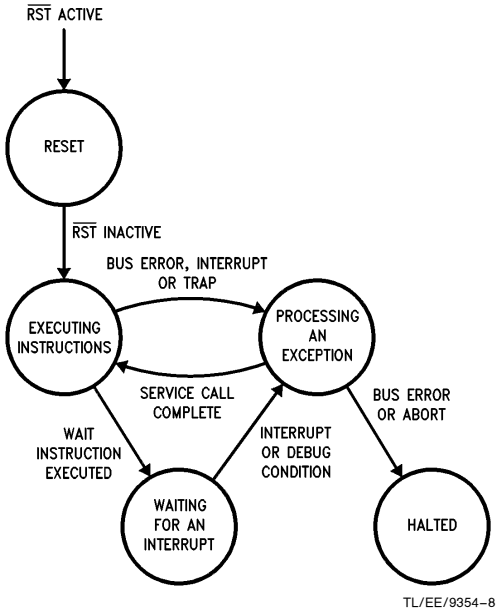
Whenever the $\overline{RST}$ signal is asserted, the CPU enters the reset state. The CPU remains in the reset state until the $\overline{RST}$ signal is driven inactive, at which time it enters the Executing-Instructions state. In the Reset state the contents of certain registers are initialized. Refer to Section 3.5.3 for details.

In the Executing-Instructions state, the CPU executes instructions. It will exit this state when an exception is recognized or a WAIT instruction is encountered. At which time it enters the Processing-An-Exception state or the Waiting-For-An-Interrupt state respectively.

While in the Processing-An-Exception state, the CPU saves the PC, PSR and MOD register contents on the stack and reads the new PC and module linkage information to begin execution of the exception service procedure (see note).

Following the completion of all data references required to process an exception, the CPU enters the Executing-Instructions state.

In the Waiting-For-An-Interrupt state, the CPU is idle. A special status identifying this state is presented on the system interface (Section 3.5). When an interrupt or a debug condition is detected, the CPU enters the Processing-An-Exception state.



TL/EE/9354–8

FIGURE 3-1. Operating States

The CPU enters the Halted state when a bus error or abort is detected while the CPU is processing an exception, thereby preventing the transfer of control to an appropriate exception service procedure. The CPU remains in the Halted state until reset occurs. A special status identifying this state is presented on the system interface.

Note: When the Direct-Exception mode is enabled, the CPU does not save the MOD Register contents nor does it read the module linkage information for the exception service procedure. Refer to Section 3.2 for details.

3.1.2 Instruction Endings

The NS32532 checks for exceptions at various points while executing instructions. Certain exceptions, like interrupts, are in most cases recognized between instructions. Other exceptions, like Divide-By-Zero Trap, are recognized during execution of an instruction. When an exception is recognized during execution of an instruction, the instruction ends in one of four possible ways: completed, suspended, terminated, or partially completed. Each type of exception causes a particular ending, as specified in Section 3.2.

3.1.2.1 Completed Instructions

When an exception is recognized after an instruction is completed, the CPU has performed all of the operations for that instruction and for all other instructions executed since the last exception occurred. Result operands have been written, flags have been modified, and the PC saved on the Interrupt Stack contains the address of the next instruction to execute. The exception service procedure can, at its conclusion, execute the RETT instruction (or the RETI instruction for vectored interrupts), and the CPU will begin executing the instruction following the completed instruction.

3.1.2.2 Suspended Instructions

An instruction is suspended when one of several trap conditions or a restartable bus error is detected during execution of the instruction. A suspended instruction has not been completed, but all other instructions executed since the last exception occurred have been completed. Result operands and flags due to be affected by the instruction may have been modified, but only modifications that allow the instruction to be executed again and completed can occur. For certain exceptions (Trap (ABT), Trap (UND), Trap (ILL), and bus errors) the CPU clears the P-flag in the PSR before saving the copy that is pushed on the Interrupt Stack. The PC saved on the Interrupt Stack contains the address of the suspended instruction.

For example, the RESTORE instruction pops up to 8 general-purpose registers from the stack. If an invalid page table entry is detected on one of the references to the stack, then the instruction is suspended. The general-purpose registers due to be loaded by the instruction may have been modified, but the stack pointer still holds the same value that it did when the instruction began.

To complete a suspended instruction, the exception service procedure takes either of two actions:

1. The service procedure can simulate the suspended instruction's execution. After calculating and writing the instruction's results, the flags in the PSR copy saved on the Interrupt Stack should be modified, and the PC saved on the Interrupt Stack should be updated to point to the next instruction to execute. The service procedure can then execute the RETT instruction, and the CPU begins executing the instruction following the suspended instruction. This is the action taken when floating-point instructions are simulated by software in systems without a hardware floating-point unit.
2. The suspended instruction can be executed again after the service procedure has eliminated the trap condition that caused the instruction to be suspended. The service procedure should execute the RETT instruction at its conclusion; then the CPU begins executing the suspended instruction again. This is the action taken by a debugger when it encounters a BPT instruction that was temporarily placed in another instruction's location in order to set a breakpoint.

Note 1: Although the NS32532 allows a suspended instruction to be executed again and completed, the CPU may have read a source operand for the instruction from a memory-mapped peripheral port before the exception was recognized. In such a case, the characteristics of the peripheral device may prevent correct reexecution of the instruction.

Note 2: It may be necessary for the exception service procedure to alter the P-flag in the PSR copy saved on the Interrupt Stack: If the exception service procedure simulates the suspended instruction and the P-flag was cleared by the CPU before saving the PSR copy, then the saved T-flag must be copied to the saved P-flag (like the floating-point instruction simulation described above). Or if the exception service procedure executes the suspended instruction again and the P-flag was not cleared by the CPU before saving the PSR copy, then the saved P-flag must be cleared (like the breakpoint trap described above). Otherwise, no alteration to the saved P-flag is necessary.

3.1.2.3 Terminated Instructions

An instruction being executed is terminated when reset or a nonrestartable bus error occurs. Any result operands and flags due to be affected by the instruction are undefined, as are the contents of the Stack Pointers. The result operands of other instructions executed since the last serializing operation may not have been written to memory. A terminated instruction cannot be completed.

3.1.2.4 Partially Completed Instructions

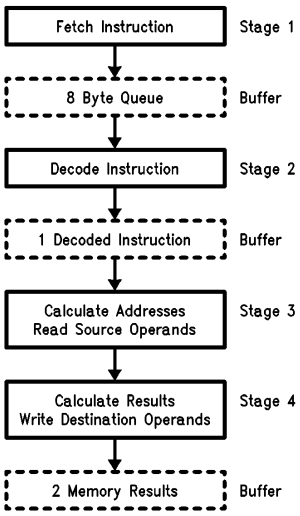
When a restartable bus error, interrupt, abort, or debug condition is recognized during execution of a string instruction, the instruction is said to be partially completed. A partially completed instruction has not completed, but all other instructions executed since the last exception occurred have been completed. Result operands and flags due to be affected by the instruction may have been modified, but the values stored in the string pointers and other general-purpose registers used during the instruction's execution allow the instruction to be executed again and completed.

The CPU clears the P-flag in the PSR before saving the copy that is pushed on the Interrupt Stack. The PC saved on the Interrupt Stack contains the address of the partially completed instruction. The exception service procedure can, at its conclusion, simply execute the RETT instruction (or the RETI instruction for vectored interrupts), and the CPU will resume executing the partially completed instruction.

3.1.3 Instruction Pipeline

The NS32532 executes instructions in a heavily pipelined fashion. This allows a significant performance enhancement since the operations of several instructions are performed simultaneously rather than in a strictly sequential manner.

The CPU provides a four-stage internal instruction pipeline. As shown in *Figure 3-2*, a write buffer, that can hold up to two operands, is also provided to allow write operations to be performed off-line.



TL/EE/9354–9

FIGURE 3-2. NS32532 Internal Instruction Pipeline

Due to the pipelining, operations like fetching one instruction, reading the source operands of a second instruction, calculating the results of a third instruction and storing the results of a fourth instruction, can all occur in parallel.

The order of memory references performed by the CPU may also differ from that related to a strictly sequential instruction execution. In fact, when an instruction is being executed, some of the source operands may be read from memory before the instruction is completely fetched. For example, the CPU may read the first source operand for an instruction before it has fetched a displacement used in calculating the address of the second source operand. The CPU, however, always completes fetching an instruction and reading its source operands before writing its results. When more than one source operand must be read from memory to execute an instruction, the operands may be read in any order. Similarly, when more than one result operand is written to memory to execute an instruction, the operands may be written in any order.

An instruction is fetched only after all previous instructions have been completely fetched. However, the CPU may begin fetching an instruction before all of the source operands have been read and results written for previous instructions.

The source operands for an instruction are read only after all previous instructions have been fetched and their source operands read. A source operand for an instruction may be read before all results of previous instructions have been written, except when the source operand's value depends on a result not yet written. The CPU compares the physical address and length of a source operand with those of any results not yet written, and delays reading the source operand until after writing all results on which the source operand depends. Also, the CPU ensures that the interlocked read and write references to execute an SBITIi or CBITIi instruction occur after writing all results of previous instructions and before reading any source operands for subsequent instructions.

The result operands for an instruction are written after all results of previous instructions have been written.

The description above is summarized in *Figure 3-3*, which shows the precedence of memory references for two consecutive instructions.

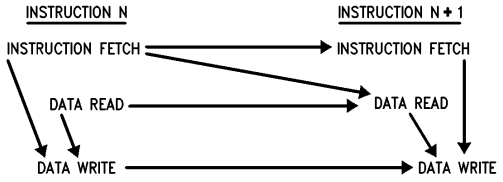


FIGURE 3-3. Memory References for Consecutive Instructions
(An arrow from one reference to another indicates that the first reference always precedes the second.)

Another consequence of overlapping the operations for several instructions, is that the CPU may fetch an instruction and read its source operands, even though the instruction is not executed (e.g., due to the occurrence of an exception). In such a case, the MMU may update the R-bit in Page Table Entries used in referring to the fetched instruction and its source operands.

Special care is needed in the handling of memory-mapped I/O devices. The CPU provides special mechanisms to ensure that the references to these devices are always performed in the order implied by the program. Refer to Section 3.1.3.2 for details.

It is also to be noted that the CPU does not check for dependencies between the fetching of an instruction and the writing of previous instructions' results. Therefore, special care is required when executing self-modifying code.

3.1.3.1 Branch Prediction

One problem inherent to all pipelined machines is what is called ''Pipeline Breakage''.

This occurs every time the sequentiality of the instructions is broken, due to the execution of certain instructions or the occurrence of exceptions.

The result of a pipeline breakage is a performance degradation, due to the fact that a certain portion of the pipeline must be flushed and new data must be brought in.

The NS32532 provides a special mechanism, called branch prediction, that helps minimize this performance penalty.

When a conditional branch instruction is decoded in the early stages of the pipeline, a prediction on the execution of the instruction is performed.

More precisely, the prediction mechanism predicts backward branches as taken and forward branches as not taken, except for the branch instructions BLE and BNE that are always predicted as taken.

Thus, the resulting probability of correct prediction is fairly high, especially for branch instructions placed at the end of loops.

The sequence of operations performed by the loader and execution units in the CPU is given below:

- Loader detects branches and calculates destination addresses
- Loader uses branch opcode and direction to select between sequential and non-sequential streams
- Loader saves address for alternate stream
- Execution unit resolves branch decision

Due to the branch predicition, some special care is required when writing self-modifying code. Refer to the appropriate section in Appendix B for more information on this subject.

3.1.3.2 Memory-Mapped I/O

The characteristics of certain peripheral devices and the overlapping of instruction execution in the pipeline of the NS32532 require that special handling be applied to memory-mapped I/O references. I/O references differ from memory references in two significant ways, imposing the following requirements:

1. Reading from a peripheral port can alter the value read on the next reference to the same port or another port in the same device. (A characteristic called here ''destructive-reading''.) Serial communication controllers and FIFO buffers commonly operate in this manner. As explained in ''Instruction Pipeline'' above, the NS32532 can read the source operands for one instruction while the previous instruction is executing. Because the previous instruction may cause a trap, an interrupt may be recognized, or the flow of control may be otherwise altered, it is a requirement that destructive-reading of source operands before the execution of an instruction be avoided.

2. Writing to a peripheral port can alter the value read from another port of the same device. (A characteristic called here ''side-effects of writing''). For example, before reading the counter's value from the NS32202 Interrupt Control Unit it is first necessary to freeze the value by writing to another control register.

However, as mentioned above, the NS32532 can read the source operands for one instruction before writing the results of previous instructions unless the addresses indicate a dependency between the read and write references. Consequently, it is a requirement that read and write references to peripheral that exhibit side-effects of writing must occur in the order dictated by the instructions.

The NS32532 supports 2 methods for handling memory-mapped I/O. The first method is more general; it satisfies both requirements listed above and places no restriction on the location of memory-mapped peripheral devices. The second method satisfies only the requirement for side effects of writing, and it restricts the location of memory-mapped I/O devices, but it is more efficient for devices that do not have destructive-read ports.

The first method for handling memory-mapped I/O uses two signals: $\overline{\text{IOINH}}$ and $\overline{\text{IODEC}}$. When the NS32532 generates a read bus cycle, it asserts the output signal $\overline{\text{IOINH}}$ if either of the I/O requirements listed above is not satisfied. That is, $\overline{\text{IOINH}}$ is asserted during a read bus cycle when (1) the read reference is for an instruction that may not be executed or (2) the read reference occurs while a write reference is pending for a previous instruction. When the read reference is to a peripheral device that implements ports with destructive-reading or side-effects of writing, the input signal $\overline{\text{IODEC}}$ must be asserted; in addition, the device must not be selected if $\overline{\text{IOINH}}$ is active. When the CPU detects that the $\overline{\text{IODEC}}$ input signal is active while the $\overline{\text{IOINH}}$ output signal is also active, it discards the data read during the bus cycle and serializes instruction execution. See the next section for details on serializing operations. The CPU then generates the read bus cycle again, this time satisfying the requirements for I/O and driving $\overline{\text{IOINH}}$ inactive.

The second method for handling memory-mapped I/O uses a dedicated region of virtual memory. The NS32532 treats all references to the memory range from address FF000000 to address FFFFFFFF inclusive in a special manner.

While a write to a location in this range is pending, reads from locations in the same range are delayed. However, reads from locations with addresses lower than FF000000 may occur. Similarly, reads from locations in the above range may occur while writes to locations outside of the range are pending.

It is to be noted that the CPU may assert $\overline{\text{IOINH}}$ even when the reference is within the dedicated region. Refer to Section 3.5.8 for more information on the handling of I/O devices.

3.1.3.3 Serializing Operations

After executing certain instructions or processing an exception, the CPU serializes instruction execution. Serializing instruction execution means that the CPU completes writing all previous instructions' results to memory, then begins fetching and executing the next instruction.

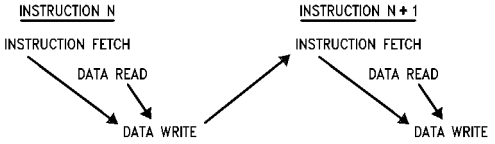
For example, when a new value is loaded into the PSR by executing an LPRW instruction, the pipeline is flushed and a serializing operation takes place. This is necessary since the privilege level might have changed and the instructions following the LPRW instruction must be fetched again with the new privilege level and possibly with a different MMU mapping. See Section 2.4.2.

The CPU serializes instruction execution after executing one of the following instructions: BICPSRW, BISPSRW, BPT, CINV, DIA, FLAG (trap taken), LMR, LPR (CFG, INTBASE, PSR, UPSR, DCR, BPC, DSR, and CAR only), RETT, RETI, and SVC. *Figure 3-4* shows the memory references after serialization.

Note 1: LPRB UPSR can be executed in User Mode to serialize instruction execution.

Note 2: After an instruction that writes a result to memory is executed, the updating of the result's memory location may be delayed until the next serializing operation.

Note 3: When reset or a nonrestartable bus error exception occurs, the CPU discards any results that have not yet been written to memory.



TL/EE/9354–11

FIGURE 3-4. Memory References after Serialization

3.1.4 Slave Processor Instructions

The NS32532 recognizes two groups of instructions being executable by external slave processors:

- Floating Point Instructions
- Custom Slave Instructions

Each Slave Instruction Set is enabled by a bit in the Configuration Register (Section 2.1.4). Any Slave Instruction which does not have its corresponding Configuration Register bit set will trap as undefined, without any Slave Processor communication attempted by the CPU. This allows software simulation of a non-existent Slave Processor.

Note that the Memory Management Instructions, like Floating Point and Custom Slave Instructions, have to be enabled through an appropriate bit in the configuration register in order to be executable.

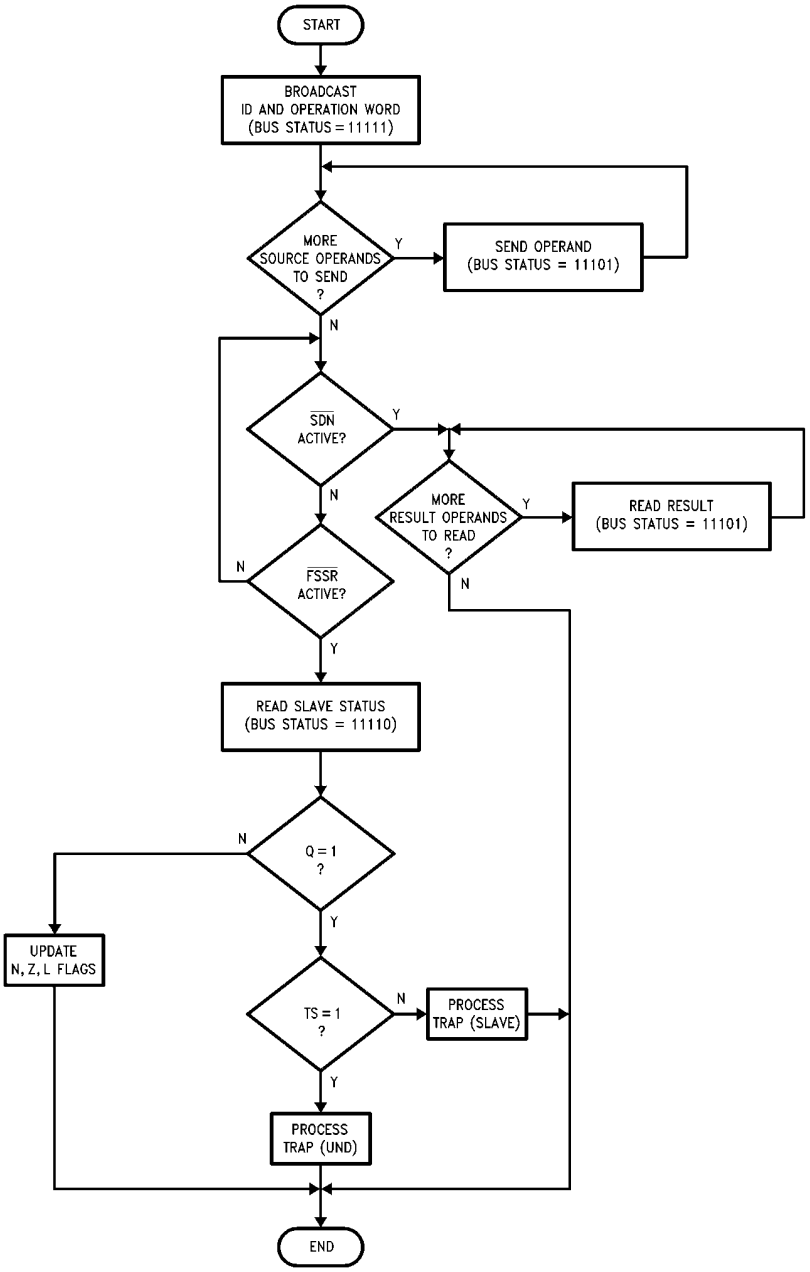
However, they are not considered here as Slave Instructions, since the NS32532 integrates the MMU on-chip and the execution of them does not follow the protocol of the Slave Instructions.

3.1.4.1 Regular Slave Instruction Protocol

Slave Processor instructions have a three-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 Slave Processor instruction.
2) It specifies which Slave Processor will execute it.
3) It determines the format of the following Operation Word of the instruction.

Upon receiving a Slave Processor instruction, the CPU initiates the sequence outlined in *Figure 3-5* . While applying Status code 11111 (Broadcast ID Section 3.5.4.1), the CPU transfers the ID Byte on bits D24–D31, the operation



TL/EE/9354–12

FIGURE 3-5. Regular Slave Instruction Protocol: CPU Actions

31			0
ID BYTE	OPCODE (LOW)	OPCODE (HIGH)	XXXXXXXX

FIGURE 3-6. ID and Operation Word

31	15		7							0
ZERO	TS	ZERO	N	Z	0	0	0	L	0	Q

FIGURE 3-7. Slave Processor Status Word

word on bits D8−D23 in a swapped order of bytes and a non-used byte XXXXXXXX (X = don't care) on bits D0−D7 (*Figure 3-6*).

All slave processors observe the bus cycle and inspect the identification code. The slave selected by the identification code continues with the protocol; other slaves wait for the next slave instruction to be broadcast.

After transferring the slave instruction, the CPU sends to the slave any source operands that are located in memory or the General-Purpose registers. The CPU then waits for the slave to assert $\overline{SDN}$ or $\overline{FSSR}$. While the CPU is waiting, it can perform bus cycles to fetch instructions and read source operands for instructions that follow the slave instruction being executed. If there are no bus cycles to perform, the CPU is idle with a special Status indicating that it is waiting for a slave processor. After the slave asserts $\overline{SDN}$ or $\overline{FSSR}$, the CPU follows one of the two sequences described below.

If the slave asserts $\overline{SDN}$, then the CPU checks whether the instruction stores any results to memory or the General-Purpose registers. The CPU reads any such results from the slave by means of 1 or 2 bus cycles and updates the destination.

If the slave asserts $\overline{FSSR}$, then the NS32532 reads a 32-bit status word from the slave. The CPU checks bit 0 in the slave's status word to determine whether to update the PSR flags or to process an exception. *Figure 3-7* shows the format of the slave's status word.

If the Q bit in the status word is 0, the CPU updates the N, Z and L flags in the PSR.

If the Q bit in the status word is set to 1, the CPU processes either a Trap (UND) if TS is 1 or a Trap (SLAVE) if TS is 0.

Note 1: Only the floating-point and custom compare instructions are allowed to return a value of 0 for the Q bit when the $\overline{FSSR}$ signal is activated. All other instructions must always set the Q bit to 1 (to signal a Trap), when activating $\overline{FSSR}$.

Note 2: While executing an LMR or CINV instruction, the CPU displays the operation code and source operand using slave processor write bus cycles, as described in the protocol above. Nevertheless, the CPU does not wait for $\overline{SDN}$ or $\overline{FSSR}$ to be asserted while executing these instructions. This information can be used to monitor the contents of the on-chip TLB, Instruction Cache, and Data Cache.

Note 3: The slave processor must be ready to accept new slave instruction at any time, even while the slave is executing another instruction or waiting for the CPU to read results. For example, the CPU may terminate an instruction being executed by a slave because a non-restartable bus error is detected while the MMU is updating a Page Table Entry for an instruction being prefetched.

Note 4: If a slave instruction stores a result to memory, the CPU checks whether Trap (ABT) would occur on the store operation before reading the result from the slave. For quad-word destination operands, the CPU checks that both double-words of the destination can be stored without an abort before reading either double-word of the result from the slave.

3.1.4.2 Pipelined Slave Instruction Protocol

In order to increase performance of floating-point instructions while maintaining full software compatibility with the Series 32000 architecture, the NS32532 incorporates a pipelined floating-point protocol. This protocol is designed to operate in conjunction with the NS32580 FPC, or any other floating-point slave which conforms to the protocol and the Series 32000 architecture. The protocol is enabled by the PF bit in the CFG register.

The basic methods of transferring data and control information between the CPU and the FPC, are the same as in the regular slave protocol.

However, in pipelined mode, the CPU may send a new floating-point instruction to the FPC before the previous instruction has been completed.

Although the CPU can advance as many as four floating-point instructions before receiving a completion pulse on $\overline{SDN}$ for the first instruction, full exception recovery is assured. This is accomplished through a FIFO mechanism which maintains the addresses of all the floating-point instructions sent to the FPC for execution.

Pipelined execution can occur only for instructions which do not require a result to be read from the FPC.

In cases where a result is to be read back, the CPU will wait for instruction completion before issuing the next instruction. Floating-point instructions can be divided into two groups, depending on the amount of pipelining permitted.

Group A. Fully-Pipelined Instructions

Instructions in this group can be sent to the FPC before previous group A instructions are completed. No instruction completion indication from the FPC is required in order to continue to another group A or group B instruction.

Group A contains floating-point instructions satisfying all of the following conditions.

1. The destination operand is in a floating-point register.
2. The source operand is not of type TOS or IMM.
3. The instruction format is either 11 or 12.

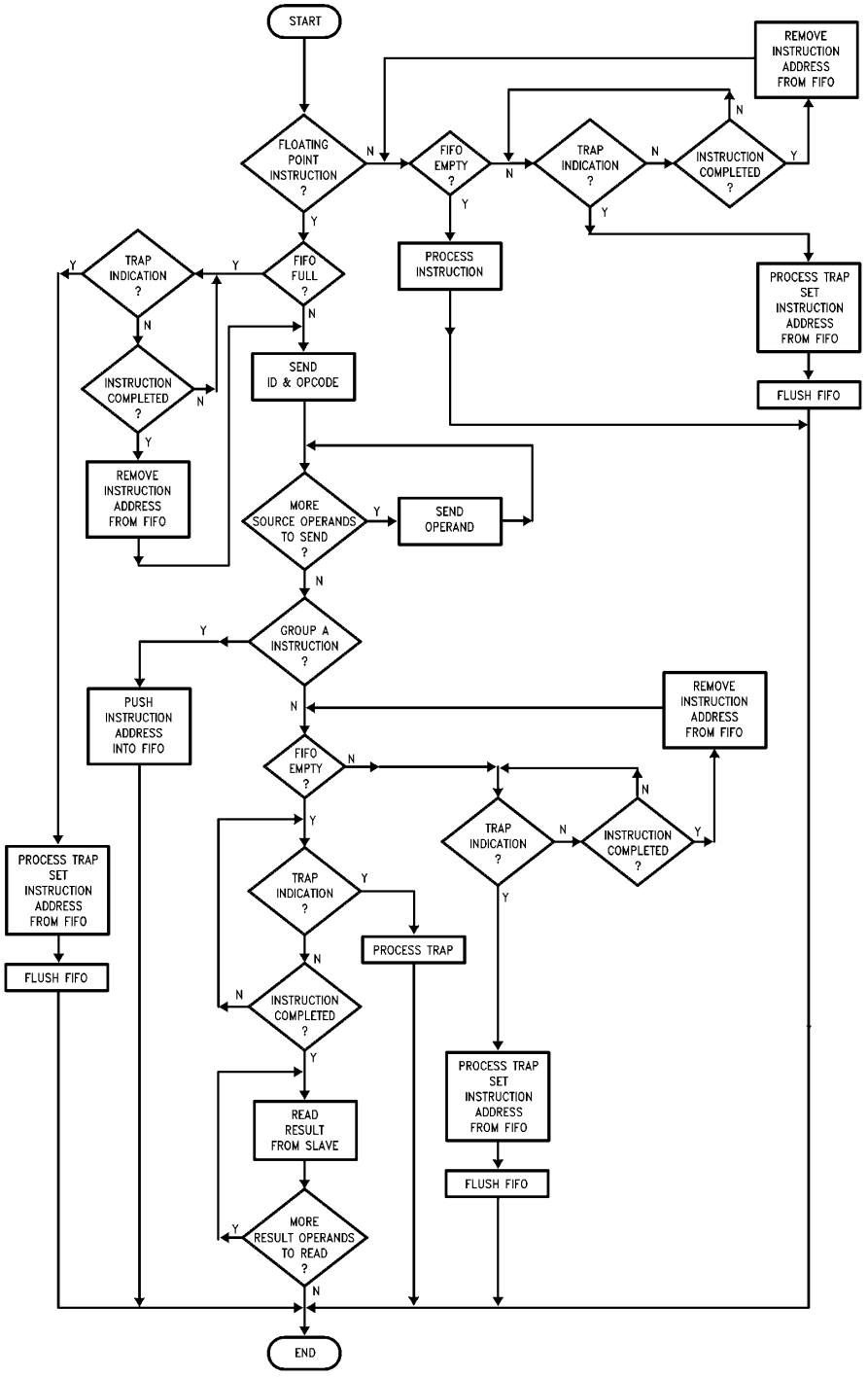
Group B. Half-Pipelined Instructions

Group B instructions can begin execution before previous group A instructions are completed. However, they cannot complete before the FPC signals completion of all the previous floating-point instructions.

Group B contains floating-point instructions satisfying at least one of the following conditions.

1. The destination operand is either in memory or in a CPU register (this includes the CMPf instruction which modifies the PSR register).
2. The source operand is of type TOS or IMM.
3. The instruction format is 9.

3.0 Functional Description (Continued)



TL/EE/9354–73

FIGURE 3-8. Instruction Flow in Pipelined Floating-Point Mode

Note: Non-floating-point instructions cannot be pipelined. They can begin execution only after all other instructions have been completed. The CPU cannot proceed to other instructions before their execution is completed.

3.1.4.3 Instruction Flow and Exceptions

When operating in pipelined mode, the CPU will push the address of group A instructions into a five-entry FIFO after the ID, opcode and source operands have been sent to the FPC. The address will be pushed into the FIFO only if no exception is detected during the transfer of the source operands needed for the execution of the instruction.

Group A instructions are only stalled when the FIFO is full, in which case the CPU will wait before sending the next instruction. Group B instructions can begin execution while some entries are still in the FIFO, but cannot complete before the FIFO is empty (i.e., before all previous instructions are completed). Non-floating-point instructions cannot begin execution until the FIFO is empty. When a normal completion indication is received, the instruction address at the bottom of the FIFO is dropped. If a trap indication is received and the FIFO is not empty, the instruction address at the bottom of the FIFO is copied to the PC register and the floating-point exception is serviced. The remaining entries in the FIFO are discarded.

A floating-point exception may be received and serviced at any time after the CPU has sent the ID and opcode for the first instruction and until the FPC has signalled completion for the last instruction.

Other exceptions may occur while the FIFO is not empty. This may be the case when an interrupt is received or a translation exception is detected in the access of an operand needed for the execution of the next floating-point instruction. These exceptions will be processed as soon as the FIFO becomes empty, and after any floating-point exception has been acknowledged.

In the event of a non-restartable bus error, the acknowledge will occur immediately. The CPU will flush the internal FIFO and will reset the FPC by performing a dummy read of the slave status word. This operation is performed for both the regular and pipelined floating-point protocol and regardless of whether any floating-point instruction is pending in the FPC instruction queue.

The CPU may cancel the last instruction sent to the FPC by sending another ID and opcode, before the last source operand for that instruction has been sent. *Figure 3-8* shows the instruction flow in pipelined floating-point mode.

3.1.4.4 Floating Point Instructions

Table 3-1 gives the protocols followed for each Floating Point instruction. The instructions are referenced by their mnemonics. For the bit encodings of each instruction, see Appendix A.

The Operand class columns give the Access Class for each general operand, defining how the addressing modes are interpreted (see Instruction Set 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 Slave Processor Status Word *(Figure 3-7)*.

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.

3.1.4.5 Custom Slave Instructions

Provided in the NS32532 is the capability of communicating with a user-defined, ''Custom'' Slave Processor. The instruction set provided for a Custom Slave Processor 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 Custom Slave and the actual types of data transferred. The protocol specifies only the size of an operand, not its data type.

Table 3-2 lists the relevant information for the Custom Slave 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 slave processor is already holding the operand internally.

For the instruction encodings, see Appendix A.

3.2 EXCEPTION PROCESSING

Exceptions are special events that alter the sequence of instruction execution. The CPU recognizes three basic types of exceptions: interrupts, traps and bus errors.

An interrupt occurs in response to an event signalled by activating the $\overline{\text{NMI}}$ or $\overline{\text{INT}}$ input signals. Interrupts are typically requested by peripheral devices that require the CPU's attention.

Traps occur as a result either of exceptional conditions (e.g., attempted division by zero) or of specific instructions whose purpose is to cause a trap to occur (e.g., supervisor call instruction).

A bus error exception occurs when the $\overline{\text{BER}}$ signal is activated during an instruction fetch or data transfer required by the CPU to execute an instruction.

When an exception is recognized, the CPU saves the PC, PSR and optionally the MOD register contents on the interrupt stack and then it transfers control to an exception service procedure.

Details on the operations performed in the various cases by the CPU to enter and exit the exception service procedure are given in the following sections.

It is to be noted that the reset operation is not treated here as an exception. Even though, like any exception, it alters the instruction execution sequence.

The reason being that the CPU handles reset in a significantly different way than it does for exceptions.

Refer to Section 3.5.3 for details on the reset operation.

3.0 Functional Description (Continued)

TABLE 3-1. 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
POLYf	read.f	read.f	f	f	f to F0	none
DOTf	read.f	read.f	f	f	f to F0	none
SCALBf	read.f	rmw.f	f	f	f to Op.2	none
LOGBf	read.f	write.f	f	N/A	f to Op.2	none
SQRTf	read.f	write.f	f	N/A	f to Op.2	none
MACf	read.f	read.f	f	f	f to F1	none

TABLE 3-2. Custom Slave 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
CMOV3c	read.c	write.c	c	N/A	c to Op.2	none
CCMP0c	read.c	read.c	c	c	N/A	N,Z,L
CCMP1c	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
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
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 = Custom 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.

3.0 Functional Description (Continued)

3.2.1 Exception Acknowledge Sequence

When an exception is recognized, the CPU goes through three major steps:

1) Adjustment of Registers. Depending on the source of the exception, 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 Supervisor Mode and selection of the Interrupt Stack. Trap (TRC) and Trap (OVF) are always disabled. Maskable interrupts are also disabled if the exception is caused by an interrupt, Trap (DBG), Trap (ABT) or bus error.
2) Vector Acquisition. A vector is either obtained from the data bus or is supplied internally by default.
3) Service Call. The CPU performs one of two sequences common to all exceptions to complete the acknowledge process and enter the appropriate service procedure. The selection between the two sequences depends on whether the Direct-Exception mode is disabled or enabled.

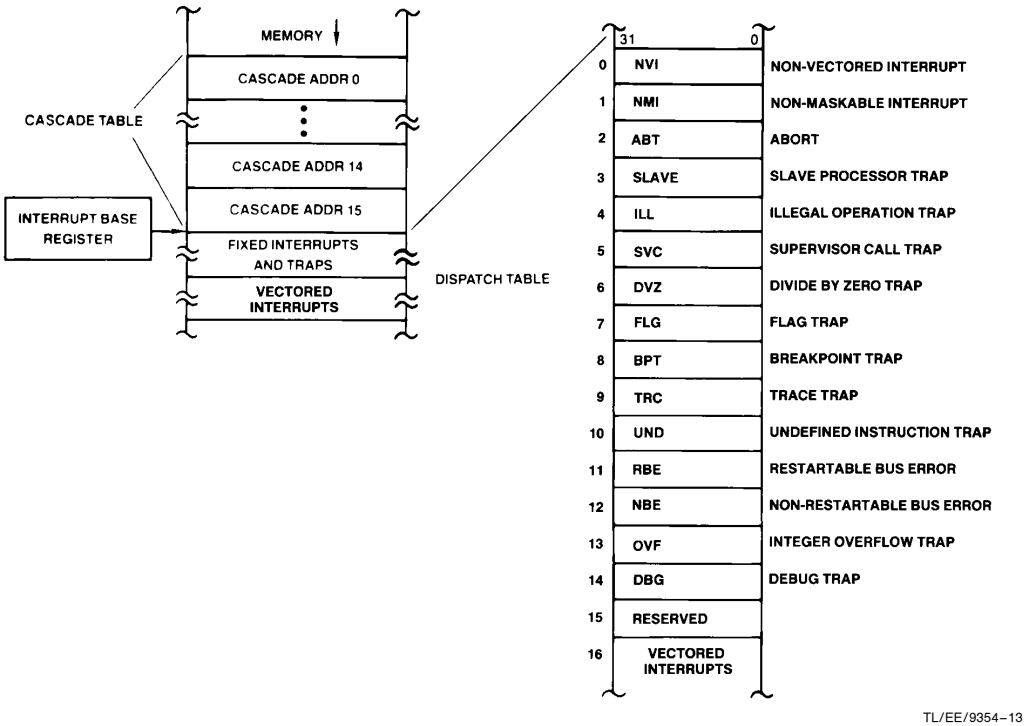
Direct-Exception Mode Disabled

The Direct-Exception mode is disabled while the DE bit in the CFG register is 0 (Section 2.1.4). In this case the CPU first pushes the saved PSR copy along with the contents of the MOD and PC registers on the interrupt stack. Then it reads the double-word entry from the Interrupt Dispatch table at address 'INTBASE + vector × 4'. See *Figures 3-9* and *3-10*. The CPU uses this entry to call the exception service procedure, interpreting the entry as an external procedure descriptor.

A new module number is loaded into the MOD register from the least-significant word of the descriptor, and the static-base pointer for the new module is read from memory and loaded into the SB register. Then the program-base pointer for the new module is read from memory and added to the most-significant word of the module descriptor, which is interpreted as an unsigned value. Finally, the result is loaded into the PC register.

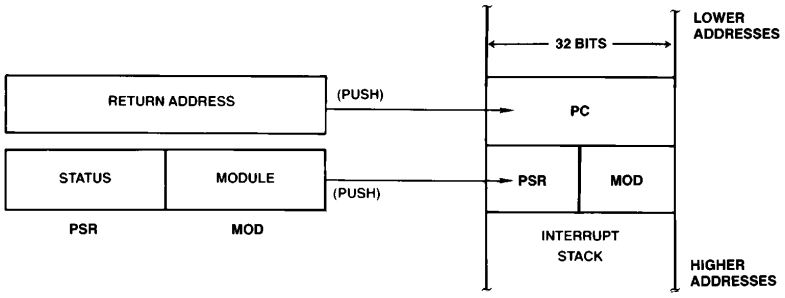
Direct-Exception Mode Enabled

The Direct-Exception mode is enabled when the DE bit in the CFG register is set to 1. In this case the CPU first pushes the saved PSR copy along with the contents of the PC register on the Interrupt Stack. The word stored on the Interrupt Stack between the saved PSR and PC register is reserved for future use; its contents are undefined. The CPU then reads the double-word entry from the Interrupt Dispatch Table at address 'INTBASE + vector × 4'. The CPU uses this entry to call the exception service procedure, interpreting the entry as an absolute address that is simply loaded into the PC register. *Figure 3-11* provides a pictorial of the acknowledge sequence. It is to be noted that while the

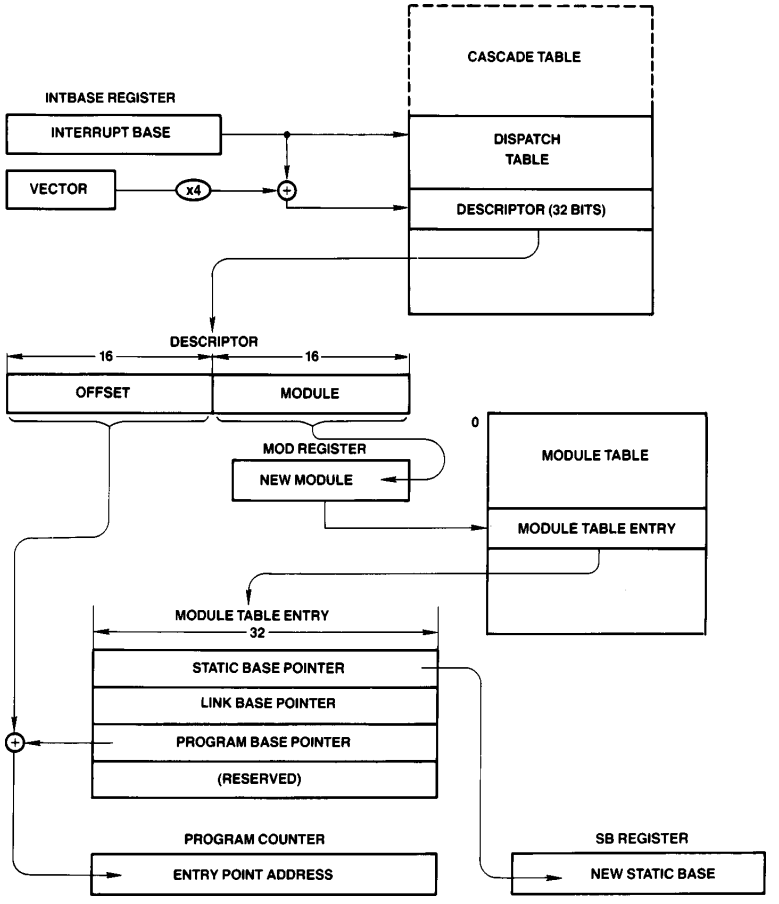


TL/EE/9354–13

FIGURE 3-9. Interrupt Dispatch Table

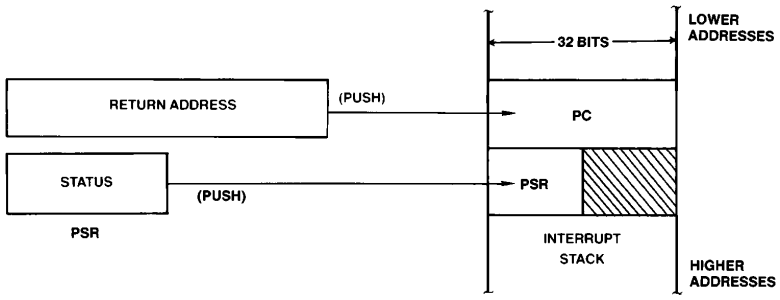


TL/EE/9354–14

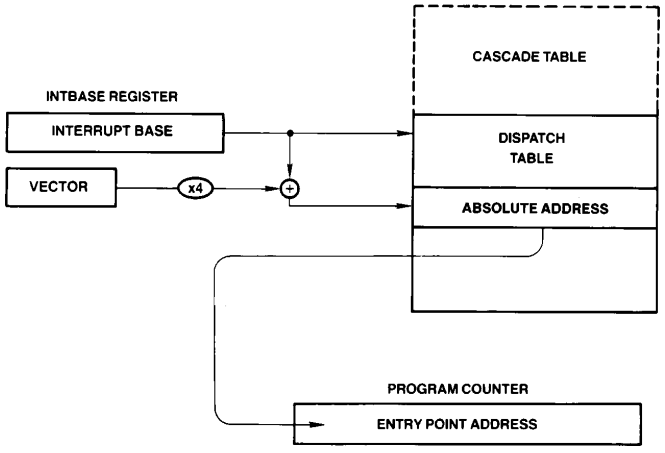


TL/EE/9354–15

FIGURE 3-10. Exception Acknowledge Sequence. Direct-Exception Mode Disabled.



TL/EE/9354–16



TL/EE/9354–17

FIGURE 3-11. Exception Acknowledge Sequence. Direct-Exception Mode Enabled.

direct-exception mode is enabled, the CPU can respond more quickly to interrupts and other exceptions because fewer memory references are required to process an exception. The MOD and SB registers, however, are not initialized before the CPU transfers control to the service procedure. Consequently, the service procedure is restricted from executing any instructions, such as CXP, that use the contents of the MOD or SB registers in effective address calculations.

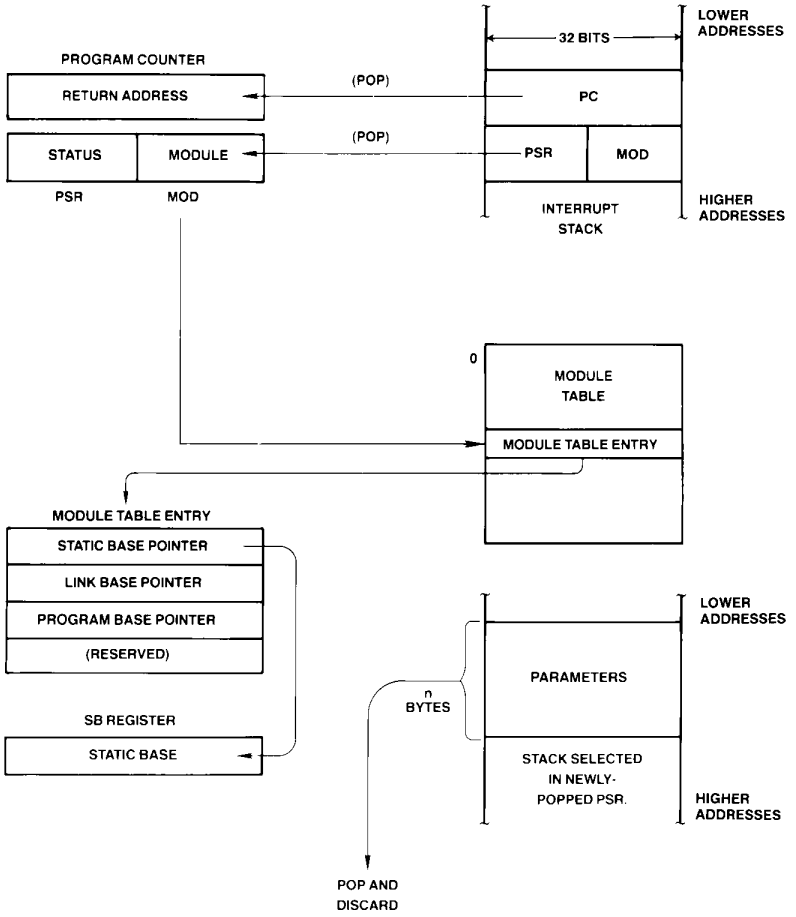
3.2.2 Returning from an Exception Service Procedure

To return control to an interrupted program, one of two instructions can be used: RETT (Return from Trap) and RETI (Return from Interrupt).

RETT is used to return from any trap, non-maskable interrupt or bus error service procedure. Since some traps are often used deliberately as a call mechanism for supervisor mode procedures, RETT can also adjust the Stack Pointer (SP) to discard a specified number of bytes from the original stack as surplus parameter space.

RETI is used to return from a maskable interrupt service procedure. A difference of RETT, RETI also informs any external interrupt control units that interrupt service has completed. Since interrupts are generally asynchronous external events, RETI does not discard parameters from the stack.

Both of the above instructions always restore the Program Counter (PC) and the Processor Status Register from the interrupt stack. If the Direct-Exception mode is disabled, they also restore the MOD and SB register contents. *Figures 3-12* and *3-13* show the RETT and RETI instruction flows when the Direct-Exception mode is disabled.



TL/EE/9354–18

FIGURE 3-12. Return from Trap (RETT n) Instruction Flow. Direct-Exception Mode Disabled.

3.2.3 Maskable Interrupts

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 $\overline{INT}$, $\overline{NMI}$, Trap (DBG), Trap (ABT) or Bus Error 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 Non-Vectored (CFG Register bit I = 0) or Vectored (bit I = 1).

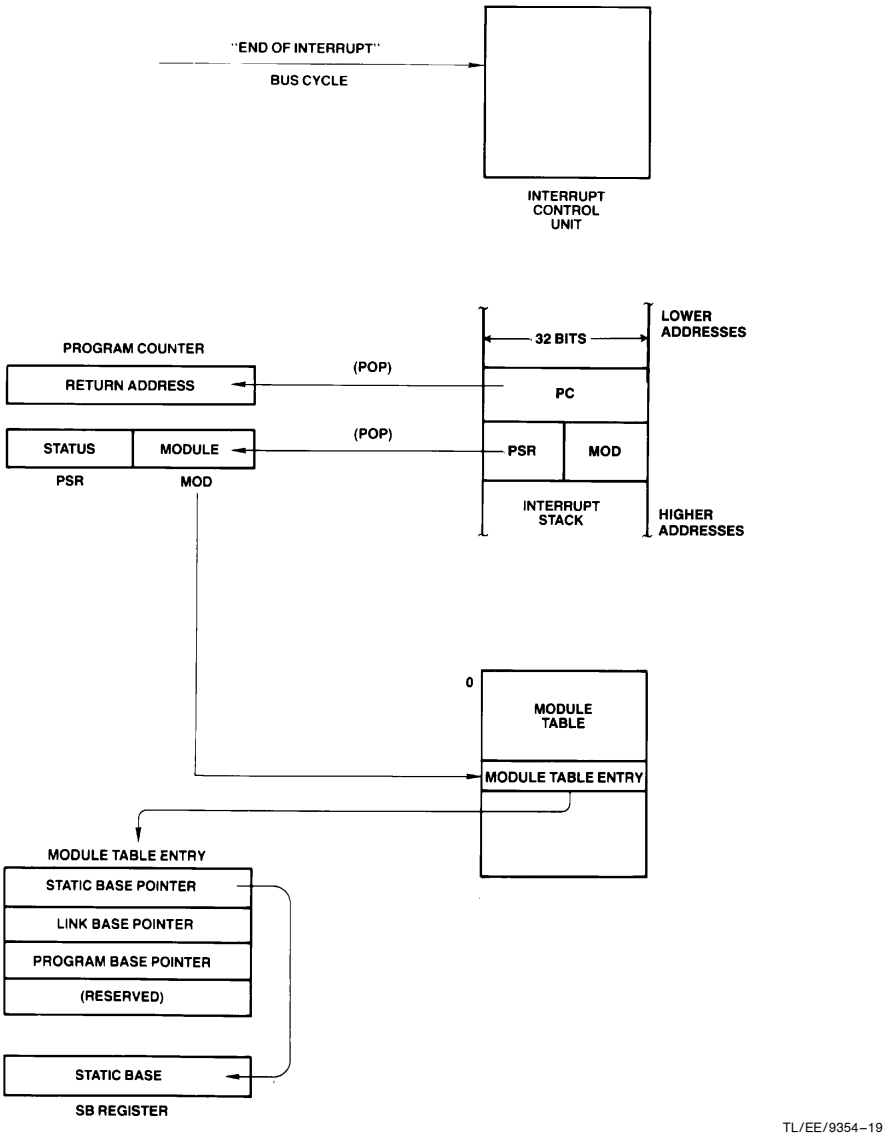
3.2.3.1 Non-Vectored Mode

In the Non-Vectored 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.

3.2.3.2 Vectored Mode: Non-Cascaded Case

In the Vectored mode, the CPU uses an Interrupt Control Unit (ICU) to prioritize many interrupt requests. Upon receipt of an interrupt request on the $\overline{INT}$ pin, the CPU performs an ''Interrupt Acknowledge, Master'' bus cycle (Section 3.5.4.6) 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 re-prioritize any interrupt requests still pending. The ICU provides the vector number again, which the CPU uses to determine whether it needs also 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



TL/EE/9354–19

FIGURE 3-13. Return from Interrupt (RETI) Instruction Flow. Direct-Exception Mode Disabled.

a negative vector number, an ICU flags the interrupt source as being a Cascaded ICU (see below).

Note: During a return from interrupt the CPU looks at bit 7 of the vector number from the master ICU. If bit 7 is 0, bits 0 through 6 are ignored.

3.2.3.3 Vectored Mode: Cascaded Case

In order to allow more levels of interrupt, provision is made in the CPU to transparently support cascading. Note that the Interrupt output from a Cascaded ICU goes to an Interrupt Request input of the Master ICU, which is the only ICU which drives the CPU $\overline{INT}$ pin. Refer to the ICU data sheet for details.

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 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 3-9 illustrates the position of the Cascade Table. To find the Cascade Table entry for a Cascaded ICU, take its Master ICU line number (0 to 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 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, 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, 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.

Note: If an interrupt must be masked off, the CPU can do so by setting the corresponding bit in the interrupt mask register of the interrupt controller.

However, if an interrupt is set pending during the CPU instruction that masks off that interrupt, the CPU may still perform an interrupt acknowledge cycle following that instruction since it might have sampled the $\overline{INT}$ line before the ICU deasserted it. This could cause the ICU to provide an invalid vector. To avoid this problem the above operation should be performed with the CPU interrupt disabled.

3.2.4 Non-Maskable Interrupt

The Non-Maskable Interrupt is triggered whenever a falling edge is detected on the $\overline{NMI}$ pin. The CPU performs an ''Interrupt Acknowledge, Master'' bus cycle (Section 3.5.4.6) when processing of this interrupt actually begins. The Interrupt Acknowledge cycle differs from that provided for Maskable Interrupts in that the address presented is $FFFFFF00_{16}$. The vector value used for the Non-Maskable Interrupt is taken as 1, regardless of the value read from the bus.

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

3.2.5 Traps

Traps are processing exceptions that are generated as direct results of the execution of an instruction.

The return address saved on the stack by any trap except Trap (TRC) and Trap (DBG) is the address of the first bye of the instruction during which the trap occurred.

When a trap is recognized, maskable interrupts are not disabled except for the case of Trap (ABT) and Trap (DBG).

There are 11 trap conditions recognized by the NS32532 as described below.

Trap (ABT): An abort trap occurs when an invalid page table entry or a protection level violation is detected for any of the memory references required to execute an instruction.

Trap (SLAVE): An exceptional condition was detected by the Floating Point Unit or another Slave Processor during the execution of a Slave Instruction. This trap is requested via the Status Word returned as part of the Slave Processor Protocol (Section 3.1.4.1).

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 PSR F bit.

Trap (BPT): The Breakpoint (BPT) instruction was executed.

Trap (TRC): The instruction just completed is being traced.

Refer to Section 3.3.1 for details.

Trap (UND): An Undefined-Instruction trap occurs when an attempt to execute an instruction is made and one or more of the following conditions is detected:

1. The instruction is undefined. Refer to Appendix A for a description of the codes that the CPU recognizes to be undefined.
2. The instruction is a floating point instruction and the F-bit in the CFG register is 0.
3. The instruction is a custom slave instruction and the C-bit in the CFG register is 0.
4. The instruction is a memory-management instruction and the M-bit in the CFG register is 0.
5. An LMR or SMR instruction is executed while the U-flag in the PSR is 0 and the most significant bit of the instruction's short field is 0.
6. The reserved general adressing mode encoding (10011) is used.
7. Immediate addressing mode is used for an operand that has access class different from read.

8. Scaled Indexing is used and the basemode is also Scaled Indexing.
9. The instruction is a floating-point or custom slave instruction that the FPU or custom slave detects to be undefined. Refer to Section 3.1.4.1 for more information.

Trap (OVF): An Integer-Overflow trap occurs when the V-bit in the PSR register is set to 1 and an Integer-Overflow condition is detected during the execution of an instruction. An Integer-Overflow condition is detected in the following cases:

1. The F-flag is 1 following execution of an ADDi, ADDQi, ADDCi, SUBi, SUBCi, NEGi, ABSi, or CHECKi instruction.
2. The product resulting from a MULi instruction cannot be represented exactly in the destination operand's location.
3. The quotient resulting from a DEIi, DIVi, or QUOi instruction cannot be represented exactly in the destination operand's location.
4. The result of an ASHi instruction cannot be represented exactly in the destination operand's location.
5. The sum of the 'INC' value and the 'INDEX' operand for an ACBi instruction cannot be represented exactly in the index operand's location.

Trap (DBG): A debug trap occurs when one or more of the conditions selected by the settings of the bits in the DCR register is detected. This trap can also be requested by activating the input signal $\overline{DBG}$. Refer to Section 3.3.2 for more information.

Note 1: Following execution of the WAIT instruction, then a Trap (DBG) can be pending for a PC-match condition. In such an event, the Trap (DBG) is processed immediately.

Note 2: If an attempt is made to execute a memory-management instruction while in User-Mode and the M-bit in the CFG register is 0, then Trap (UND) occurs.

Note 3: If an attempt is made to execute a privileged custom instruction while in User-Mode and the C-bit in the CFG register is 0, then Trap (UND) occurs.

Note 4: While operating in User-Mode, if an attempt is made to execute a privileged instruction with an undefined use of a general addressing mode (either the reserved encoding is used or else scaled-index or immediate modes are incorrectly used), the Trap (UND) occurs.

Note 5: If an undefined instruction or illegal operation is detected, then no data references are performed for the instruction.

Note 6: For certain instructions that are relatively long to execute, such as DEID, the CPU checks for pending interrupts during execution of the instruction. In order to reduce interrupt latency, the NS32532 can suspend executing the instruction and process the interrupt. Refer to Section B.5 in Appendix B for more information about recognizing interrupts in this manner.

3.2.6 Bus Errors

A bus error exception occurs when the $\overline{BER}$ signal is asserted in response to an instruction fetch or data transfer that is required to execute an instruction.

Two types of bus errors are recognized: Restartable and Non-Restartable. Restartable bus errors are recognized during read bus cycles, except for MMU read cycles (from Page Tables) needed to translate the address of a result being stored into memory. All other bus errors are non-restartable.

The CPU responds to restartable bus errors by suspending the instruction that it was executing. When a non-restartable bus error is detected, the CPU responds immediately and the instruction being executed is terminated. See Section 3.1.2.3.

The PC value saved on the stack is undefined.

The NS32532 does not respond to bus errors indicated for instructions that are not executed. For example, no bus error exception occurs in response to asserting the $\overline{BER}$ signal during a bus cycle to prefetch an instruction that is not executed because the previous instruction caused a trap.

An exception to this rule occurs if the bus error is detected during an MMU write cycle to update the R-bit in a page table entry.

In this case the CPU recognizes the bus error and considers it as non-restartable even though the bus cycle that caused it belongs to a non-executed instruction.

If a bus error is detected during a data transfer required for the processing of another exception or during the ICU read cycle of a RETI instruction, then the CPU considers it as a fatal bus error and enters the 'HALTED' state.

Note 1: If the address and control signals associated with the last bus cycle that caused a bus error are latched by external hardware, then the information they provide can be used by the service procedure for restartable bus errors to analyze and resolve the exception recognized by the CPU. This can be accomplished because upon detecting a restartable bus error, the NS32532 stops making memory references for subsequent instructions until it determines whether the instruction that caused the bus error is executed and the exception is processed.

Note 2: When a non-restartable bus error is recognized, the service procedure must execute the CINV and LMR instructions to invalidate the on-chip caches and TLB. This is necessary to maintain coherence between them and external memory.

Note 3: If the instruction causing a non-restartable bus error is followed by a slave instruction, the service procedure should reset the slave by reading the slave status register.

3.2.7 Priority Among Exceptions

The CPU checks for specific exceptions at various points while executing an instruction. It is possible that several exceptions occur simultaneously. In that event, the CPU responds to the exception with highest priority.

Figure 3-14 shows an exception processing flowchart. A non-restartable bus error is assigned highest priority and is serviced immediately regardless of the execution state of the CPU.

Before executing an instruction, the CPU checks for pending Trap (DBG), interrupts, and Trap (TRC), in that order. If a Trap (DBG) is pending, then the CPU processes that exception, otherwise the CPU checks for pending interrupts. At this point, the CPU responds to any pending interrupt requests; nonmaskable interrupts are recongized with higher priority than maskable interrupts. If no interrupts are pending, then the CPU checks the P-flag in the PSR to determine whether a Trap (TRC) is pending. If the P-flag is 1, a Trap (TRC) is processed. If no Trap (DBG), interrupt or Trap (TRC) is pending, the CPU begins executing the instruction.

While executing an instruction, the CPU may recognize up to four exceptions:

1. trap (ABT)
2. restartable bus error
3. trap (DBG) or interrupt, if the instruction is interruptible
4. one of 7 mutually exclusive traps: SLAVE, ILL, SVC, DVZ, FLG, BPT, UND

Trap (ABT) and restartable bus error have equal priority; the CPU responds to the first one detected.

If no exception is detected while the instruction is executing, then the instruction is completed and the PC is updated to point to the next instruction. If a Trap (OVF) is detected, then it is processed at this time.

3.0 Functional Description (Continued)

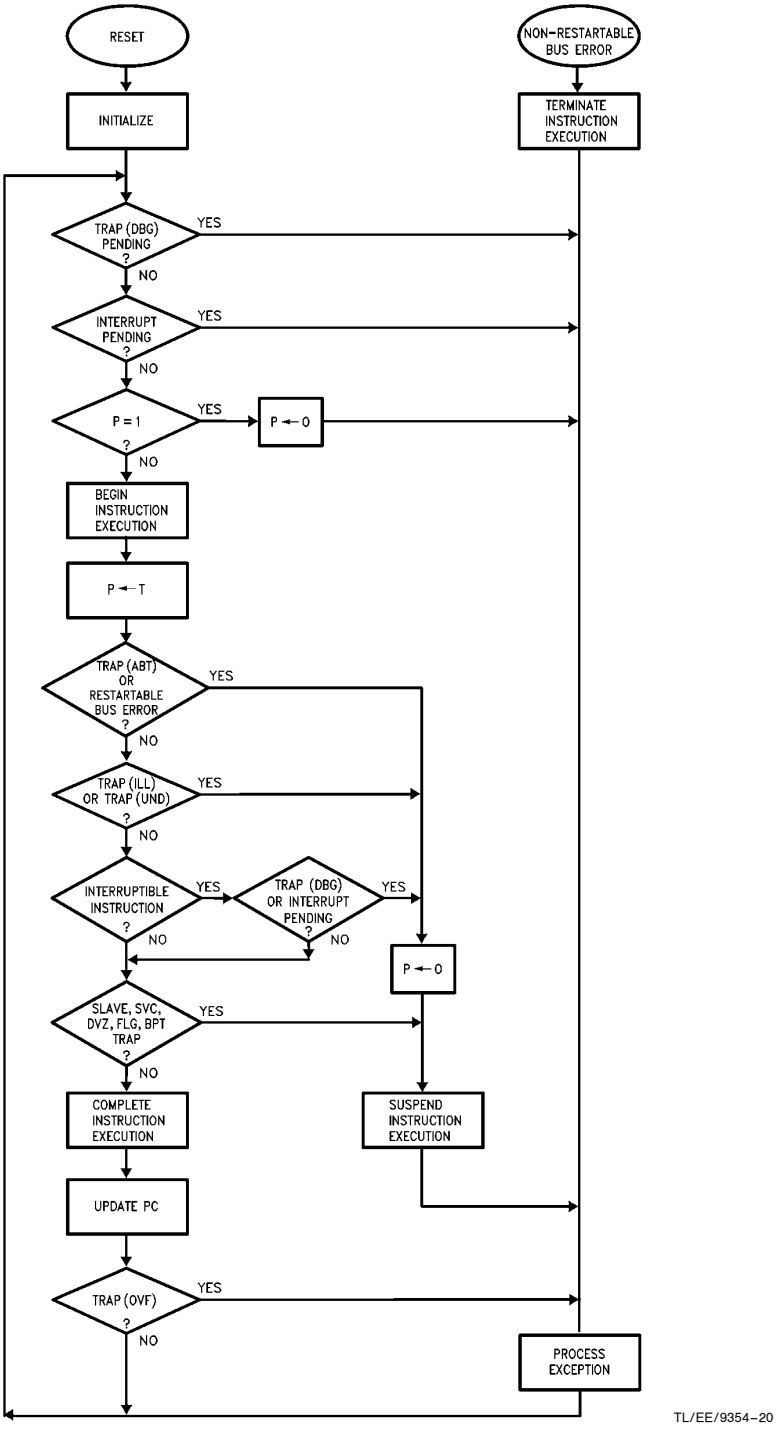


FIGURE 3-14. Exception Processing Flowchart

3.0 Functional Description (Continued)

While executing the instruction, the CPU checks for enabled debug conditions. If an enabled debug condition is met, a Trap (DBG) is held pending until after the instruction is completed (see Note 3). If another exception is detected before the instruction is completed, the pending Trap (DBG) is removed and the DSR register is not updated.

Note 1: Trap (DBG) can be detected simultaneously with Trap (OVF). In this event, the Trap (OVF) is processed before the Trap (DBG).

Note 2: An address-compare debug condition can be detected while processing a bus error, interrupt, or trap. In this event, the Trap (DBG) is held pending until after the CPU has processed the first exception.

Note 3: Between operations of a string instruction, the CPU responds to pending operand address compare and external debug conditions as well as interrupts. If a PC-match debug condition is detected while executing a string instruction, then Trap (DBG) is held pending until the instruction has completed.

3.2.8 Exception Acknowledge Sequences: Detailed Flow

For purposes of the following detailed discussion of exception acknowledge sequences, a single sequence called ''service'' is defined in *Figure 3-15*.

Upon detecting any interrupt request, trap or bus error condition, the CPU first performs a sequence dependent upon the type of exception. This sequence will include saving a copy of the Processor Status Register and establishing a vector and a return address. The CPU then performs the service sequence.

3.2.8.1 Maskable/Non-Maskable 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 an interruptible instruction (e.g., string instruction), at the next interruptible point during its execution.

1. If an interruptible 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 T, V, U, S, P and I.
3. If the interrupt is Non-Maskable:
   a. Read a byte from address $FFFFFF00_{16}$, applying Status Code 00100 (Interrupt Acknowledge, Master). Discard the byte read.
   b. Set ''Vector'' to 1.
   c. Go to Step 8.
4. If the interrupt is Non-Vectored:
   a. Read a byte from address $FFFFFE00_{16}$, applying Status Code 00100 (Interrupt Acknowledge, Master). Discard the byte read.
   b. Set ''Vector'' to 0.
   c. Go to Step 8.
5. Here the interrupt is Vectored. Read ''Byte'' from address $FFFFFE00_{16}$, applying Status Code 00100 (Interrupt Acknowledge, Master).
6. If ''Byte'' ≥ 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 00101 (Interrupt Acknowledge, Cascaded).
8. Perform Service (Vector, Return Address), *Figure 3-15*.

3.2.8.2 Abort/Restartable Bus Error Sequence

1. Suspend instruction and restore the currently selected Stack Pointer to its original contents at the beginning of the instruction.
2. Clear the PSR P bit.
3. Copy the PSR into a temmporary register, then clear PSR bits T, V, U, S and I.
4. Set ''Vector'' to the value corresponding to the exception type:

   Abort: Vector = 2

   Restartable Bus Error: Vector = 11
5. Set ''Return Address'' to the address of the first byte of the suspended instruction.
6. Perform Service (Vector, Return Address), *Figure 3-15*.

3.2.8.3 SLAVE/ILL/SVC/DVZ/FLG/BPT/UND Trap Sequence

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.

   SLAVE: Vector = 3.
   ILL: Vector = 4.
   SVC: Vector = 5.
   DVZ: Vector = 6.
   FLG: Vector = 7.
   BPT: Vector = 8.
   UND: Vector = 10.
3. If Trap (ILL) or Trap (UND)
   a. Clear the Processor Status Register P bit.
4. Copy the Processor Status Register (PSR) into a temporary register, then clear PSR bits T, V, U, S and P.
5. Set ''Return Address'' to the address of the first byte of the trapped instruction.
6. Perform Service (Vector, Return Address), *Figure 3-15*.

3.2.8.4 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 T, V, U and S.
3. Set ''Vector'' to 9.
4. Set ''Return Address'' to the address of the next instruction.
5. Perform Service (Vector, Return Address), *Figure 3-15.*

3.2.8.5 Integer-Overflow Trap Sequence

1. Copy the PSR into a temporary register, then clear PSR bits T, V, U, S and P.
2. Set ''Vector'' to 13.
3. Set ''Return Address'' to the address of the next instruction.

4. Perform Service (Vector, Return Address), *Figure 3-15*.

3.2.8.6 Debug Trap Sequence

A debug condition can be recognized either at the next instruction boundary or, in the case of an interruptible instruction, at the next interruptible point during its execution.

1. If PC-match condition, then go to Step 3.
2. 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 instruction.
   c. Go to Step 4.
3. Set ''Return Address'' to the address of the next instruction.
4. Set ''Vector'' to 14.
5. Copy the Processor Status Register (PSR) into a temporary register, then clear PSR bits T, V, U, S, P and I.
6. Perform Service (Vector, Return Address), *Figure 3-15*.

Note: In case of PC-match or address-compare on write, the Trap (DBG) may occur before the instruction is executed.

3.2.8.7 Non-Restartable Bus Error Sequence

1. Set ''Vector'' to 12.
2. Set ''Return Address'' to ''Undefined''.
3. Copy the Processor Status Register (PSR) into a temporary register, then clear PSR bits T, V, U, S, P and I.
4. Perform a dummy read of the Slave Status Word to reset the Slave Processor.
5. Perform Service (Vector, Return Address), *Figure 3-15*.

3.3 DEBUGGING SUPPORT

The NS32532 provides serveral features to assist in program debugging.

Besides the Breakpoint (BPT) instruction that can be used to generate soft breaks, the CPU also provides instruction tracing as well as debug trap (or hardware breakpoints) capabilities. Details on these features are provided in the following sub-sections.

3.3.1 Instruction Tracing

Instruction tracing is a very useful feature that can be used during debugging to single-step through selected portions of a program. Tracing is enabled by setting the T-bit in the PSR Register. When enabled, the CPU generates a Trace Trap (TRC) after the execution of each instruction.

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 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.

Due to the fact that some instructions can clear the T and P bits in the PSR, in some cases a Trace Trap may not occur at the end of the instruction. This happens when one of the privileged instructions BICPSRW or LPRW PSR is executed.

TABLE 3-3. Summary of Exception Processing

Exception	Instruction Ending	Cleared Before Saving PSR	Cleared After Saving PSR
Restartable Bus Error	Suspended	P	TVUSI
Nonrestartable Bus Error	Terminated	Undefined	TVUS
Interrupt	Before Instruction	None/P*	TVUSPI
ABT	Suspended	P	TVUSI
ILL, UND	Suspended	P	TVUS
SLAVE, SVC, DVZ, FLG, BPT	Suspended	None	TVUSP
OVF	Completed	None	TVUSP
TRC	Before Instruction	P	TVUS
DBG	Before Instruction	None/P*	TVUSPI

***Note:** The P bit of the saved PSR is cleared in case the exception is acknowledged before the instruction is completed (e.g., interrupted string instruction). This is to avoid a mid-instruction trace trap upon return from the Exception Service Routine.

Service (Vector, Return Address):

1) Push the PSR copy onto the Interrupt Stack as a 16-bit value.

2) If Direct-Exception mode is selected, then go to step 4.

3) Push MOD Register into the Interrupt Stack as a 16-bit value.

4) Read 32-bit Interrupt Dispatch Table (IDT) entry at address 'INTBASE + vector × 4'.

5) If Direct-Exception mode is selected, then go to Step 10.

6) Move the L.S. word of the IDT entry (Module Field) into the MOD register.

7) Read the Program Base pointer from memory address 'MOD + 8', and add to it the M.S. word of the IDT entry (Offset Field), placing the result in the Program Counter.

8) Read the new Static Base pointer from the memory address contained in MOD, placing it into the SB Register.

9) Go to Step 11.

10) Place IDT entry in the Program Counter.

11) Push the Return Address onto the Interrupt Stack as a 32-bit quantity.

12) Serialize: Non-sequentially fetch first instruction of Exception Service Routine.

Note: Some of the Memory Accesses indicated in the service sequence may be performed in an order different from the one shown.

FIGURE 3-15. Service Sequence

In other cases, it is still possible to guarantee that a Trace Trap occurs at the end of the instruction, provided that special care is taken before returning from the Trace Trap Service Procedure. In case a BICPSRB instruction has been executed, the service procedure should make sure that the T bit in the PSR copy saved on the Interrupt Stack is set before executing the RETT instruction to return to the program begin traced. If the RETT or RETI instructions have to be traced, the Trace Trap Service Procedure should set the P and T bits in the PSR copy on the Interrupt Stack that is going to be restored in the execution of such instructions.

Note: If instruction tracing is enabled while the WAIT instruction is executed, the Trap (TRC) occurs after the next interrupt, when the interrupt service procedure has returned.

3.3.2 Debug Trap Capability

The CPU recognizes three different conditions to generate a Debug Trap:

1) Address Compare

2) PC Match

3) External

These conditions can be enabled and monitored through the CPU Debug Registers.

An address-compare condition is detected when certain memory locations are either read or written. The double-word address used for the comparison is specified in the CAR Register. The address-compare condition can be separately enabled for each of the bytes in the specified double-word, under control of the CBE bits of the DCR Register. The VNP bit in the DCR controls whether virtual or physical addresses are compared. The CRD and CWR bits in the DCR separately enable the address compare condition for read and write references; the CAE bit in the DCR can be used to disable the compare-address condition independently from the other control bits. The CPU examines the address compare condition for all data reads and writes, reads of memory locations for effective address calculations, Interrupt-Acknowledge and End-of-Interrupt bus cycles, and memory references for exception processing. An address-compare condition is not detected for MMU references to Page Table Entries.

The PC-match condition is detected when the address of the instruction equals the value specified in the BPC register. The PC-match condition is enabled by the PCE bit in the DCR.

Detection of address-compare and PC-match conditions is enabled for User and Supervisor Modes by the UD and SD bits in the DCR. The DEN-bit can be used to disable detection of these two conditions independently from the other control bits.

An external condition is recognized whenever the $\overline{\text{DBG}}$ signal is activated.

When the CPU detects an address-compare or PC-match condition while executing an instruction or processing an exception, then Trap (DBG) occurs if the TR bit in the DCR is 1. When an external debug condition is detected, Trap (DBG) occurs regardless of the TR bit. The cause of the Trap (DBG) is indicated in the DSR Register.

When an address-compare or PC-match condition is detected while executing an instruction, the CPU asserts the $\overline{\text{BP}}$ signal at the beginning of the next instruction, synchronously with $\overline{\text{PFS}}$. If the instruction is not completed because a higher priority trap (i.e., ABORT) is detected, the $\overline{\text{BP}}$ signal may or may not be asserted.

Note 1: The assertion of $\overline{\text{BP}}$ is not affected by the setting of the TR bit in the DCR register.

Note 2: While executing the MOVUS and MOVSU instructions, the compare-address condition is enabled for the User space memory reference under control of the UD-bit in the DCR.

Note 3: When the LPRi instruction is executed to load a new value into the BPC, CAR or DCR, it is undefined whether the address-compare and PC-match conditions, in effect while executing the instruction, are detected under control of the old or new contents of the loaded register. Therefore, any LPRi instruction that alters the control of the address-compare or PC-match conditions should use register or immediate addressing mode for the source operand.

Note 4: If an exception occurred during the previous instruction, trap (DBG) may be taken prior to instruction execution.

3.4 ON-CHIP CACHES

The NS32532 provides three on-chip caches: the Instruction Cache (IC), the Data Cache (DC) and the Translation Look-aside Buffer (TLB).

The first two are used to hold the contents of frequently used memory locations, while the TLB holds address-translation information.

The IC and DC can be individually enabled by setting appropriate bits in the CFG Register (See Section 2.1.4); the TLB is automatically enabled when address-translation is enabled.

The CPU also provides a locking feature that allows the contents of the IC and DC to be locked to specific memory locations. This is accomplished by setting the LIC and LDC bits in the CFG register.

Cache locking can be successfully used in real-time applications to guarantee fast access to critical instruction and data areas.

Details on the organization and function of each of the caches are provided in the following sections.

Note: The size and organization of the on-chip caches may change in future Series 32000 microprocessors. This however, will not affect software compatibility.

3.4.1 Instruction Cache (IC)

The basic structure of the instruction cache (IC) is shown in *Figure 3-16*.

The IC stores 512 bytes of code in a direct-mapped organization with 32 sets. Direct-mapped means that each set contains only one block, thus each memory location can be loaded into the IC in only one place.

Each block contains a 23-bit tag, which holds the most-significant bits of the physical address for the locations stored in the block, along with 4 double-words and 4 validity bits (one for each double-word).

A 4-double-word instruction buffer is also provided, which is loaded either from a selected cache block or from external memory. Instructions are read from this buffer by the loader unit and transferred to an 8-byte instruction queue.

The IC may or may not be enabled to cache an instruction being fetched by the CPU. It is enabled when the IC bit in the CFG Register is set to 1 and either the address translation is disabled or the CI bit in the Level-2 PTE used to translate the virtual address of the instruction is set to 0.

If the IC is disabled, the CPU bypasses it during the instruction fetch and its contents are not affected. The instruction is read directly from external memory into the instruction buffer.

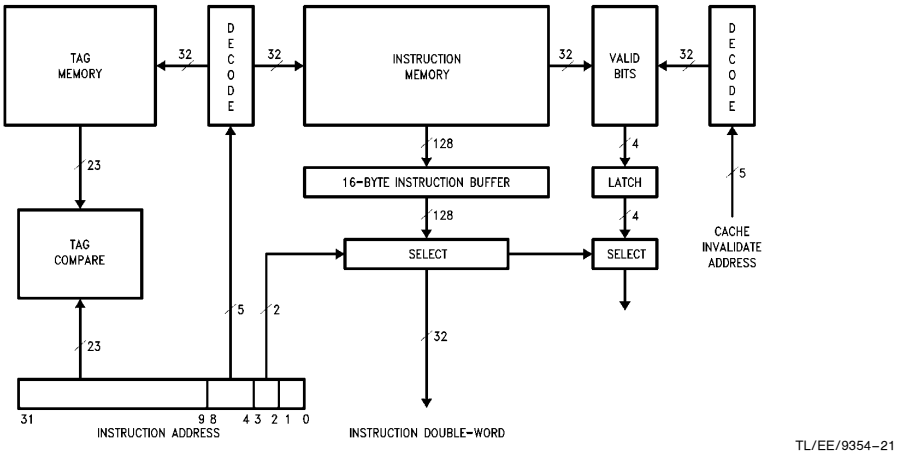


FIGURE 3-16. Instruction Cache Structure

When the IC is enabled, the instruction address bits 4 to 8 are used to select the IC set where the instruction may be stored. The tag corresponding to the single block in the set is compared with the 23 most-significant bits of the instruction's physical address. The 4 double-words in this block are loaded into the instruction buffer and the 4 validity bits are also retrieved. Bits 2 and 3 of the instruction's physical address select one of these double-words and the associated validity bit.

If the tag matches and the selected double-word is valid, a cache 'hit' occurs and the double-word is directly transferred to the instruction queue for decoding; otherwise a cache 'miss' will result.

In the latter case, if the cache is not locked, the CPU will take the following actions.

First, if the tag of the selected block does not match, the tag is loaded with the 23 most-significant bits of the instruction address and all the validity bits are cleared. Then, the instruction is read from external memory into the instruction buffer.

If the CIIN input signal is not active during the fetching of the missing instruction, then the IC is updated and the instruction double-words fetched from memory are stored into it with the validity bits set.

If the cache is locked, its contents are not affected, as the CPU reads the missing instruction from external memory.

Whenever the CPU accesses external memory, whether or not the IC is enabled, it always fetches instruction double-words in a non-wrap-around fashion. Refer to Sections 3.5.4.3 and 3.5.6 for more information.

The contents of the instruction cache can be invalidated by software through the CINV instruction or by hardware through the appropriate cache invalidation input signals. Clearing the IC bit in the CFG Register also invalidates the instruction cache. Refer to Sections 3.5.10 and C.3 for details.

Note: If the IC is enabled for a certain instruction and a 'miss' occurs due to a tag mismatch, the CPU will clear all the validity bits of the selected tag before fetching the instruction from external memory. If the CIIN input signal is activated during the fetching of that instruction, the validity bits are not set and the IC is not updated.

3.4.2 Data Cache (DC)

The Data Cache (DC) stores 1,024 bytes of data in a two-way set associative organization as shown in *Figure 3-17*.

Each of the 32 sets has 2 cache blocks. Each block contains a 23-bit tag, which holds the most-significant bits of the physical address for the locations stored in the block, along with 4 double-words and 4 validity bits (one for each double-word).

The DC is enabled for a data read when all of the following conditions are satisfied.

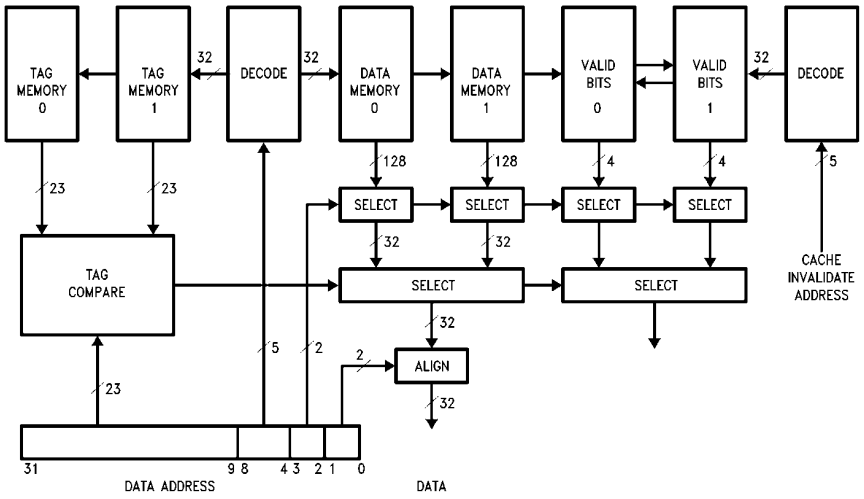
- The DC bit in the CFG Register is set to 1.
- Either the address translation is disabled or the CI bit in the Level-2 PTE used to translate the virtual address of the data reference is set to 0.
- The reference is not an interlocked read resulting from executing a CBITI or SBITI instruction.

If the DC is disabled, the CPU bypasses it during the data read and its contents are not affected. The data is read directly from external memory. The DC is also bypassed for MMU reads from Page Table entries during address translation and for Interrupt-Acknowledge and End-of-Interrupt bus cycles.

When the DC is enabled for a data read, the address bits 4 to 8 are used to select the DC set where the data may be stored.

The tags corresponding to the two blocks in the set are compared to the 23 most-significant bits of the physical address. Bits 2 and 3 of the address select one double-word in each block and the associated validity bit.

If one of the tag matches and the selected double-word in the corresponding block is valid, a cache 'hit' occurs and the data is used to execute the instruction; otherwise a cache 'miss' will result. In the latter case, if the cache is not locked, the CPU will take the following actions.



TL/EE/9354–22

FIGURE 3-17. Data Cache Structure

First, if the tag of either block in the set matches the data address, that block is selected for updating. Otherwise, if neither tag matches, then the least recently used block is selected; its tag is loaded with the 23 most-significant bits of the data address, and all the validity bits are cleared.

Then, the data is read from external memory; up to 4 double-word bits are read into the cache in a wrap-around fashion. Refer to Sections 3.5.4.3 and 3.5.6 for more information.

If the CIIN and $\overline{\text{IODEC}}$ input signals are both inactive during the bus cycles performed to read the missing data, then the DC is updated, as each double-word is read from memory, and the corresponding validity bit is set. If the cache is locked, its contents are not affected, as the CPU reads the missing data from external memory.

The DC is enabled for a data write whenever the DC bit in the CFG Register is set to 1, including interlocked writes resulting from executing the CBITI and SBITI instructions, and MMU writes to Page Table entries during address translation.

The DC does not use write allocation. This means that, during a write, if a cache 'hit' occurs, the DC is updated, otherwise it is unaffected. The data is always written through to external memory.

The contents of the data cache can be invalidated by software through the CINV instruction or by hardware through the appropriate cache invalidation input signals. Clearing the DC bit in the CFG Register also invalidates the data cache. Refer to Sections 3.5.10 and C.3 for details.

Note: If the DC is enabled for a certain data reference and a ''miss'' occurs due to tag mismatch, the CPU will clear all the validity bits for the least recently used tag before reading the data from external memory. If either CIIN or $\overline{\text{IODEC}}$ are activated during the data read bus cycles, the validity bits are not set and the DC is not updated.

3.4.3 Cache Coherence Support

The NS32532 provides several mechanisms for maintaining coherence between the on-chip caches and external memory. In software, the use of caches can be inhibited for individual pages using the CI-bit in the level-2 Page Table Entries. The CINV instruction can be executed to invalidate entriely the Instruction Cache and/or Data Cache; the CINV instruction can also be executed to invalidate a single 16-byte block in either or both caches.

In hardware, the use of the caches can be inhibited for individual locations using the CIIN input signal. A cache invalidation request can cause the entire Instruction Cache and/or Data Cache to be invalidated; a cache invalidation request can also cause invalidation of a single set in either or both caches. Refer to Section 3.5.7 for more information.

An external ''Bus Watcher'' circuit can also be used to help maintain cache coherence. The Bus Watcher observes the CPU's bus cycles to maintain a copy of the on-chip cache tags while also monitoring writes to main memory by DMA controllers and other microprocessors in the system. When the Bus Watcher detects that a location in one of the on-chip caches has been modified in main memory, it issues an invalidation request to the CPU. The CPU provides the necessary information on the system interface to help maintain an external copy of the on-chip tags.

The status codes differentiate between instruction fetches and data reads.

The set, affected during the bus access (if CIOUT is low), as well as the tag can be determined from the address bits A4 through A8 and A9 through A31 respectively.

During a data read the CPU also indicates, by means of the CASEC signal, which block in the set is being updated.

Whenever a CINV instruction is executed, the operation code and operand appear on the system interface using slave processor bus cycles. Thus, invalidations of the on-chip caches by software can be monitored externally.

Note, however, that the software is responsible for communicating to the external circuitry the values of the cache enable and lock bits in the CFG Register, since the CPU does not generate any special cycle (e.g., Slave Cycle) when the CFG Register is loaded.

3.4.4 Translation Look-aside Buffer (TLB)

The Translation Look-aside Buffer is an on-chip fully associative memory. It provides direct virtual to physical mapping for 64 pages, thus minimizing the time needed to perform the address translation.

The efficiency of the on-chip MMU is greatly increased by the TLB, which bypasses the much longer Page Table lookup in over 99% of the accesses made by the CPU.

Entries in the TLB are allocated and replaced automatically; the operating system is not involved. The TLB entries cannot be read or written by software; however, they can be purged from it under program control.

Figure 3-18 shows a model of the TLB. Information is placed into the TLB whenever a Page Table lookup is performed. If the retrieved mapping is valid (V = 1 in both levels of the Page Tables), and the access attempted is permitted by the protection level, an entry of the TLB is loaded from the information retrieved from memory.

The on-chip MMU places the Virtual Page Number (VPN) and the Address Space qualifier (AS) into the tag portion of the TLB entry.

The value portion of the entry is loaded from the Page Tables as follows:

- The PFN field (20 bits) as well as the CI and M bits are loaded from the Level-2 Page Table Entry (PTE2).
- The PL field (2 bits) is loaded to reflect the most restrictive of the protection levels imposed by the PL fields of the Level-1 and Level-2 Page Table Entries (PTE1 and PTE2).

Not shown in the figure is an additional bit associated with each TLB entry which indicates whether the entry is valid.

Address translation can be either enabled or disabled for a memory reference. If translation is disabled, then the TLB is bypassed and the physical address is identical to the virtual address.

When translation is enabled and a virtual address needs to be translated, the high-order 20 bits (VPN) and the Address Space qualifier are compared associatively to the corresponding fields in all entries of the TLB.

For a read reference, if the tag portion of a valid TLB entry, completely matches the input values, then the value portion of the entry is used to complete the address translation and protection checking.

For a write reference, if a valid entry with a matching tag is present in the TLB, then the M bit is examined. If the M bit is 1, the value portion of the entry is used to complete the address translation and protection checking. If the M bit is 0, the entry is invalidated.

In either case, if a protection level violation is detected, a translation exception (Trap (ABT)) is generated. When no matching entry is found or a matching entry is invalidated because the M bit is 0 in a write reference, a Page Table lookup is performed. The virtual address is translated according to the algorithm given in Section 2.4.5 and the translation information is loaded into the TLB.

The recipient entry is selected by an on-chip circuit that implements a First-In-First-Out (FIFO) algorithm.

Note that for a translation to be loaded into the TLB it is necessary that the Level-1 and Level-2 Page Table Entries be valid (V bit = 1). Also, it is guaranteed that in the process of loading a TLB entry (during a Page Table lookup) the Level-1 and Level-2 R bits will be set in memory if they were not already set. For these reasons, there is no need to replicate either the V bit or the R bit in the TLB entries.

Whenever a Page Table Entry in memory is altered by software, it is necessary to purge any matching entry from the TLB, otherwise the corresponding addresses would be translated according to obsolete information. TLB entries may be selectively purged by writing a virtual address to one of the IVARn registers using the LMR instruction. The TLB entry (if any) that matches that virtual address is then purged, and its space is made available for another translation. Purging is also performed whenever an address space is remapped by altering the contents of the PTB0 or PTB1 register. When this is done, all the TLB entries corresponding to the address space mapped by that register are purged. Turning translation on or off (via the MCR TU and TS bits) does not affect the contents of the TLB.

It is possible to maintain an external copy of the valid contents of the on-chip TLB by observing the CPU's system interface during the replacement and invalidation of TLB entries. Whenever the CPU replaces a TLB entry, the page tables are accessed in external memory using bus cycles with a special Status. Because a FIFO replacement algorithm is used, it is possible to determine which entry is being replaced by using a 6-bit counter that is incremented whenever a Level-1 PTE is accessed. The contents of the new entry can be found as follows:

- VPN appears on A2 through A11 during the PTE1 and PTE2 accesses. The most-significant 10 bits appear during the PTE1 access, and the least-significant 10 bits appear during the PTE2 access.
- AS can be determined from the U/$\overline{S}$ signal during the PTE1 access.
- PFN, M and CI can be determined from the PTE2 value read on the Data Bus. PL can be determined from the most restrictive of the PTE1 and PTE2 values read on the Data Bus.

Whenever a LMR instruction is executed, the operation code and operand appear on the system interface using slave processor bus cycles. Thus, the information is available externally to determine the translation modes controlled by the MCR and to identify that a TLB entry has been invalidated.

When the PTB0 register is loaded by executing the 'LMR PTB0 src' instruction, the internal FIFO pointer is also reset to point to the first TLB entry.

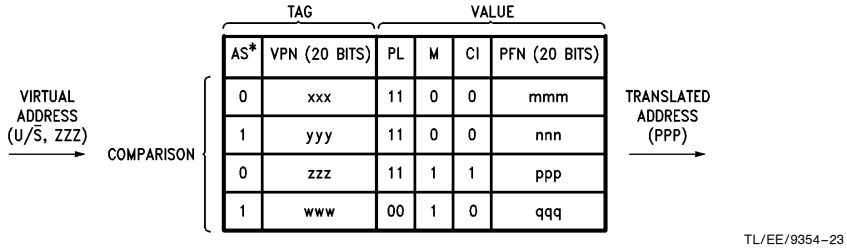
Note that the contents of the TLB maintained externally include copies of all valid entries in the on-chip TLB, but the external copy may include some entries that are invalid in the on-chip TLB. For example, when the TLB is searched for a write reference and a matching entry is found with the M bit clear, then the on-chip entry is invalidated and a miss is processed. It is not possible to detect externally that the old matching entry on-chip has been invalidated.

3.5 SYSTEM INTERFACE

This section provides general information on the NS32532 interface to the external world. Descriptions of the CPU requirements as well as the various bus characteristics are provided here. Details on other device characteristics including timing are given in Chapter 4.

3.5.1 Power and Grounding

The NS32532 requires a single 5-volt power supply, applied on 21 pins. The logic voltage pins (VCCL1 to VCCL6) supply



*AS represents the virtual address space qualifier.

FIGURE 3-18. TLB Model

the power to the on-chip logic. The buffer voltage pins (VCCB1 to VCCB14) supply the power to the output drivers of the chip. The bus clock power pin (VCCCLK) is the power supply for the on-chip clock drivers. All the voltage pins should be connected together by a power (VCC) plane on the printed circuit board.

The NS32532 grounding connections are made on 20 pins. The logic ground pins (GNDL1 to GNDL6) are the ground pins for the on-chip logic. The buffer ground pins (GNDB1 to GNDB13) are the ground pins for the output drivers of the chip. The bus clock ground pin (GNDCLK) is the ground connection for the on-chip clock drivers. All the ground pins should be connected together by a ground plane on the printed circuit board.

Both power and ground connections are shown in *Figure 3-19*.

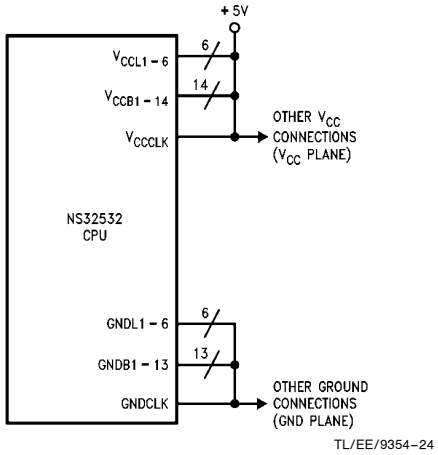


FIGURE 3-19. Power and Ground Connections

3.5.2 Clocking

The NS32532 requires a single-phase input clock signal (CLK) with frequency twice the CPU's operating frequency.

This clock signal is internally divided by two to generate two non-overlapping phases PHI1 and PHI2. One single-phase clock signal BCLK in phase with PHI1 and its complement $\overline{\text{BCLK}}$, are also generated and output by the CPU for timing reference.

Following power-on, the phase relationship between BCLK and CLK is undefined. Nevertheless, in some systems it may be necessary to synchronize the CPU bus timing to an external reference. The $\overline{\text{SYNC}}$ input signal can be used to initialize the phase relationship between CLK and BCLK. $\overline{\text{SYNC}}$ can also be used to stretch BCLK (Low) while CLK is toggling.

$\overline{\text{SYNC}}$ is sampled on each rising edge of CLK. As shown in *Figure 3-20*, whenever $\overline{\text{SYNC}}$ is sampled low, BCLK stops toggling and stays low. On the first rising edge that $\overline{\text{SYNC}}$ is sampled high, BCLK is driven high and then toggles on each subsequent rising edge of CLK.

Every rising edge of BCLK 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.

Note: The CPU requirement on the maximum period of BCLK must be satisfied when $\overline{\text{SYNC}}$ is asserted at times other than reset.

3.5.3 Resetting

The $\overline{\text{RST}}$ input pin is used to reset the NS32532. The CPU samples $\overline{\text{RST}}$ synchronously on the rising edge of BCLK. Whenever a low level is detected, the CPU responds immediately. Any instruction being executed is terminated; any results that have not yet been written to memory are discarded; and any pending bus errors, interrupts, and traps are eliminated. The internal latches for the edge-sensitive $\overline{\text{NMI}}$ and $\overline{\text{DBG}}$ signals are cleared.

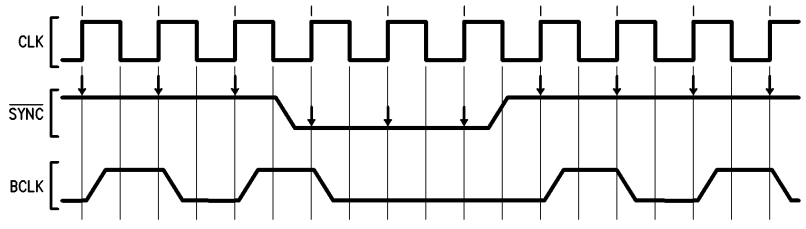


FIGURE 3-20. Bus Clock Synchronization

3.0 Functional Description (Continued)

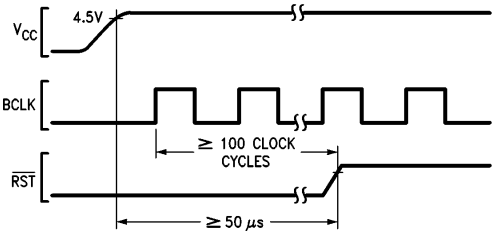
The CPU stores the PC contents in the R0 Register and the PSR contents in the least-significant word of R1, leaving the most-significant word undefined. The PC is then cleared to 0 and so are all the implemented bits in the PSR, MSR, MCR and CFG registers. The DEN-bit in the DCR Register is also cleared to 0. After reset, the remaining implemented bits in DCR and the contents of all other registers are undefined. The CPU begins executing the instruction at Address 0.

On application of power, $\overline{RST}$ 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 also remain active for not less than 64 BCLK cycles. See *Figures 3-21* and *3-22*.

While in the Reset state, the CPU drives the signals $\overline{ADS}$, $\overline{BE}$0–3, $\overline{BMT}$, $\overline{CONF}$ and $\overline{HLDA}$ inactive. The data bus is floated and the state of all other output signals is undefined.

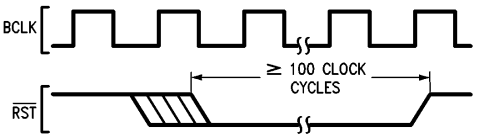
Note 1: If $\overline{HOLD}$ is active at the time $\overline{RST}$ is deasserted, the CPU acknowledges $\overline{HOLD}$ before performing any bus cycle.

Note 2: If $\overline{SYNC}$ is asserted while the CPU is being reset, then BCLK does not toggle. Consequently, $\overline{SYNC}$ must be high for at least 128 CLK cycles while $\overline{RST}$ is low.



TL/EE/9354–26

FIGURE 3-21. Power-On Reset Requirements



TL/EE/9354–27

FIGURE 3-22. General Reset Timing

3.5.4 Bus Cycles

The NS32532 CPU will perform bus cycles for one of the following reasons:

1. To fetch instructions from memory.
2. To write or read data to or from memory or peripheral devices. Peripheral input and output are memory mapped in the Series 32000 family.
3. To read and update Page Table Entries in memory to perform memory management functions.
4. To acknowledge an interrupt and allow external circuitry to provide a vector number, or to acknowledge completion of an interrupt service routine.
5. To transfer information to or from a Slave Processor.

In terms of bus timing, cases 1 through 4 above are identical. For timing specifications, see Section 4. The only external difference between them is the 5-bit code placed on the Bus Status pins (ST0–ST4). Slave Processor cycles differ in that separate control signals are applied (Section 3.5.4.7).

3.5.4.1 Bus Status

The CPU presents five bits of Bus Status information on pins ST0–ST4. 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 is it idle.

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

00000 The bus is idle because the CPU does not yet need to access the bus.

00001 The bus is idle because the CPU is waiting for an interrupt following execution of the WAIT instruction.

00010 The bus is idle because the CPU has halted after detecting an abort or bus error while processing an exception.

00011 The bus is idle because the CPU is waiting for a Slave Processor to complete executing an instruction.

00100 Interrupt Acknowledge, Master.

The CPU is reading an interrupt vector to acknowledge an interrupt request.

00101 Interrupt Acknowledge, Cascaded.

The CPU is reading an interrupt vector to acknowledge a maskable interrupt request from a Cascaded Interrupt Control Unit.

00110 End of Interrupt, Master.

The CPU is performing a read cycle to indicate that it is executing a Return from Interrupt (RETI) instruction at the completion of an interrupt's service procedure.

00111 End of Interrupt, Cascaded.

The CPU is performing a read cycle from a Cascaded Interrupt Control Unit to indicate that it is executing a Return from Interrupt (RETI) instruction at the completion of an interrupt's service procedure.

01000 Sequential Instruction Fetch.

The CPU is fetching the next double-word in sequence from the instruction stream.

01001 Non-Sequential Instruction Fetch.

The CPU is fetching the first double-word of a new sequence of instruction. This will occur as a result of any JUMP or BRANCH, any exception, or after the execution of certain instructions.

01010 Data Transfer.

The CPU is reading or writing an operand for an instruction, or it is referring to memory while processing an exception.

01011 Read RMW Class Operand.

The CPU is reading an operand with access class of read-modify-write.

01100 Read for Effective Address Calculation.

The CPU is reading a pointer from memory in order to calculate an effective address for Memory Relative or External addressing modes.

01101 Access PTE1 by MMU.

The CPU is reading or writing a Level-1 Page Table Entry while the on-chip MMU is translating virtual address.

01110 Access PTE2 by MMU.

The CPU is reading or writing a Level-2 Page Table Entry while the on-chip MMU is translating a virtual address.

11101 Transfer Slave Processor Operand.

The CPU is transferring an operand to or from a Slave Processor.

11110 Read Slave Processor Status.

The CPU is reading a status word from a slave processor after the slave processor has activated the $\overline{FSSR}$ signal.

11111 Broadcast Slave Processor ID + OPCODE.

The CPU is initiating the execution of a Slave Instruction by transferring the first 3 bytes of the instruction, which specify the Slave Processor identification and operation.

3.5.4.2 Basic Read and Write Cycles

The sequence of events occurring during a basic CPU access to either memory or peripheral device is shown in *Figure 3-23* for a read cycle, and *Figure 3-24* for a write cycle.

The cases shown assume that the selected memory or peripheral device is capable of communicating with the CPU at full speed. If not, then cycle extension may be requested through the $\overline{RDY}$ line. See Section 3.5.4.4.

A full speed bus cycle is performed in two cycles of the BCLK clock, labeled T1 and T2. For both read and write bus cycles the CPU asserts $\overline{ADS}$ during the first half of T1 indicating the beginning of the bus cycle. From the beginning of T1 until the completion of the bus cycle the CPU drives the Address Bus and other relevant control signals as indicated in the timing diagrams. For cacheable data read cycles the CPU also drives the CASEC signal to indicate the block in the DC set where the data will be stored. If the bus cycle is not cancelled (e.g., state T2 is entered in the next clock cycle), the confirm signal ($\overline{CONF}$) is asserted in the middle of T1. Note that due to a bus cycle cancellation, the $\overline{BMT}$ signal may be asserted at the beginning of T1, and then deasserted before the time in which it is guaranteed valid (see Section 4.4.2).

A confirmed bus cycle is completed at the end of T2, unless a cycle extension is requested. Following state T2 is either state T1 of the next bus cycle, or an idle T-state, if the CPU has no bus cycle to perform.

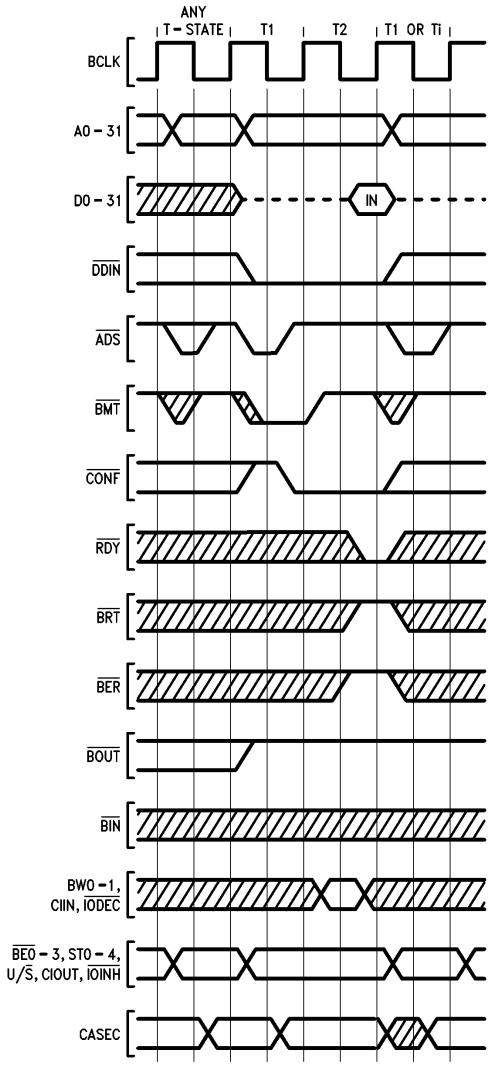
In case of a read cycle the CPU samples the data bus at the end of state T2.

If a bus exception is detected, the data is ignored.

For write bus cycles, valid data is output from the middle of T1 until the end of the cycle. When a write bus cycle is immediately followed by another write cycle, the CPU keeps driving the bus with the data related to the previous cycle until the middle of state T1 of the second bus cycle.

The CPU always inserts an idle state before a write cycle when the write immediately follows a read cycle.

Note: The CPU can initiate a bus cycle with a T1-state and then cancel the cycle, such as when a TLB miss or a Cache hit occurs. In such a case, the $\overline{CONF}$ signal remains High and the $\overline{BMT}$ signal is driven High; the T1-state is followed by another T1-state or an idle T-state.



TL/EE/9354–28

FIGURE 3-23. Basic Read Cycle

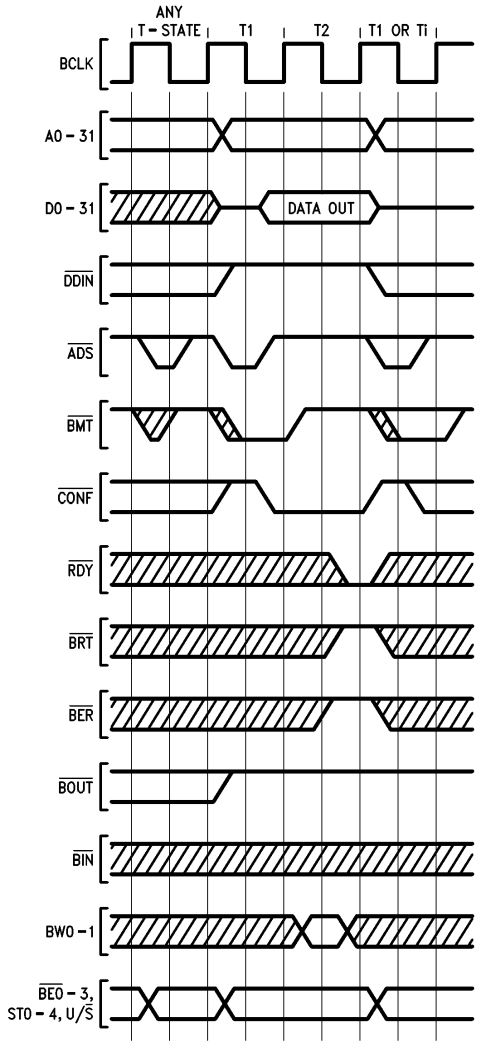


FIGURE 3-24. Write Cycle

3.5.4.3 Burst Cycles

The NS32532 is capable of performing burst cycles in order to increase the bus transfer rate. Burst is only available in instruction fetch cycles and data read cycle from 32-bit wide memories. Burst is not supported in operand write cycles or slave cycles.

The sequence of events for burst cycles is shown in *Figure 3-25*. The case shown assumes that the selected memory is capable of communicating with the CPU at full speed. If not, then cycle extension can be requested through the $\overline{RDY}$ line. See Section 3.5.4.4.

A Burst cycle is composed of two parts. The first part is a regular cycle (opening cycle), in which the CPU outputs the new status and asserts all the other relevant control signals. In addition, the Burst Out Signal ($\overline{BOUT}$) is activated by the CPU indicating that the CPU can perform Burst cycles. If the selected memory allows Burst cycles, it will notify the CPU by activating the burst in signal ($\overline{BIN}$). $\overline{BIN}$ is sampled by the CPU in the middle of T2 on the falling edge of BCLK. If the memory does not allow burst ($\overline{BIN}$ high), the cycle will terminate at the end of T2 and $\overline{BOUT}$ will go inactive immediately. If the memory allows burst ($\overline{BIN}$ low), and the CPU has not deasserted $\overline{BOUT}$, the second part of the Burst cycle will be performed and $\overline{BOUT}$ will remain active until termination of the Burst.

The second part consists of up to 3 nibbles, labeled T2B. In each of them a data item is read by the CPU. For each nibble in the burst sequence the CPU forces the 2 least-significant bits of the address to 0 and increments address bits 2 and 3 to select the next double-word; all the byte enable signals ($\overline{BE}0-3$) are activated.

As shown in *Figures 3-25* and *4-8* (in Section 4), the CPU samples $\overline{RDY}$ at the end of each nibble and extends the access time for the burst transfer if $\overline{RDY}$ is inactive.

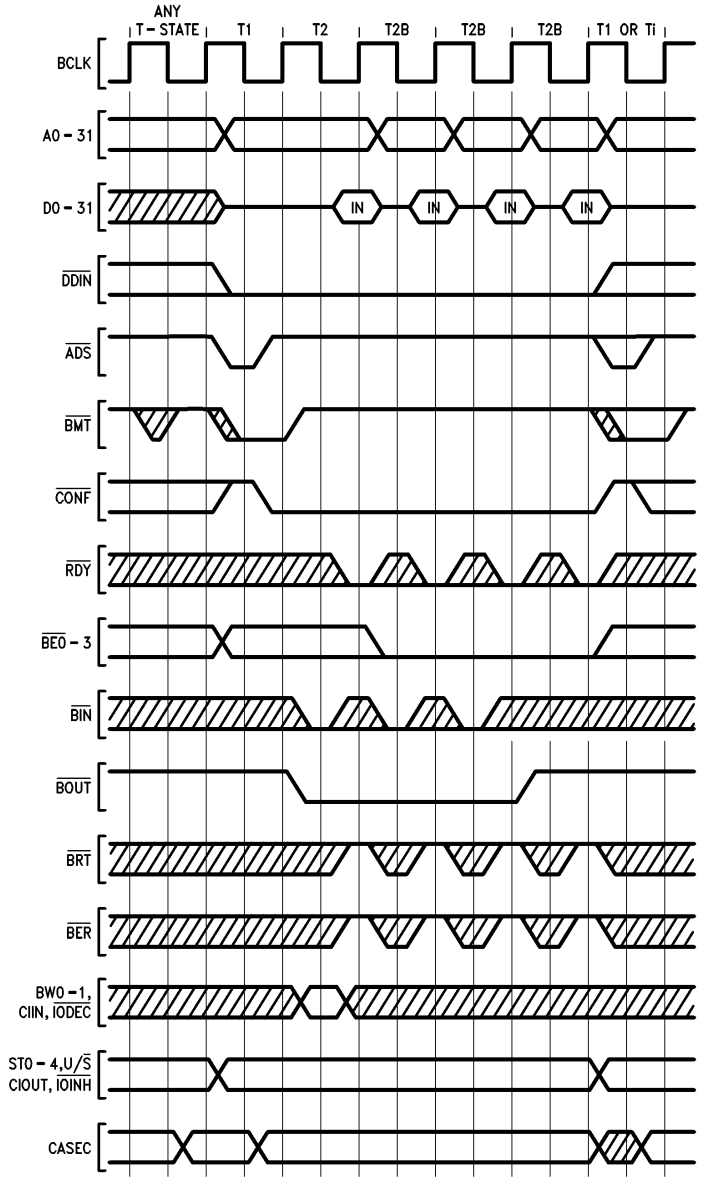
The CPU initiates burst read cycles in the following cases.

1. An instruction must be fetched (Status = 01000 or 01001), and the instruction address does not fall within the last double-word in an aligned 16-byte block (e.g., address bits 2 and 3 are not both equal to 1).
2. A data item must be read (Status = 01010, 01011 or 01100), and all of the following conditions are met.
   - The data cache is enabled and not locked. (DC = 1 and LDC = 0 in the CFG register.)
   - The addressed page is cacheable as indicated in the Level-2 Page Table Entry.
   - The bus cycle is not an interlocked data access performed while executing a CBITI or SBITI instruction.

The Burst sequence will be terminated when one of the following events occurs.

1. The last instruction double-word in an aligned 16-byte block has been fetched.
2. The CPU detects that the instructions being prefetched are no longer needed due to an alteration of the flow of control. This happens, for example, when a Branch instruction is executed or an exception occurs.
3. 4 double-words of data have been read by the CPU. The double-words are transferred within an aligned 16-byte block in a wrap-around order. For example, if a source operand is located at address 104_{16}, then the burst read cycle transfers the double-words at 104, 108, 10C, and 100, in that order.

3.0 Functional Description (Continued)



TL/EE/9354–30

FIGURE 3-25. Burst Read Cycles

4. The $\overline{BIN}$ signal is deasserted.
5. $\overline{BRT}$ is asserted to signal a bus retry.
6. $\overline{IODEC}$ is asserted or the BW0−1 signals indicate a bus width other than 32-bits. The CPU samples these signals during state T2 of the opening cycle. During T2B-states BW0−1 are ignored and $\overline{IODEC}$ must be kept HIGH.

The CPU uses only the values of the above signals sampled during the last state of the transfer when the cycle is extended. See Section 3.5.4.4.

Note: A burst sequence is not stopped by the assertion of either $\overline{BER}$ or CIIN. See Note 3 in Section 3.5.5.

3.5.4.4 Cycle Extension

To allow sufficient access time for any speed of memory or peripheral device, the NS32532 provides for extension of a bus cycle. Any type of bus cycle except a slave processor cycle can be extended.

A bus cycle can be extended by causing state T2 for a normal cycle or state T2B for a Burst cycle to be repeated.

At the end of each T2 or T2B state, on the rising edge of BCLK, the $\overline{RDY}$ line is sampled by the CPU. If $\overline{RDY}$ is active, then the transfer cycle will be completed. If $\overline{RDY}$ is inactive, then the bus cycle is extended by repeating the T-state for another clock cycle. These additional T-states inserted by the CPU in this manner are called 'WAIT' states.

During a transfer the CPU samples the input control signals $\overline{BIN}$, $\overline{BER}$, $\overline{BRT}$, BW0−1, CIIN and $\overline{IODEC}$.

When wait states are inserted, only the values of these signals sampled during the last wait state are significant.

Figures 3-26 and *4-8* (in Section 4) illustrate both a normal read cycle and a Burst cycle with wait states added through the $\overline{RDY}$ pin.

Note: If $\overline{RST}$ is asserted during a bus cycle, then the cycle is terminated without regard of $\overline{RDY}$.

3.5.4.5 Interlocked Bus Cycles

The NS32532 supports indivisible read-modify-write transactions by asserting the $\overline{ILO}$ signal during consecutive read and write operations. See *Figure 4-7* in Section 4.

Interlocked transactions are always preceded and followed by one or more idle T-states.

The $\overline{ILO}$ signal is asserted in the middle of the idle T-state preceding state T1 of the read operation, and is deasserted in the middle of one of the idle T-states following completion of the write operation, including any retried bus cycles.

No other bus operations (e.g., instruction fetches) will occur while an interlocked transaction is taking place.

Interlocked transactions are required in multiprocessor systems to handle shared resources. The CPU uses them to reference data while executing a CBITIi or SBITIi instruction, during which a single byte of data is read and written. They are also used when the on-chip MMU is updating a Level-2 Page Table Entry during a Page Table Lookup.

In this case a double-word is read and written. If the Level-2 Page Tables are located in a memory area whose width is other than 32 bits, multiple interlocked reads followed by multiple interlocked writes will result. The $\overline{ILO}$ signal is always released for one or more clock cycles in the middle of two consecutive interlocked transactions.

Note 1: If a bus error is detected during an interlocked read cycle, the subsequent interlocked write cycle will not be performed, and $\overline{ILO}$ is deasserted before the next bus cycle begins.

Note 2: The CPU may assert $\overline{ILO}$ before a read cycle that is cancelled (for example, due to a TLB miss). In such a case, the CPU deasserts $\overline{ILO}$ before performing any additional bus cycles.

3.5.4.6 Interrupt Control Cycles

The CPU generates Interrupt-Acknowledge bus cycles in response to non-maskable interrupt and enabled maskable interrupt requests.

The CPU also generates one or two End-of-Interrupt bus cycles during execution of the Return-from-Interrupt (RETI) instruction.

The timing for the interrupt control cycles is the same as for the basic memory read cycle shown in *Figure 3-23* ; only the status presented on pins ST0−4 is different. These cycles are single-byte read cycles, and they always bypass the data cache.

Table 3-4 shows the interrupt control sequences associated with each interrupt and with the return from its service procedure.

3.5.4.7 Slave Processor Bus Cycles

The NS32532 performs bus cycles to transfer information to or from slave processors while executing floating-point or custom-slave instructions.

The CPU uses slave write bus cycles to broadcast the identification and operation codes of a slave instruction as well as to transfer operands from memory or general purpose registers to a slave.

Figure 3-27 shows the timing for a slave write bus cycle. The CPU asserts $\overline{SPC}$ during T1; the status is valid during T1 and T2. The operation code or operand is output on the data bus from the middle of T1 until the end of T2.

The CPU uses a slave read bus cycle to transfer a result operand from a slave to either memory or a general purpose register. A slave read cycle is also used to read a status word when the $\overline{FSSR}$ signal is asserted. *Figure 3-28* shows the timing for a slave read bus cycle.

During T1 and T2 the CPU drives the status lines and asserts $\overline{SPC}$. The data from the slave is sampled at the end of T2.

The CPU will never perform another slave cycle immediately following a slave read cycle.

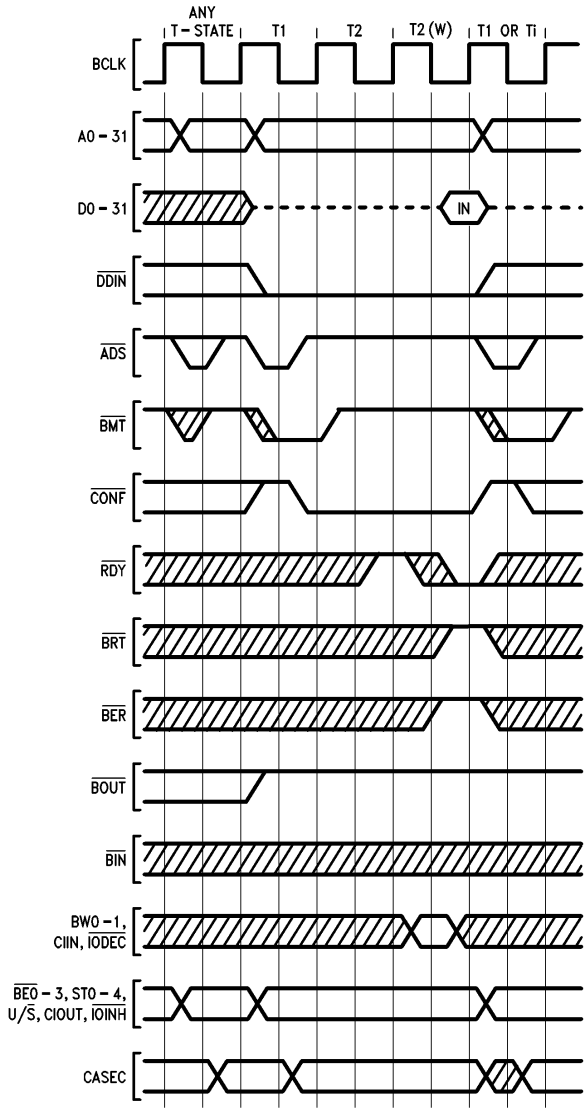
Slave processor data transfers are always 32 bits wide. If the operand is a single byte, then it is transferred on D0 through D7. If it is a word, then it is transferred on D0 through D15.

When two operands are transferred, operand 1 is transferred before operand 2. For double-precision operands, the least-significant double-word is transferred before the most-significant double-word.

During a slave bus cycle the output signals $\overline{BE}$0−3 are undefined while the input signals BW0−1 and $\overline{RDY}$ are ignored.

$\overline{BER}$ and $\overline{BRT}$ must be kept high.

3.0 Functional Description (Continued)



TL/EE/9354–31

3-26. Cycle Extension of a Basic Read Cycle

3.0 Functional Description (Continued)

TABLE 3-4. Interrupt Sequences

								Data Bus			
Cycle	**Status**	**Address**	**$\overline{DDIN}$**	**$\overline{BE3}$**	**$\overline{BE2}$**	**$\overline{BE1}$**	**$\overline{BE0}$**	**Byte 3**	**Byte 2**	**Byte 1**	**Byte 0**
A. Non-Maskable Interrupt Control Sequences											
Interrupt Acknowledge											
1	00100	$FFFFFF00_{16}$	0	1	1	1	0	X	X	X	X
Interrupt Return											
None: Performed through Return from Trap (RETT) instruction.											
B. Non-Vectored Interrupt Control Sequences											
Interrupt Acknowledge											
1	00100	$FFFFFE00_{16}$	0	1	1	1	0	X	X	X	X
Interrupt Return											
1	00110	$FFFFFE00_{16}$	0	1	1	1	0	X	X	X	X
C. Vectored Interrupt Sequences: Non-Cascaded											
Interrupt Acknowledge											
1	00100	$FFFFFE00_{16}$	0	1	1	1	0	X	X	X	Vector: Range: 0–127
Interrupt Return											
1	00110	$FFFFFE00_{16}$	0	1	1	1	0	X	X	X	Vector: Same as in Previous Int. Ack. Cycle
D. Vectored Interrupt Sequences: Cascaded											
Interrupt Acknowledge											
1	00100	$FFFFFE00_{16}$	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	001101	Cascade Address	0	See Note				Vector, range 16–255; on appropriate byte of data bus.			
Interrupt Return											
1	00110	$FFFFFE00_{16}$	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	00111	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

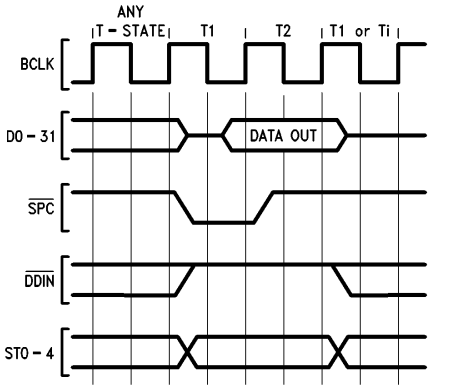


FIGURE 3-27. Slave Processor Write Cycle

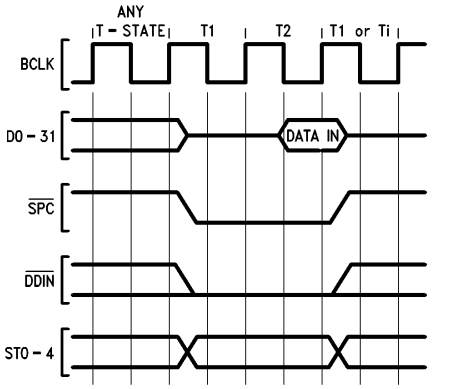


FIGURE 3-28. Slave Processor Read Cycle

3.5.5 Bus Exceptions

The NS32532 has the capability of handling errors occurring during the execution of a bus cycle. These errors can be either correctable or incorrectable, and the CPU can be notified of their occurrence through the input signals $\overline{BRT}$ and/or $\overline{BER}$.

Bus Retry

If a bus error can be corrected, the CPU may be requested to repeat the erroneous bus cycle. The request is done by asserting the $\overline{BRT}$ signal. $\overline{BRT}$ is sampled at the end of state T2 or T2B.

When the CPU detects that $\overline{BRT}$ is active, it completes the bus cycle normally, but ignores the data read in case of a read cycle, and maintains a copy of the data to be written in case of a write cycle. Then, after a delay of two clock cycles, it will start executing the bus cycle again.

If the transfer cycle is multiple (e.g., for non-aligned data), only the problematic part will be repeated.

For instance, if a non-aligned double-word is being transferred and the second half of the transfer fails, only the second part will be repeated.

The same applies for a retry during a burst sequence. The repeated cycle will begin where the read operation failed (rather than the first address of the burst) and will finish the original burst.

Figures 3-29 and *4-10* (in Section 4) show the $\overline{BRT}$ timing for a basic access cycle and for burst cycles respectively.

The CPU always waits for $\overline{BRT}$ to be HIGH before repeating the bus cycle. While $\overline{BRT}$ is LOW, the CPU places all the output signals shown in *Figure 4-11* in a TRI-STATE® condition.

Bus Error

If a bus error is incorrectable the CPU may be requested to interrupt the current process and branch to an appropriate procedure to handle the error. The request is performed by activating the $\overline{BER}$ signal. $\overline{BER}$ is sampled by the CPU at the end of state T2 or T2B on the rising edge of BCLK. When $\overline{BER}$ is sampled active, the CPU completes the bus cycle normally. If a bus error occurs during a bus cycle for a reference required to execute an instruction, then a bus error exception is recognized. However, if an error occurs during an acknowledge cycle of another exception or during the ICU read cycle of a RETI instruction, the CPU interprets the event as a fatal bus error and enters the 'halted' state.

In this state the CPU floats its address and data buses and places a special status code on the ST0–4 lines. The CPU can exit this condition only through a hardware reset. Refer to Section 3.2.6 for more details on bus error.

Note 1: If the erroneous bus cycle is extended by means of wait states, then the CPU uses the values of $\overline{BRT}$ and/or $\overline{BER}$ sampled during the last wait state.

Note 2: If the CPU samples both $\overline{BRT}$ and $\overline{BER}$ active, $\overline{BRT}$ has higher priority. The bus error indication is ignored, and the bus cycle is repeated.

Note 3: If $\overline{BER}$ is asserted during a bus cycle of a multi-cycle data transfer, the CPU completes the entire transfer normally, but the data will be ignored. The CPU also ignores any subsequent assertion of $\overline{BER}$ during the same data transfer.

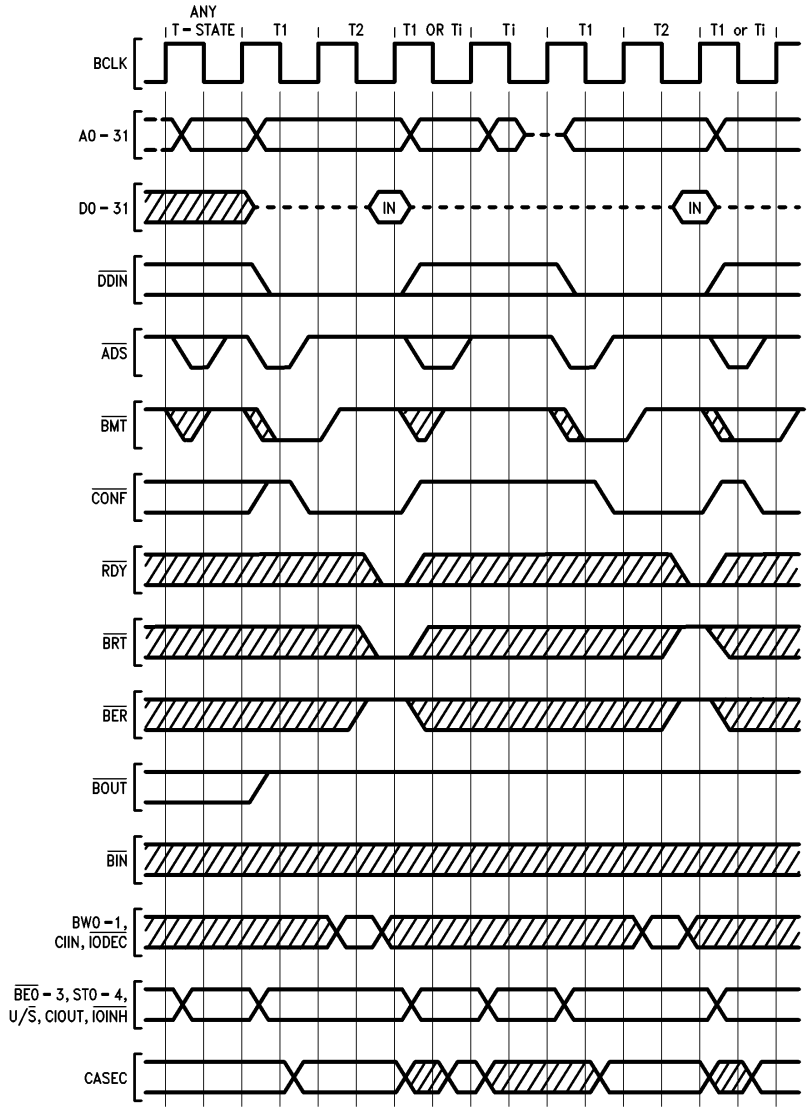
Note 4: Neither $\overline{BRT}$ nor $\overline{BER}$ should be asserted during the T2 state of a slave processor bus cycle.

3.5.6 Dynamic Bus Configuration

The NS32532 is tuned to operate with 32-bit wide memory and peripheral devices. The bus also supports 8-bit and 16-bit data widths, but at reduced efficiency. The CPU can switch from one bus width to another dynamically; the only restriction is that the bus width cannot change for locations within an aligned 16-byte block.

The CPU determines the bus width in effect for a bus cycle by using the values of the BW0 and BW1 signals sampled during the last T2 state. Values of BW0 and BW1 sampled before the last T2 state or during T2B states are ignored. Whenever a bus width other than 32-bit is detected by the CPU, two idle states are inserted before the next bus cycle is initiated. These idle states are only inserted once during an operand access, even if more than two bus cycles are needed to complete the access.

3.0 Functional Description (Continued)



TL/EE/9354–34

FIGURE 3-29. Bus Retry During a Basic Read Cycle

3.0 Functional Description (Continued)

The various combinations for BW0 and BW1 are shown below.

BW1	BW0	
0	0	Reserved
0	1	8-Bit Bus
1	0	16-Bit Bus
1	1	32-Bit Bus

The bus width must always be 32 bits during slave cycles. An important feature of the NS32532 is that it does not impose any restrictions on the data alignment, regardless of the bus width.

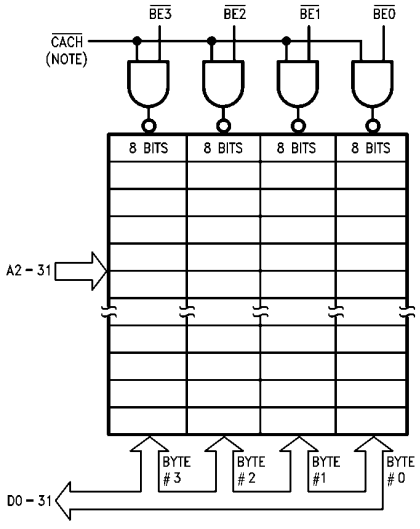
Bus accesses are performed in double-word units. Accesses of data operands that cross double-word boundaries are decomposed into two or more aligned double-word accesses.

The CPU provides four byte enable signals ($\overline{BE}0-3$) which facilitate individual byte accessing on either a 32-bit or a 16-bit bus.

Figures 3-30 and *3-31* show the basic interfaces for 32-bit and 16-bit memories. An 8-bit memory interface (not shown) is even simpler since it does not use any of the $\overline{BE}0-3$ signals and its single bank is always enabled whenever the memory is selected. Each byte location in this case is selected by address bits A0–31.

The NS32532 does not keep track of the bus width used in previous instruction fetches or data accesses. At the beginning of every memory transaction, the CPU always assumes that the bus is 32-bit wide and the $\overline{BE}0-3$ signals are activated accordingly.

The $\overline{BOUT}$ signal is also asserted during instruction fetches or data reads if the conditions for bursting are satisfied. If the bus is other than 32-bit wide, the $\overline{BIN}$ signal is ignored and $\overline{BOUT}$ is deasserted at the beginning of the T state following T2, since burst cycles are not allowed for 8-bit or 16-bit buses.



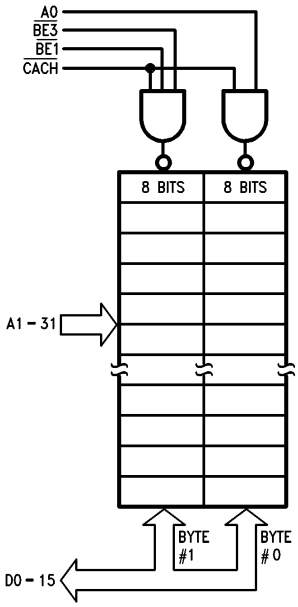
TL/EE/9354–35

FIGURE 3-30. Basic Interface for 32-Bit Memories

Note: The $\overline{CACH}$ signal must be asserted during cacheable read accesses.

The following subsections provide detailed descriptions of the access sequences performed in the various cases.

Note: Although the NS32532 ignores the $\overline{BIN}$ signal for 8-bit and 16-bit bus widths, it is recommended that $\overline{BIN}$ be asserted only if the system supports burst transfers. This is to ensure compatibility with future versions of the CPU that might support burst transfers for 8-bit and 16-bit buses.



TL/EE/9354–36

FIGURE 3-31. Basic Interface for 16-Bit Memories

3.5.6.1 Instruction Fetch Sequences

The CPU performs two types of instruction fetch cycles: sequential and non-sequential. These can be distinguished from each other by the differing status combinations on pins ST0–4. For non-sequential instruction fetches the CPU presents on the address bus the exact byte address of the first instruction in the instruction stream that is about to begin; for sequential instruction fetches, the address of the next aligned instruction double-word is presented on the address bus. The CPU always activates all byte enable signals ($\overline{BE}0-3$) for both sequential and non-sequential fetches. $\overline{BOUT}$ is also asserted during T2 if the addressed double-word is not the last in an aligned 16-byte block. Tables 3-5 to 3-7 show the fetch sequence for the various bus widths.

32-Bit Bus Width

The CPU reads the entire double-word present on the data bus into its internal instruction buffer.

If $\overline{BOUT}$ and $\overline{BIN}$ are both active, the CPU reads up to 3 consecutive double-words using burst cycles. Burst cycles are used for instruction fetches regardless of whether the accesses are cacheable.

3.0 Functional Description (Continued)

Example: JUMP @5

- The CPU performs a fetch cycle at address 5 with $\overline{BE0}$–3 all active.
- Two burst cycles are then performed and addresses 8 and 12 are output while $\overline{BE0}$–3 are kept active.

16-Bit Bus Width

The word on the least-significant half of the data bus is read by the CPU. This is either the even or the odd word within the required instruction double-word, as determined by address bit 1.

The CPU then complements address bit 1, clears address bit 0 and initiates a bus cycle to read the other word, while keeping all the $\overline{BE0}$–3 signals active.

These two words are then assembled into a double-word and transferred into the instruction buffer.

In case of a non-sequential fetch, if the access is not cacheable and the instruction address selects the odd word within the instruction double-word, the even word is not fetched.

Example JUMP @6

- A fetch cycle is performed at address 6 with $\overline{BE0}$–3 all active.
- The word at address 4 is then fetched if the access is cacheable.

8-Bit Bus Width

The instruction byte on the bus lines D0–7 is fetched. The CPU performs three consecutive cycles to read the remaining bytes within the required double-word, while keeping $\overline{BE0}$–3 all active. The 4 bytes are then assembled into a double-word and transferred into the instruction buffer. For a non-sequential fetch, if the access is not cacheable, the CPU will only read the upper bytes within the instruction double-word starting with the byte at the instruction address.

Example: JUMP @7

- The CPU performs a fetch cycle at address 7 with $\overline{BE0}$–3 all active.
- Bytes at addresses 4, 5 and 6 are then fetched consecutively if the access is cacheable.

TABLE 3-5. Cacheable/Non-Cacheable Instruction Fetches from a 32-Bit Bus

1. In a burst access four bytes are fetched with the L.S. bits of the address set to 00.
2. A 'C' on the data bus refers to cacheable fetches and indicates that the byte is placed in the instruction cache. An 'I' refers to non-cacheable fetches and indicates that the byte is ignored.

Number of Bytes	Address LSB	Bytes to be Fetched				Address Bus	$\overline{BE0-3}$	Data Bus			
1	11	B0	—	—	—	A	L L L L	B0	C/I	C/I	C/I
2	10	B1	B0	—	—	A	L L L L	B1	B0	C/I	C/I
3	01	B2	B1	B0	—	A	L L L L	B2	B1	B0	C/I
4	00	B3	B2	B1	B0	A	L L L L	B3	B2	B1	B0

TABLE 3-6. Cacheable/Non-Cacheable Instruction Fetches from a 16-Bit Bus

1. A bus access marked with '*' in the 'Address Bus' column is performed only if the fetch is cacheable.

Number of Bytes	Address LSB	Bytes to be Fetched				Address Bus	$\overline{BE0-3}$	Data Bus			
1	11	B0	—	—	—	A *A − 3	L L L L L L L L	— —	— —	B0 C	C/I C
2	10	B1	B0	—	—	A *A − 2	L L L L L L L L	— —	— —	B1 C	B0 C
3	01	B2	B1	B0	—	A A + 1	L L L L L L L L	— —	— —	B0 B2	C/I B1
4	00	B3	B2	B1	B0	A A + 2	L L L L L L L L	— —	— —	B1 B3	B0 B2

TABLE 3-7. Cacheable/Non-Cacheable Instruction Fetches from an 8-Bit Bus

Number of Bytes	Address LSB	Bytes to be Fetched				Address Bus	$\overline{BE}$0–3	Data Bus			
1	11	B0	—	—	—	A	LLLL	—	—	—	B0
						* A − 3	LLLL	—	—	—	C
						* A − 2	LLLL	—	—	—	C
						* A − 1	LLLL	—	—	—	C
2	10	B1	B0	—	—	A	LLLL	—	—	—	B0
						A + 1	LLLL	—	—	—	B1
						* A − 2	LLLL	—	—	—	C
						* A − 1	LLLL	—	—	—	C
3	01	B2	B1	B0	—	A	LLLL	—	—	—	B0
						A + 1	LLLL	—	—	—	B1
						A + 2	LLLL	—	—	—	B2
						* A − 1	LLLL	—	—	—	C
4	00	B3	B2	B1	B0	A	LLLL	—	—	—	B0
						A + 1	LLLL	—	—	—	B1
						A + 2	LLLL	—	—	—	B2
						A + 3	LLLL	—	—	—	B3

3.5.6.2 Data Read Sequences

The CPU starts a data read access by placing the exact address of the operand on the address bus. The byte enable lines are activated to select only the bytes required by the instruction being executed. This prevents spurious accesses to peripheral devices that might be sensitive to read accesses, such as those which exhibit the characteristic of destructive reading. If the on-chip data cache is internally enabled for the read access, the $\overline{BOUT}$ signal is asserted at the beginning of state T2. $\overline{BOUT}$ will be deasserted if the data cache is externally inhibited (through CIIN or $\overline{IODEC}$), or the bus width is other than 32 bits. During cacheable accesses the CPU always reads all the bytes in the double-word, whether or not they are needed to execute the instruction, and stores them into the data cache. The external memory, in this case, must place the data on the bus regardless of the state of the byte enable signals.

If the data cache is either internally or externally inhibited during the access, the CPU ignores the bytes not selected by the $\overline{BE}$0–3 signals. Data read sequences for the various bus widths are shown in tables 3-8 to 3-10.

32-Bit Bus Width

The entire double-word present on the bus is read by the CPU. If the access is cacheable and the memory allows burst accesses, the CPU reads up to 3 additional double-words within the aligned 16-byte block containing the first byte of the operand. These burst accesses are performed in a wrap-around fashion within the 16-byte block.

Example: MOVW @5, R0

- The CPU reads a double-word at address 5 while keeping $\overline{BE}$1 and $\overline{BE}$2 active.
- If the access is not-cacheable, $\overline{BOUT}$ is deasserted and the data bytes 0 and 3 are ignored.
- If the access is cacheable, the CPU performs burst cycles with $\overline{BE}$0–3 all active, to read the double-words at addresses 8, 12, and 0.

16-Bit Bus Width

The word on the least-significant half of the data bus is read by the CPU. The CPU can then perform another access cycle with address bit 1 complemented and address bit 0 cleared to read the other word within the addressed double-word.

If the access is cacheable, the entire double-word is read and stored into the cache.

If the access is not cacheable, the CPU ignores the bytes in the double-word not selected by $\overline{BE}$0–3. In this case, the second access cycle is not performed, unless selected bytes are contained in the second word.

Example: MOVB @5, R0

- The CPU reads a word at address 5 while keeping $\overline{BE}$1 active.
- If the access is not cacheable, the CPU ignores byte 0.
- If the access is cacheable, the CPU performs another access cycle, with $\overline{BE}$0–3 all active, to read the word at address 6.

8-Bit Bus Width

The data byte on the bus lines D0–7 is read by the CPU. The CPU can then perform up to 3 access cycles to read the remaining bytes in the double-word.

If the access is cacheable, the entire double-word is read and stored into the cache.

If the access is not cacheable, the CPU will only perform those access cycles needed to read the selected bytes.

Example: MOVW @5, R0

- The CPU reads the byte at address 5 while keeping $\overline{BE}$1 and $\overline{BE}$2 active.
- If the access is not cacheable, the CPU activates $\overline{BE}$2 and reads the byte at address 6.
- If the access is cacheable, the CPU performs three bus cycles with $\overline{BE}$0–3 all active, to read the bytes at addresses 6, 7 and 4.

3.0 Functional Description (Continued)

TABLE 3-8. Cacheable/Non-Cacheable Data Reads from a 32-Bit Bus

1. In a burst access four bytes are read with the L.S. bits of the address set to 00.
2. A 'C' on the data bus refers to cacheable reads and indicates that the byte is placed in the data cache. An 'I' refers to non-cacheable reads and indicates that the byte is ignored.

Number of Bytes	Address LSB	Bytes to be Read				Address Bus	$\overline{BE}$0–3	Data Bus			
1	00	—	—	—	B0	A	H H H L	C/I	C/I	C/I	B0
1	01	—	—	B0	—	A	H H L H	C/I	C/I	B0	C/I
1	10	—	B0	—	—	A	H L H H	C/I	B0	C/I	C/I
1	11	BO	—	—	—	A	L H H H	B0	C/I	C/I	C/I
2	00	—	—	B1	B0	A	H H L L	C/I	C/I	B1	B0
2	01	—	B1	B0	—	A	H L L H	C/I	B1	B0	C/I
2	10	B1	B0	—	—	A	L L H H	B1	B0	C/I	C/I
3	00	—	B2	B1	B0	A	H L L L	C/I	B2	B1	B0
3	01	B2	B1	B0	—	A	L L L H	B2	B1	B0	C/I
4	00	B3	B2	B1	B0	A	L L L L	B3	B2	B1	B0

TABLE 3-9. Cacheable/Non-Cacheable Data Reads from a 16-Bit Bus

1. A bus access marked with '*' in the 'Address Bus' column is performed only if the read is cacheable.

Number of Bytes	Address LSB	Data to be Read				Address Bus	$\overline{BE}$0–3		Data Bus			
							Cach.	Non Cach.				
1	00	—	—	—	B0	A *A + 2	H H H L L L L L	H H H L	— —	— —	C/I C	B0 C
1	01	—	—	B0	—	A *A + 1	H H L H L L L L	H H L H	— —	— —	B0 C	C/I C
1	10	—	B0	—	—	A *A − 2	H L H H L L L L	H L H H	— —	— —	C/I C	B0 C
1	11	B0	—	—	—	A *A − 3	L H H H L L L L	L H H H	— —	— —	B0 C	C/I C
2	00	—	—	B1	B0	A *A + 2	H H L L L L L L	H H L L	— —	— —	B1 C	B0 C
2	01	—	B1	B0	—	A A + 1	H L L H L L L L	H L L H H L H H	— —	— —	B0 C/I	C/I B1
2	10	B1	B0	—	—	A *A − 2	L L H H L L L L	L L H H	— —	— —	B1 C	B0 C
3	00	—	B2	B1	B0	A A + 2	H L L L L L L L	H L L L H L H H	— —	— —	B1 C/I	B0 B2
3	01	B2	B1	B0	—	A A + 1	L L L H L L L L	L L L H L L H H	— —	— —	B0 B2	C/I B1
4	00	B3	B2	B1	B0	A A + 2	L L L L L L L L	L L L L L L H H	— —	— —	B1 B3	B0 B2

TABLE 3-10. Cacheable/Non-Cacheable Data Reads from an 8-Bit Bus D8–12

Number of Bytes	Address LSB	Data to be Read				Address Bus	$\overline{BE}$0–3		Data Bus			
							Cach.	Non Cach.				
1	00	—	—	—	B0	A	HHHL	HHHL	—	—	—	B0
						*A + 1	LLLL		—	—	—	C
						*A + 2	LLLL		—	—	—	C
						*A + 3	LLLL		—	—	—	C
1	01	—	—	B0	—	A	HHLH	HHLH	—	—	—	B0
						*A + 1	LLLL		—	—	—	C
						*A + 2	LLLL		—	—	—	C
						*A − 1	LLLL		—	—	—	C
1	10	—	B0	—	—	A	HLHH	HLHH	—	—	—	B0
						*A + 1	LLLL		—	—	—	C
						*A − 2	LLLL		—	—	—	C
						*A − 1	LLLL		—	—	—	C
1	11	B0	—	—	—	A	LHHH	LHHH	—	—	—	B0
						*A − 3	LLLL		—	—	—	C
						*A − 2	LLLL		—	—	—	C
						*A − 1	LLLL		—	—	—	C
2	00	—	—	B1	B0	A	HHLL	HHLL	—	—	—	B0
						A + 1	LLLL	HHLH	—	—	—	B1
						*A + 2	LLLL		—	—	—	C
						*A + 3	LLLL		—	—	—	C
2	01	—	B1	B0	—	A	HLLH	HLLH	—	—	—	B0
						A + 1	LLLL	HLHH	—	—	—	B1
						*A + 2	LLLL		—	—	—	C
						*A − 1	LLLL		—	—	—	C
2	10	B1	B0	—	—	A	LLHH	LLHH	—	—	—	B0
						A + 1	LLLL	LHHH	—	—	—	B1
						*A − 2	LLLL		—	—	—	C
						*A − 1	LLLL		—	—	—	C
3	00	—	B2	B1	B0	A	HLLL	HLLL	—	—	—	B0
						A + 1	LLLL	HLLH	—	—	—	B1
						A + 2	LLLL	HLHH	—	—	—	B2
						*A + 3	LLLL		—	—	—	C
3	01	B2	B1	B0	—	A	LLLH	LLLH	—	—	—	B0
						A + 1	LLLL	LLHH	—	—	—	B1
						A + 2	LLLL	LHHH	—	—	—	B2
						*A − 1	LLLL		—	—	—	C
4	00	B3	B2	B1	B0	A	LLLL	LLLL	—	—	—	B0
						A + 1	LLLL	LLLH	—	—	—	B1
						A + 2	LLLL	LLHH	—	—	—	B2
						A + 3	LLLL	LHHH	—	—	—	B3

3.5.6.3 Data Write Sequences

In a write access the CPU outputs the operand address and asserts only the byte enable lines needed to select the specific bytes to be written.

In addition, the CPU duplicates the data to be written on the appropriate bytes of the data bus in order to handle 8-bit and 16-bit buses.

The various access sequences as well as the duplication of data are summarized in tables 3-11 to 3-13.

32-Bit Bus Width

The CPU performs only one access cycle to write the selected bytes within the addressed double-word.

Example: MOVB R0, @6

- The CPU duplicates byte 2 of the data bus into byte 0 and performs a write cycle at address 6 with $\overline{BE}$2 active.

16-Bit Bus Width

Up to two access cycles are needed to complete the write operation.

Example: MOVW R0, @5

- The CPU duplicates byte 1 of the data bus into byte 0 and performs a write cycle at address 5 with $\overline{BE}1$ and $\overline{BE}2$ active.
- A write at address 6 is then performed with $\overline{BE}2$ active and the original byte 2 of the data bus placed on byte 0.

8-Bit Bus Width

Up to 4 access cycles are needed in this case to complete the write operation.

Example: MOVB R0, @7

- The CPU duplicates byte 3 of the data bus into bytes 0 and 1, and then performs a write cycle at address 7 with $\overline{BE}3$ active.

3.5.7 Bus Access Control

The NS32532 has the capability of relinquishing its control of the bus upon request from a DMA device or another CPU. This capability is implemented with the $\overline{HOLD}$ and $\overline{HLDA}$ signals. By asserting $\overline{HOLD}$, 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 placed all the output signals shown in *Figure 3-32* into the TRI-STATE condition.

To return control of the bus to the CPU, the external device sets $\overline{HOLD}$ inactive, and the CPU acknowledges return of the bus by setting $\overline{HLDA}$ inactive.

The CPU samples $\overline{HOLD}$ in the middle of each T-state on the falling edge of BCLK. If $\overline{HOLD}$ is asserted when the bus is idle between access sequences, then the bus is granted immediately (see *Figure 3-31*). If $\overline{HOLD}$ is asserted during an access sequence, then the bus is granted immediately after the access sequence, including any retried bus cycles, has completed (see *Figure 4-13*). Note that an access sequence can be composed of several bus cycles if the bus width is 8 or 16 bits.

TABLE 3-11. Data Writes to a 32-Bit Bus

1. Bytes on the data bus marked with '•' are undefined.

Number of Bytes	Address LSB	Data to be Written				Address Bus	$\overline{BE}0-3$	Data Bus			
1	00	—	—	—	B0	A	H H H L	•	•	•	B0
1	01	—	—	B0	—	A	H H L H	•	•	B0	B0
1	10	—	B0	—	—	A	H L H H	•	B0	•	B0
1	11	B0	—	—	—	A	L H H H	B0	•	B0	B0
2	00	—	—	B1	B0	A	H H L L	•	•	B1	B0
2	01	—	B1	B0	—	A	H L L H	•	B1	B0	B0
2	10	B1	B0	—	—	A	L L H H	B1	B0	B1	B0
3	00	—	B2	B1	B0	A	H L L L	•	B2	B1	B0
3	01	B2	B1	B0	—	A	L L L H	B2	B1	B0	B0
4	00	B3	B2	B1	B0	A	L L L L	B3	B2	B1	B0

TABLE 3-12. Data Writes to a 16-Bit Bus

Number of Bytes	Address LSB	Data to be Written				Address Bus	$\overline{BE}0-3$	Data Bus			
1	00	—	—	—	B0	A	H H H L	•	•	•	B0
1	01	—	—	B0	—	A	H H L H	•	•	B0	B0
1	10	—	B0	—	—	A	H L H H	•	B0	•	B0
1	11	B0	—	—	—	A	L H H H	B0	•	B0	B0
2	00	—	—	B1	B0	A	H H L L	•	•	B1	B0
2	01	—	B1	B0	—	A A + 1	H L L H H L H H	• •	B1 •	B0 •	B0 B1
2	10	B1	B0	—	—	A	L L H H	B1	B0	B1	B0
3	00	—	B2	B1	B0	A A + 2	H L L L H L H H	• •	B2 •	B1 •	B0 B2
3	01	B2	B1	B0	—	A A + 1	L L L H L L H H	B2 •	B1 •	B0 B2	B0 B1
4	00	B3	B2	B1	B0	A A + 2	L L L L L L H H	B3 •	B2 •	B1 B3	B0 B2

3.0 Functional Description (Continued)

TABLE 3-13. Data Writes to an 8-Bit Bus

Number of Bytes	Address LSB	Data to be Written				Address Bus	$\overline{BE}0-3$	Data Bus			
1	00	—	—	—	B0	A	H H H L	•	•	•	B0
1	01	—	—	B0	—	A	H H L H	•	•	B0	B0
1	10	—	B0	—	—	A	H L H H	•	B0	•	B0
1	11	B0	—	—	—	A	L H H H	B0	•	B0	B0
2	00	—	—	B1	B0	A A + 1	H H L L H H L H	• •	• •	B1 •	B0 B1
2	01	—	B1	B0	—	A A + 1	H L L H H L H H	• •	B1 •	B0 •	B0 B1
2	10	B1	B0	—	—	A A + 1	L L H H L H H H	B1 •	B0 •	B1 •	B0 B1
3	00	—	B2	B1	B0	A A + 1 A + 2	H L L L H L L H H L H H	• • •	B2 • •	B1 • •	B0 B1 B2
3	01	B2	B1	B0	—	A A + 1 A + 2	L L L H L L H H L H H H	B2 • •	B1 • •	B0 • •	B0 B1 B2
4	00	B3	B2	B1	B0	A A + 1 A + 2 A + 3	L L L L L L L H L L H H L H H H	B3 • • •	B2 • • •	B1 • • •	B0 B1 B2 B3

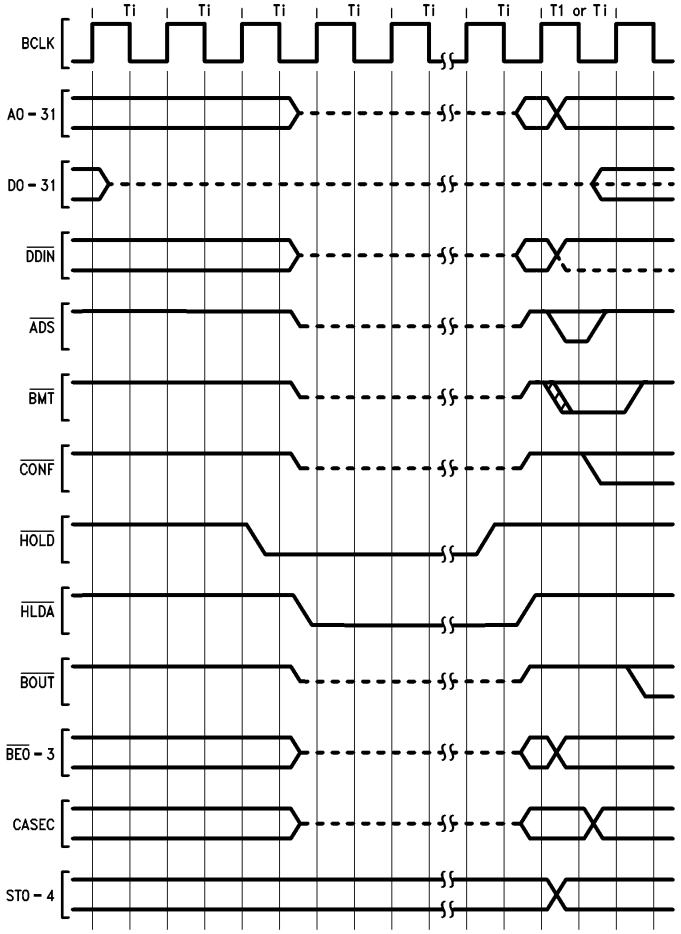


FIGURE 3-32. Hold Acknowledge. (Bus Initially Idle.)

Note: The status indicates 'IDLE' while the bus is granted. If the cause of the IDLE changes (e.g., CPU starts waiting for an interrupt), the status also changes.

The CPU will never grant the bus between interlocked read and write bus cycles.

Note: If an external device requires a very short latency to get control of the bus, the bus retry signal ($\overline{BRT}$) can be used instead of hold. See Section 3.5.5.

3.5.8 Interfacing Memory-Mapped I/O Devices

In Section 3.1.3.2 it was mentioned that some special precautions are needed when interfacing I/O devices to the NS32532 due to its internal pipelined implementation. Two special signals are provided for this purpose: $\overline{IOINH}$ and $\overline{IODEC}$. The CPU asserts $\overline{IOINH}$ during a read bus cycle to indicate that the bus cycle should be ignored if an I/O device is selected. The system responds by asserting $\overline{IODEC}$ to indicate to the CPU that an I/O device has been selected. $\overline{IODEC}$ is sampled by the CPU in the middle of state T2. If the cycle is extended, then the CPU uses the $\overline{IODEC}$ value sampled during the last wait state. If a bus error or a bus retry occurs, the sampled $\overline{IODEC}$ value is ignored. $\overline{IODEC}$ must be kept high during burst transfer cycles.

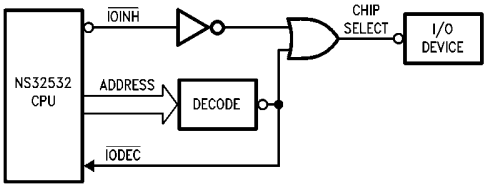
When $\overline{IODEC}$ is active during a bus cycle for which $\overline{IOINH}$ is asserted, the CPU discards the data and applies the special handling required for I/O devices. *Figure 3-33* shows a possible implementation of an I/O device interface where the address mapping of the I/O devices is fixed.

In an open system configuration, $\overline{IODEC}$ could be generated by the decoding logic of each I/O device subsystem.

When the on-chip MMU is enabled, the ClOUT signal could also be used for this purpose, since I/O devices are located in noncacheable areas. In this case however, a small performance degradation could result, due to the fact that the special I/O handling is also applied on references to noncacheable program and/or data areas.

Note 1: When $\overline{IODEC}$ is active in response to a read bus cycle, the CPU treats the reference as noncacheable.

Note 2: $\overline{IOINH}$ is kept inactive during write cycles.



TL/EE/9354–38

FIGURE 3-33. Typical I/O Device Interface

3.5.9 Interrupt and Debug Trap Requests

Three signals are provided by the CPU to externally request interrupts and/or a debug trap. $\overline{INT}$ and $\overline{NMI}$ are for maskable and non-maskable interrupts respectively. $\overline{DBG}$ is used for requesting an external debug trap.

The CPU samples $\overline{INT}$ and $\overline{NMI}$ on every other rising edge of BCLK, starting with the second rising edge of BCLK after $\overline{RST}$ goes high.

$\overline{NMI}$ is edge-sensitive; a high-to-low transition on it is detected by the CPU and stored in an internal latch, so that there is no need to keep it asserted until it is acknowledged.

$\overline{INT}$ is level-sensitive and, as such, once asserted, it must be kept asserted until it is acknowledged.

The $\overline{DBG}$ signal, like $\overline{NMI}$, is edge-sensitive; it differs from $\overline{NMI}$ in that the CPU samples it on each rising edge of BCLK. $\overline{DBG}$ can be asserted asynchronously to the CPU clock, but it should be at least 1.5 clock cycles wide in order to be recognized.

If $\overline{DBG}$ meets the specified setup and hold times, it will be recognized on the rising edge of BCLK deterministically.

Refer to *Figures 4-19* and *4-20* for more details on the timing of the above signals.

Note: If the $\overline{NMI}$ signal is pulsed to request a non-maskable interrupt, it may be necessary to keep it asserted for a minimum of two clock cycles to guarantee its detection, unless extra logic ensures that the pulse occurs around the BCLK sampling edge.

3.5.10 Cache Invalidation Requests

The contents of the on-chip Instruction and Data Caches can be invalidated by external requests from the system. It is possible to invalidate a single set or all sets in the Instruction Cache, Data Cache or both. The input signals $\overline{INVIC}$ and $\overline{INVDC}$ request invalidation of the Instruction Cache and Data Cache respectively. The input signal $\overline{INVSET}$ indicates whether the invalidation applies to a single set (16 bytes for the Instruction Cache and 32 bytes for the Data Cache) or to the entire cache. When only a single set is invalidated, the set number is specified on CIA0–CIA6.

$\overline{INVIC}$, $\overline{INVDC}$, $\overline{INVSET}$ and CIA0–CIA6 are all sampled synchronously by the CPU on the rising edge of BCLK. The CPU can respond to cache invalidation requests at a rate of one per BCLK cycle.

As shown in *Figures 3-16* and *3-17*, the validity bits of the on-chip caches are dual-ported. One port is used for accessing and updating the caches, while the other port is used independently for invalidation requests. Consequently, invalidation of the on-chip caches occurs with no interference to on-going cache accesses or bus cycles.

A cache invalidation request can occur during a read bus cycle for a location affected by the invalidation. In such a case, the data will be invalid in the cache if the invalidation request occurs after the T2- or T2B-state of the bus cycle.

Note: In the case of the Data Cache, the cache location will also be invalidated if the invalidation occurs during T2 or T2B of the read cycle.

Refer to *Figure 4-18* in Section 4 for timing details.

3.5.11 Internal Status

The NS32532 provides information on the system interface concerning its internal activity.

The U/$\overline{S}$ signal indicates the Address Space for a memory reference (See Section 2.4.2).

Note that U/$\overline{S}$ does not necessarily reflect the value of the U bit in the PSR register. For example, U/$\overline{S}$ is high during the memory access used to store the destination operand of a MOVSU instruction.

The $\overline{PFS}$ signal is asserted for one BCLK cycle when the CPU begins executing a new instruction. The $\overline{ISF}$ signal is driven High along with $\overline{PFS}$ if the new instruction does not follow the previous instruction in sequence. More specifically, $\overline{ISF}$ is High along with $\overline{PFS}$ after processing an exception or after executing one of the following instructions: ACB (branch taken), Bcond (branch taken), BR, BSR, CASE, CXP, CXPD, DIA, JSR, JUMP, RET, RETT, RETI, and RXP.

The $\overline{BP}$ signal is asserted for one BCLK cycle when an address-compare or PC-match condition is detected. If the $\overline{BP}$ signal is asserted one BCLK cycle after $\overline{PFS}$, it indicates that an address-compare debug condition has been detected. If $\overline{BP}$ is asserted at any other time, it indicates that a PC-Match debug condition has been detected.

While executing an LMR or CINV instruction, the CPU displays the operation code and source operand using slave processor write bus cycles. This information can be used to monitor the contents of the on-chip TLB, Instruction Cache and Data Cache.

During idle bus cycles, the signals ST0–ST4 indicate whether the CPU is waiting for an interrupt, waiting for a Slave Processor to complete executing an instruction or halted.

4.0 Device Specifications

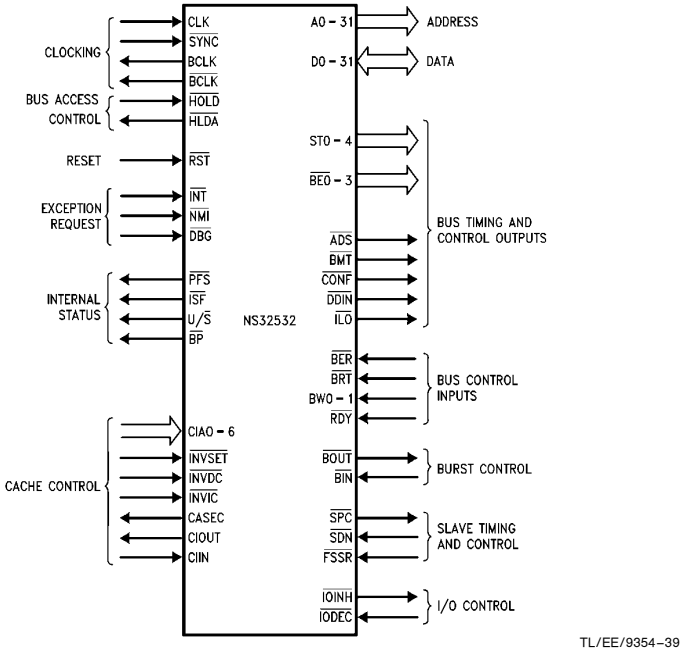


FIGURE 4-1. NS32532 Interface Signals

4.1 NS32532 PIN DESCRIPTIONS

Descriptions of the NS32532 pins are given in the following sections.

Included are also references to portions of the functional description, Section 3.

Figure 4-1 shows the NS32532 interface signals grouped according to related functions.

Note: An asterisk next to the signal name indicates a TRI-STATE condition for that signal when $\overline{\text{HOLD}}$ is acknowledged or during an extended retry.

4.1.1 Supplies

VCCL1–6 **Logic Power.**

+5V positive supplies for on-chip logic.

VCCB1–14 **Buffers Power.**

+5V positive supplies for on-chip output buffers.

VCCCLK **Bus Clock Power.**

+5V positive supply for on-chip clock drivers.

GNDL1–6 **Logic Ground.**

Ground references for on-chip logic.

GNDB1–13 **Buffers Ground.**

Ground references for on-chip output buffers.

GNDCLK **Bus Clock Ground.**

Ground reference for on-chip clock drivers.

4.1.2 Input Signals

CLK **Clock.**

Input Clock used to derive all CPU Timing.

$\overline{\text{SYNC}}$ **Synchronize.**

When $\overline{\text{SYNC}}$ is active, BCLK will stop toggling. This signal can be used to synchronize two or more CPUs (Section 3.5.2).

$\overline{\text{HOLD}}$ **Hold Request.**

When active, causes the CPU to release the bus for DMA or multiprocessing purposes (Section 3.5.7).

Note:

If the $\overline{\text{HOLD}}$ signal is generated asynchronously, its set up and hold times may be violated. In this case it is recommended to synchronize it with the falling edge of BCLK to minimize the possibility of metastable states.

The CPU provides only one synchronization stage to minimize the $\overline{\text{HLDA}}$ latency. This is to avoid speed degradations in cases of heavy $\overline{\text{HOLD}}$ activity (i.e. DMA controller cycles interleaved with CPU cycles).

$\overline{\text{RST}}$ **Reset.**

When $\overline{\text{RST}}$ is active, the CPU is initialized to a known state (Section 3.5.3).

$\overline{\text{INT}}$ **Interrupt.**

A low level on this signal requests a maskable interrupt (Section 3.5.9).

$\overline{\text{NMI}}$ **Nonmaskable Interrupt.**

A High-to-Low transition of this signal requests a nonmaskable interrupt (Section 3.5.9).

$\overline{DBG}$ **Debug Trap Request.**

A High-to-Low transition of this signal requests a debug trap (Section 3.5.9).

CIA0–6 **Cache Invalidation Address Bus.**

Bits 0 through 4 specify the set address to invalidate in the on-chip caches. CIA0 is the least significant. Bits 5 and 6 are reserved (Section 3.5.10).

$\overline{INVSET}$ **Invalidate Set.**

When Low, only a set in the on-chip cache(s) is invalidated; when High, the entire cache(s) is (are) invalidated.

$\overline{INVDC}$ **Invalidate Data Cache.**

When Low, the Data Cache contents are invalidated. $\overline{INVSET}$ determines whether a single set or the entire Data Cache is invalidated.

$\overline{INVIC}$ **Invalidate Instruction Cache.**

When Low, the Instruction Cache contents are invalidated. $\overline{INVSET}$ determines whether a single set or the entire Instruction Cache is invalidated.

CIIN **Cache Inhibit In.**

When active, indicates that the location referenced in the current bus cycle is not cacheable. CIIN must not change within an aligned 16-byte block.

$\overline{IODEC}$ **I/O Decode.**

Indicates to the CPU that a peripheral device is addressed by the current bus cycle. The value of $\overline{IODEC}$ must not change within an aligned 16-byte block (Section 3.5.8).

$\overline{FSSR}$ **Force Slave Status Read.**

When asserted, indicates that the slave status word should be read by the CPU (Section 3.1.4.1). An external 10 kΩ resistor should be connected between $\overline{FSSR}$ and V_{CC}.

$\overline{SDN}$ **Slave Done.**

Used by a slave processor to signal the completion of a slave instruction (Section 3.1.4.1). An external 10 kΩ resistor should be connected between $\overline{SDN}$ and V_{CC}.

$\overline{BIN}$ **Burst In.**

When active, indicates to the CPU that the memory supports burst cycles (Section 3.5.4.3).

$\overline{RDY}$ **Ready.**

While this signal is not active, the CPU extends the current bus cycle to support a slow memory or peripheral device.

BW0–1 **Bus Width.**

These lines define the bus width (8, 16 or 32 bits) for each data transfer; BW0 is the least significant bit. The bus width must not change within an aligned 16-byte block—encodings are:

00—Reserved

01—8 Bits

10—16 Bits

11—32 Bits

$\overline{BRT}$ **Bus Retry.**

When active, the CPU will reexecute the last bus cycle (Section 3.5.5).

$\overline{BER}$ **Bus Error.**

When active, indicates that an error occurred during a bus cycle. It is treated by the CPU as the highest priority exception after reset.

4.1.3 Output Signals

BCLK **Bus Clock.**

Output clock for bus timing (Section 3.5.2).

$\overline{BCLK}$ **Bus Clock Inverse.**

Inverted output clock.

$\overline{HLDA}$ **Hold Acknowledge.**

Activated by the CPU in response to the $\overline{HOLD}$ input to indicate that the CPU has released the bus.

$\overline{PFS}$ **Program Flow Status.**

A pulse on this signal indicates the beginning of execution for each instruction (Section 3.5.11).

$\overline{ISF}$ **Internal Sequential Fetch.**

Indicates along with $\overline{PFS}$ that the instruction beginning execution is sequential ($\overline{ISF}$ Low) or non-sequential ($\overline{ISF}$ High).

U/$\overline{S}$ **User/Supervisor.**

User or supervisor mode status.

$\overline{BP}$ **Break Point.**

This signal is activated when the CPU detects a PC or operand-address match debug condition (Section 3.3.2).

CASEC ***Cache Section.**

For cacheable data read bus cycles indicates the Section of the on-chip Data Cache where the data will be placed; undefined for other bus cycles. This signal can be used for external monitoring of the data cache contents.

CIOUT **Cache Inhibit Out.**

This signal reflects the state of the CI bit in the second level page table entry (PTE). It is used to specify non-cacheable pages. It is held low while address translation is disabled and for MMU references to page table entries.

$\overline{IOINH}$ **I/O Inhibit.**

Indicates that the current bus cycle should be ignored if a peripheral device is addressed.

$\overline{SPC}$ **Slave Processor Control.**

Data strobe for slave processor transfers.

$\overline{BOUT}$ ***Burst Out.**

When active, indicates that the CPU is requesting to perform burst cycles.

$\overline{ILO}$ **Interlocked Operation.**

When active, indicates that interlocked cycles are being performed (Section 3.5.4.5).

4.0 Device Specifications (Continued)

$\overline{DDIN}$ *__Data Direction.__

Indicates the direction of a data transfer. It is low for reads and high for writes.

$\overline{CONF}$ *__Confirm Bus Cycle.__

When active, indicates that a bus cycle initiated by $\overline{ADS}$ is valid; that is, the bus cycle has not been cancelled (Section 3.5.4.2).

$\overline{BMT}$ *__Begin Memory Transaction.__

When Stable Low indicates that the current bus cycle is valid; that is, the bus cycle has not been cancelled (Section 3.5.4.2).

$\overline{ADS}$ *__Address Strobe.__

When active, indicates that a bus cycle has begun and a valid address is on the address bus.

$\overline{BE}$0–3 *__Byte Enables.__

Used to selectively enable data transfers on bytes 0–3 of the data bus.

ST0–4 **Status.**

Bus cycle status code; ST0 is the least significant. Encodings are:

00000—Idle: CPU Inactive on Bus.

00001—Idle: WAIT Instruction.

00010—Idle: Halted.

00011—Idle: The bus is idle while the slave processor is executing an instruction.

00100—Interrupt Acknowledge, Master.

00101—Interrupt Acknowledge, Cascaded.

00110—End of Interrupt, Master.

00111—End of Interrupt, Cascaded.

01000—Sequential Instruction Fetch.

01001—Non-Sequential Instruction Fetch.

01010—Data Transfer.

01011—Read Read-Modify-Write Operand.

01100—Read for Effective Address.

01101—Access PTE1 by MMU.

01110—Access PTE2 by MMU.

01111 • • • 11100 } Reserved.

11101—Transfer Slave Operand.

11110—Read Slave Status Word.

11111—Broadcast Slave ID.

A0–31 *__Address Bus.__

Used by the CPU to output a 32-bit address at the beginning of a bus cycle. A0 is the least significant.

4.1.4 Input/Output Signals

D0–31 *__Data Bus.__

Used by the CPU to input or output data during a read or write cycle respectively.

4.2 ABSOLUTE MAXIMUM RATINGS

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Case Temperature Under Bias	0°C to +95°C
Storage Temperature	−65°C to +150°C
All Input or Output Voltages with Respect to GND	−0.5V to +7V
Power Dissipation	4 W

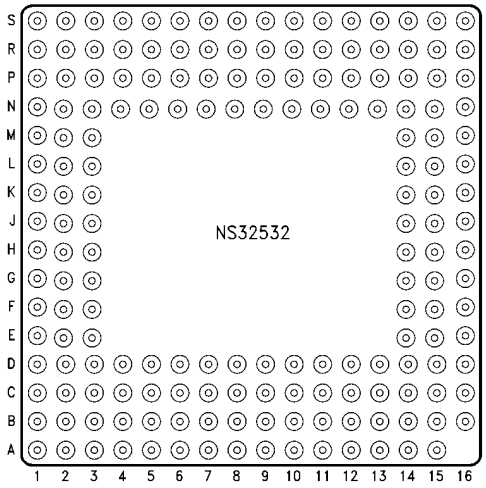
Note: *Absolute maximum ratings indicate limits beyond which permanent damage may occur. Continuous operation at these limits is not intended; operation should be limited to those conditions specified under Electrical Characteristics.*

4.3 ELECTRICAL CHARACTERISTICS T_{CASE} = 0° to +95°C, V_{CC} = 5V ±5%, GND = 0V

Symbol	Parameter	Conditions	Min	Typ	Max	Units
V_{IH}	High Level Input Voltage		2.0		V_{CC} + 0.5	V
V_{IL}	Low Level Input Voltage		−0.5		0.8	V
V_{OH}	High Level Output Voltage	I_{OH} = −400 μA	2.4			V
V_{OL}	Low Level Output Voltage					
	A0–11, D0–31, $\overline{DDIN}$	I_{OL} = 4 mA			0.4	V
	$\overline{CONF}$, $\overline{BMT}$	I_{OL} = 6 mA			0.4	V
	BCLK, $\overline{BCLK}$	I_{OL} = 16 mA			0.4	V
	All Other Outputs	I_{OL} = 2 mA			0.4	V
I_L	Input Load Current	$0 \le V_{IN} \le V_{CC}$	−20		20	μA
I_L	Leakage Current (Output and I/O pins in TRI-STATE/Input Mode)	$0.4 \le V_{IN} \le V_{CC}$	−20		20	μA
C_{IN}	CLK Input Capacitance			15		pF
I_{CC}	Active Supply Current	I_{OUT} = 0, T_A = 25°C, V_{CC} = 5V		650 @ 30 MHz 550 @ 25 MHz 450 @ 20 MHz	800 @ 30 MHz 675 @ 25 MHz 575 @ 20 MHz	mA

4.0 Device Specifications (Continued)

Connection Diagram



TL/EE/9354–40

Bottom View
Order Number NS32532-20, NS32532-25 or NS32532-30

FIGURE 4-2. 175-Pin PGA Package

NS32532 Pinout Descriptions

Desc	Pin	Desc	Pin	Desc	Pin
Reserved	A1	D26	B16	GNDB13	D14
Reserved	A2	Reserved	C1	VCCB14	D15
Reserved	A3	Reserved	C2	D23	D16
$\overline{BP}$	A4	VCCL2	C3	$\overline{IOINH}$	E1
$\overline{ISF}$	A5	Reserved	C4	$\overline{ILO}$	E2
$\overline{RST}$	A6	$\overline{PFS}$	C5	GNDB3	E3
$\overline{NMI}$	A7	$\overline{SDN}$	C6	D24	E14
GNDB1	A8	Reserved	C7	D22	E15
Reserved	A9	$\overline{BCLK}$	C8	D20	E16
VCCB2	A10	VCCCLK	C9	A30	F1
$\overline{INVIC}$	A11	$\overline{SYNC}$	C10	$\overline{CASEC}$	F2
Reserved (1)	A12	CIA0	C11	Reserved	F3
CIA1	A13	CIA6	C12	D21	F14
CIA4	A14	VCCL6	C13	D19	F15
VCCB1	A15	D29	C14	D18	F16
Reserved	B1	D27	C15	A29	G1
VCCB4	B2	D25	C16	A31	G2
Reserved	B3	U/$\overline{S}$	D1	VCCB5	G3
Reserved	B4	Reserved	D2	GNDB12	G14
VCCB3	B5	Reserved	D3	D17	G15
$\overline{FSSR}$	B6	GNDL3	D4	D16	G16
$\overline{INT}$	B7	GNDB2	D5	A27	H1
VCCL1	B8	$\overline{DBG}$	D6	A28	H2
GNDL2	B9	Reserved	D7	GNDB4	H3
$\overline{INVSET}$	B10	BCLK	D8	VCCB13	H14
$\overline{INVDC}$	B11	GNDCLK	D9	D15	H15
CIA3	B12	CLK	D10	D14	H16
CIA5	B13	CIA2	D11	A26	J1
D30	B14	D31	D12	A25	J2
D28	B15	GNDL1	D13	A24	J3

Desc	Pin	Desc	Pin	Desc	Pin
GNDL6	J14	GNDL5	N9	A0	R6
VCCL5	J15	$\overline{CONF}$	N10	VCCB9	R7
D13	J16	$\overline{RDY}$	N11	CIOUT	R8
VCCB6	K1	$\overline{HOLD}$	N12	$\overline{SPC}$	R9
A23	K2	VCCB11	N13	$\overline{BE3}$	R10
GNDL4	K3	GNDB10	N14	VCCB10	R11
GNDB11	K14	D4	N15	$\overline{ADS}$	R12
D11	K15	D6	N16	BW1	R13
D12	K16	A16	P1	$\overline{BER}$	R14
A22	L1	VCCB7	P2	CIIN	R15
A21	L2	GNDB6	P3	D2	R16
VCCL3	L3	A10	P4	A13	S1
D8	L14	A6	P5	A8	S2
D9	L15	A2	P6	A5	S3
D10	L16	ST3	P7	A3	S4
A20	M1	GNDB8	P8	A1	S5
GNDB5	M2	VCCL4	P9	ST2	S6
A17	M3	$\overline{BE1}$	P10	ST1	S7
D5	M14	GNDB9	P11	ST0	S8
D7	M15	BW0	P12	$\overline{BOUT}$	S9
VCCB12	M16	$\overline{BIN}$	P13	$\overline{DDIN}$	S10
A19	N1	Reserved	P14	$\overline{BE2}$	S11
A18	N2	D0	P15	$\overline{BE0}$	S12
A14	N3	D3	P16	$\overline{BMT}$	S13
A11	N4	A15	R1	$\overline{BRT}$	S14
VCCB8	N5	A12	R2	$\overline{IODEC}$	S15
GNDB7	N6	A9	R3	D1	S16
ST4	N7	A7	R4		
$\overline{HLDA}$	N8	A4	R5		

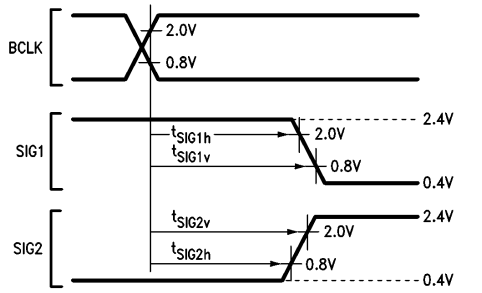
Note 1: This pin should be grounded.
All other reserved pins should be left open.

4.0 Device Specifications (Continued)

4.4 SWITCHING CHARACTERISTICS

4.4.1 Definitions

All the timing specifications given in this section refer to 0.8V or 2.0V on all the signals as illustrated in *Figures 4-3* and *4-4*, unless specifically stated otherwise.



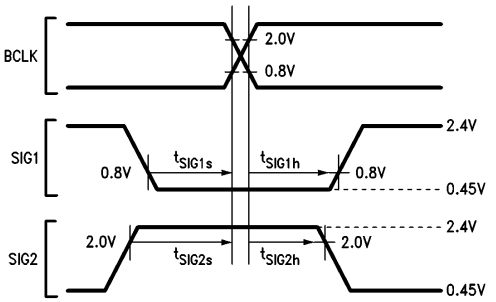
TL/EE/9354–41

FIGURE 4-3. Output Signals Specification Standard

ABBREVIATIONS:

L.E.—leading edge R.E.—rising edge

T.E.—training edge F.E.—falling edge



TL/EE/9354–42

FIGURE 4-4. Input Signals Specification Standard

4.0 Device Specifications (Continued)

4.4.2 Timing Tables

4.4.2.1 Output Signals: Internal Propagation Delays, NS32532-20, NS32532-25, NS32532-30

- Maximum times assume capacitive loading of 100 pF on the clock signals and 50 pF on all the other signals. A minimum capacitance load of 50 pF on BCLK and $\overline{BCLK}$ is also assumed.

Name	Figure	Description	Reference/Conditions	NS32532-20		NS32532-25		NS32532-30		Units
				Min	Max	Min	Max	Min	Max	
t_{BC_p}	4–24	Bus Clock Period	R.E., BCLK to Next R.E., BCLK	50	100	40	100	33.3	100	ns
t_{BC_h}	4–24	BCLK High Time	At 2.0V on BCLK (Both Edges)	0.5 t_{BCp} −5		0.5 t_{BCp} −4		0.5 t_{BCp} −3.65		ns
t_{BC_l}	4–24	BCLK Low Time	At 0.8V on BCLK (Both Edges)	0.5 t_{BCp} −5		0.5 t_{BCp} −4		0.5 t_{BCp} −3.65		ns
t_{BC_r}(1)	4–24	BCLK Rise Time	0.8V to 2.0V on R.E., BCLK		5		4		3	ns
t_{BC_f}(1)	4–24	BCLK Fall Time	2.0V to 0.8V on F.E., BCLK		5		4		3	ns
t_{NBC_h}	4–24	$\overline{BCLK}$ High Time	At 2.0V on $\overline{BCLK}$ (Both Edges)	0.5 t_{BCp} −5		0.5 t_{BCp} −4		0.5 t_{BCp} −3.65		ns
t_{NBC_l}	4–24	$\overline{BCLK}$ Low Time	At 0.8V on $\overline{BCLK}$ (Both Edges)	0.5 t_{BCp} −5		0.5 t_{BCp} −4		0.5 t_{BCp} −3.65		ns
t_{NBC_r}(1)	4–24	$\overline{BCLK}$ Rise Time	0.8V to 2.0V on R.E., $\overline{BCLK}$		5		4		3	ns
t_{NBC_f}(1)	4–24	$\overline{BCLK}$ Fall Time	2.0V to 0.8V on F.E., $\overline{BCLK}$		5		4		3	ns
$t_{CBC_{dr}}$	4–24	CLK to BCLK R.E. Delay	2.0V on R.E., CLK to 2.0V on R.E., BCLK		20		17		15	ns
$t_{CBC_{df}}$	4–24	CLK to BCLK F.E. Delay	2.0V on R.E., CLK to 0.8V on F.E., BCLK		20		17		15	ns
$t_{CNBC_{dr}}$	4–24	CLK to $\overline{BCLK}$ R.E. Delay	2.0V on R.E., CLK to 0.8V on R.E., $\overline{BCLK}$		20		17		15	ns
$t_{CNBC_{df}}$	4–24	CLK to $\overline{BCLK}$ F.E. Delay	2.0V on R.E., CLK to 0.8V on F.E., $\overline{BCLK}$		20		17		15	ns
$t_{BCNBC_{rf}}$	4–24	Bus Clocks Skew	2.0V on R.E., BCLK to 0.8V on F.E., $\overline{BCLK}$	−2	+2	−2	+2	−1	+1	ns
$t_{BCNBC_{fr}}$	4–24	Bus Clocks Skew	0.8V on F.E., BCLK to 2.0V on R.E., $\overline{BCLK}$	−2	+2	−2	+2	−1	+1	ns
t_{A_v}	4–5, 4–6	Address Bits 0–31 Valid	After R.E., BCLK T1		11		9		8	ns
t_{A_h}	4–5, 4–6	Address Bits 0–31 Hold	After R.E., BCLK T1 or Ti	0		0		0		ns
t_{A_f}	4–11, 4–12	Address Bits 0–31 Floating	After F.E., BCLK Ti		21		17		13	ns
t_{An_f}	4–11, 4–12	Address Bits 0–31 Not Floating	After F.E., BCLK Ti	0		0		0		ns

Note 1: Guaranteed by characterization. Due to tester conditions this parameter is not 100% tested.

4.0 Device Specifications (Continued)

4.4.2.1 Output Signals: Internal Propagation Delays, NS32532-20, NS32532-25, NS32532-30 (Continued)

Name	Figure	Description	Reference/Conditions	NS32532-20		NS32532-25		NS32532-30		Units
				Min	Max	Min	Max	Min	Max	
t_{AB_v}	4–8	Address Bits A2, A3 Valid (Burst Cycle)	After R.E., BCLK T2B		11		9		8	ns
t_{AB_h}	4–8	Address Bits A2, A3 Hold (Burst Cycle)	After R.E., BCLK T2B	0		0		0		ns
t_{DO_v}	4–6, 4–15	Data Out Valid	After R.E., BCLK T1	0.5 t_{BCp}	0.5 t_{BCp} + 13 ns	0.5 t_{BCp}	0.5 t_{BCp} + 12 ns	0.5 t_{BCp}	0.5 t_{BCp} +11 ns	ns
t_{DO_h}	4–6, 4–15	Data Out Hold	After R.E., BCLK T1 or Ti	0		0		0		ns
$t_{DO_{spc}}$	4–15	Data Out Setup (Slave Write)	Before $\overline{SPC}$ T.E.	12		10		8		ns
t_{DO_f}	4–7	Data Bus Floating	After R.E., BCLK T1 or Ti		21		17		13	ns
$t_{DO_{nf}}$	4–7	Data Bus Not Floating	After F.E., BCLK T1	0		0		0		ns
t_{BMT_v}	4–5, 4–7	$\overline{BMT}$ Signal Valid	After R.E., BCLK T1		30		25		21	ns
t_{BMT_h}	4–5, 4–7	$\overline{BMT}$ Signal Hold	After R.E., BCLK T2	0		0		0		ns
t_{BMT_f}	4–11, 4–12	$\overline{BMT}$ Signal Floating	After F.E., BCLK Ti		21		17		13	ns
$t_{BMT_{hf}}$	4–11, 4–12	$\overline{BMT}$ Signal Not Floating	After F.E., BCLK Ti	0		0		0		ns
t_{CONF_a}	4–5, 4–8	$\overline{CONF}$ Signal Active	After R.E., BCLK T1	0.5 t_{BCp}	0.5 t_{BCp} + 11	0.5 t_{BCp}	0.5 t_{BCp} + 9	0.5 t_{BCp}	0.5 t_{BCp} + 8	ns
$t_{CONF_{ia}}$	4–5, 4–8	$\overline{CONF}$ Signal Inactive	After R.E., BCLK T1 or Ti		11		9		8	ns
t_{CONF_f}	4–11, 4–12	$\overline{CONF}$ Signal Floating	After F.E., BCLK Ti		21		17		13	ns
$t_{CONF_{nf}}$	4–11, 4–12	$\overline{CONF}$ Signal Not Floating	After F.E., BCLK Ti	0		0		0		ns
t_{ADS_a}	4–5, 4–8	$\overline{ADS}$ Signal Active	After R.E., BCLK T1		11		9		8	ns
$t_{ADS_{ia}}$	4–5, 4–8	$\overline{ADS}$ Signal Inactive	After F.E., BCLK T1		11		9		8	ns
t_{ADS_w}	4–6	$\overline{ADS}$ Pulse Width	At 0.8V (Both Edges)	15		12		10		ns
t_{ADS_f}	4–11, 4–12	$\overline{ADS}$ Signal Floating	After F.E., BCLK Ti		21		17		13	ns
$t_{ADS_{nf}}$	4–11, 4–12	$\overline{ADS}$ Signal Not Floating	After F.E., BCLK Ti	0		0		0		ns
t_{BE_v}	4–6, 4–8	$\overline{BE}_n$ Signals Valid	After R.E., BCLK T1		11		9		8	ns
t_{BE_h}	4–6, 4–8	$\overline{BE}_n$ Signals Hold	After R.E., BCLK T1, Ti or T2B	0		0		0		ns
t_{BE_f}	4–11, 4-12	$\overline{BE}_n$ Signals Floating	After F.E., BCLK Ti		21		17		13	ns
$t_{BE_{nf}}$	4–11, 4–12	$\overline{BE}_n$ Signals Not Floating	After F.E., BCLK Ti	0		0		0		ns
t_{DDIN_v}	4–5, 4–6	$\overline{DDIN}$ Signal Valid	After R.E., BCLK T1		11		9		8	ns
t_{DDIN_h}	4–5, 4–6	$\overline{DDIN}$ Signal Hold	After R.E., BCLK T1 or Ti	0		0		0		ns
t_{DDIN_f}	4–11, 4–12	$\overline{DDIN}$ Signal Floating	After F.E., BCLK Ti		21		17		13	ns
$t_{DDIN_{nf}}$	4–11, 4–12	$\overline{DDIN}$ Signal Not Floating	After F.E., BCLK Ti	0		0		0		ns
t_{SPC_a}	4–14, 4–15	$\overline{SPC}$ Signal Active	After R.E., BCLK T1		19		15		12	ns

Name	Figure	Description	Reference/Conditions	NS32532-20		NS32532-25		NS32532-30		Units
				Min	Max	Min	Max	Min	Max	
$t_{SPC_{ia}}$	4–14, 4–15	$\overline{SPC}$ Signal Inactive	After R.E., BCLK Ti, T1 or T2		19		15		12	ns
t_{DDSPC}(1)	4–14	$\overline{DDIN}$ Valid to $\overline{SPC}$ Active	Before $\overline{SPC}$ L.E.	0		0		0		ns
t_{HLDA_a}	4–12, 4–13	$\overline{HLDA}$ Signal Active	After F.E., BCLK Ti		15		11		10	ns
$t_{HLD_{Aia}}$	4–12	$\overline{HLDA}$ Signal Inactive	After F.E., BCLK Ti		15		11		10	ns
t_{ST_v}	4–5, 4–14	Status (ST0–4) Valid	After R.E., BCLK T1		11		9		8	ns
t_{ST_h}	4–5, 4–14	Status (ST0–4) Hold	After R.E., BCLK T1 or Ti	0		0		0		ns
t_{BOUT_a}	4–8, 4–9	$\overline{BOUT}$ Signal Active	After R.E., BCLK T2		15		12		11	ns
$t_{BOUT_{ia}}$	4–8, 4–9	$\overline{BOUT}$ Signal Inactive	After R.E., BCLK Last T2B, T1 or Ti		15		12		11	ns
t_{BOUT_f}	4–11, 4–12	$\overline{BOUT}$ Signal Floating	After F.E., BCLK Ti		21		17		13	ns
$t_{BOUT_{nf}}$	4–11, 4–12	$\overline{BOUT}$ Signal Not Floating	After F.E., BCLK Ti	0		0		0		ns
t_{ILO_a}	4–7	Interlock Signal Active	After F.E., BCLK Ti		11		9		8	ns
$t_{ILO_{ia}}$	4–7	Interlock Signal Inactive	After F.E., BCLK Ti		11		9		8	ns
t_{PFS_a}	4–21	$\overline{PFS}$ Signal Active	After F.E., BCLK		15		11		10	ns
$t_{PFS_{ia}}$	4–21	$\overline{PFS}$ Signal Inactive	After F.E., Next BCLK		15		11		10	ns
t_{ISF_a}	4–22	$\overline{ISF}$ Signal Active	After F.E., BCLK		15		11		10	ns
$t_{ISF_{ia}}$	4–22	$\overline{ISF}$ Signal Inactive	After F.E., Next BCLK		15		11		10	ns
t_{BP_a}	4–23	$\overline{BP}$ Signal Active	After F.E., BCLK		15		11		10	ns
$t_{BP_{ia}}$	4–23	$\overline{BP}$ Signal Inactive	After F.E., Next BCLK		15		11		10	ns
t_{US_v}	4–5	U/$\overline{S}$ Signal Valid	After R.E., BCLK T1		11		9		8	ns
t_{US_h}	4–5	U/$\overline{S}$ Signal Hold	After R.E., BCLK T1 or Ti	0		0		0		ns
t_{CAS_v}	4–5	$\overline{CASEC}$ Signal Valid	After F.E., BCLK T1		15		11		10	ns
t_{CAS_h}	4–5	$\overline{CASEC}$ Signal Hold	After R.E., BCLK T1 or Ti	0		0		0		ns
t_{CAS_f}	4–11, 4–12	$\overline{CASEC}$ Signal Floating	After F.E., BCLK Ti		21		17		13	ns
$t_{CAS_{nf}}$	4–11, 4–12	$\overline{CASEC}$ Signal Not Floating	After F.E., BCLK Ti	0		0		0		ns
t_{CIO_v}	4–5	CIOUT Signal Valid	After R.E., BCLK T1		15		11		10	ns
t_{CIO_h}	4–5	CIOUT Signal Hold	After R.E., BCLK T1 or Ti	0		0		0		ns
t_{IOI_v}	4–5	$\overline{IOINH}$ Signal Valid	After R.E., BCLK T1		15		11		10	ns
t_{IOI_h}	4–5	$\overline{IOINH}$ Signal Hold	After R.E., BCLK T1 or Ti	0		0		0		ns

4.0 Device Specifications (Continued)

4.4.2.2 Input Signal Requirements: NS32532-20, NS32532-25, NS32532-30

Name	Figure	Description	Reference/Conditions	NS32532-20		NS32532-25		NS32532-30		Units
				Min	Max	Min	Max	Min	Max	
t_{Cp}	4–24	Input Clock Period	R.E., CLK to Next R.E., CLK	25	50	20	50	16.6	50	ns
t_{Ch}	4–24	CLK High Time	At 2.0V on CLK (Both Edges)	0.5 t_{Cp} −5		0.5 t_{Cp} −5		0.5 t_{Cp} −4		ns
t_{Cl}	4–24	CLK Low Time	At 0.8V on CLK (Both Edges)	0.5 t_{Cp} −5		0.5 t_{Cp} −5		0.5 t_{Cp} −4		ns
t_{Cr} (1)	4–24	CLK Rise Time	0.8V to 2.0V on R.E., CLK		5		4		3	ns
t_{Cf} (1)	4–24	CLK Fall Time	2.0V to 0.8V on F.E., CLK		5		4		3	ns
t_{DIs}	4–5, 4–14	Data In Setup	Before R.E., BCLK T1 or Ti	12		10		8		ns
t_{DIh}	4–5, 4–14	Data In Hold	After R.E., BCLK T1 or Ti	1		1		1		ns
t_{RDYs}	4–5	$\overline{RDY}$ Setup Time	Before R.E., BCLK T2(W), T1 or Ti	19		15		12		ns
t_{RDYh}	4–5	$\overline{RDY}$ Hold Time	Ater R.E., BCLK T2(W), T1 or Ti	1		1		1		ns
t_{BWs}	4–5	BW0–1 Setup Time	Before F.E., BCLK T2 or T2(W)	19		15		12		ns
t_{BWh}	4–5	BW0–1 Hold Time	After F.E., BCLK T2 or T2(W)	1		1		1		ns
t_{HOLDs}	4–12, 4–13	$\overline{HOLD}$ Setup Time	Before F.E., BCLK	19		15		12		ns
t_{HOLDh}	4–12	$\overline{HOLD}$ Hold Time	After F.E., BCLK	1		1		1		ns
t_{BINs}	4–8	$\overline{BIN}$ Setup Time	Before F.E., BCLK T2 or T2(W)	18		14		11		ns
t_{BINh}	4–8	$\overline{BIN}$ Hold Time	After F.E., BCLK T2 or T2(W)	1		1		1		ns
t_{BERs}	4–6, 4–8	$\overline{BER}$ Setup Time	Before R.E., BCLK T1 or Ti	19		15		12		ns
t_{BERh}	4–6, 4–8	$\overline{BER}$ Hold Time	After R.E., BCLK T1 or Ti	1		1		1		ns
t_{BRTs}	4–6, 4–8	$\overline{BRT}$ Setup Time	Before R.E., BCLK T1 or Ti	19		15		12		ns
t_{BRTh}	4–6, 4–8	$\overline{BRT}$ Hold Time	After R.E., BCLK T1 or Ti	1		1		1		ns
t_{IODs}	4–5	$\overline{IODEC}$ Setup Time	Before F.E., BCLK T2 or T2(W)	18		14		11		ns
t_{IODh}	4–5	$\overline{IODEC}$ Hold Time	After F.E., BCLK T2 or T2(W)	1		1		1		ns
t_{PWR} (1)	4–26	Power Stable to R.E. of $\overline{RST}$	After VCC Reaches 4.5V	50		40		30		μs
t_{RSTs}	4–27	$\overline{RST}$ Setup Time	Before R.E., BCLK	14		12		11		ns
t_{RSTw}	4–27	$\overline{RST}$ Pulse Width	At 0.8V (Both Edges)	64		64		64		t_{BCp}

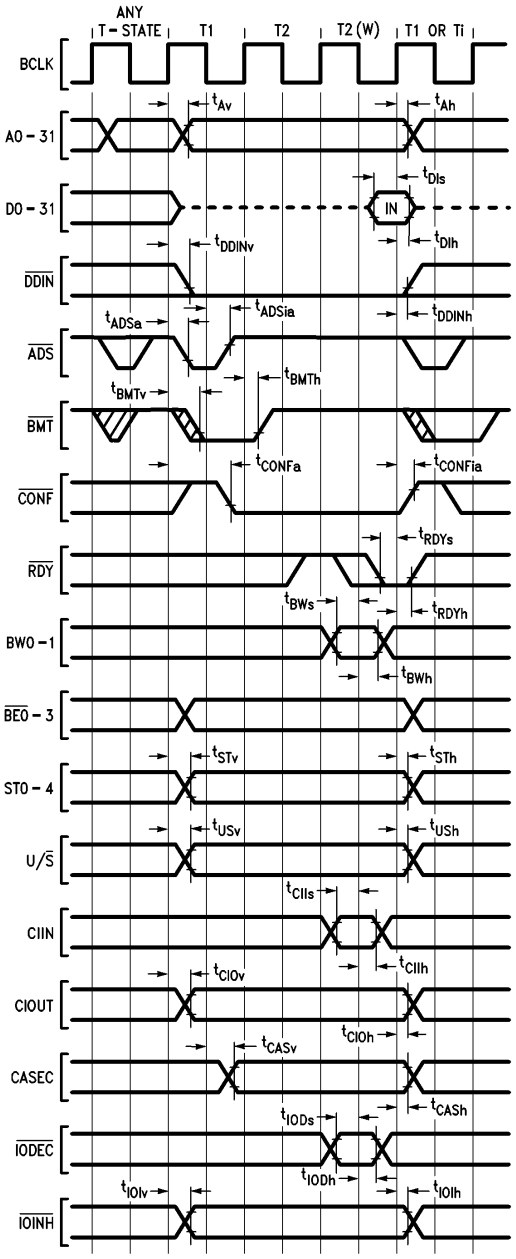
Note 1: Guaranteed by characterization. Due to tester conditions this parameter is not 100% tested.

4.4.2.2 Input Signal Requirements: NS32532-20, NS32532-25, NS32532-30 (Continued)

Name	Figure	Description	Reference/Conditions	NS32532-20		NS32532-25		NS32532-30		Units
				Min	Max	Min	Max	Min	Max	
t_{CII_s}	4–5	$\overline{CIIN}$ Setup Time	Before F.E., BCLK T2	19		15		12		ns
t_{CII_h}	4–5	$\overline{CIIN}$ Hold Time	After F.E., BCLK T2	1		1		1		ns
t_{INT_s}	4–19	$\overline{INT}$ Setup Time	Before R.E., BCLK	12		10		9		ns
t_{INT_h}	4–19	$\overline{INT}$ Hold Time	After R.E., BCLK	1		1		1		ns
t_{NMI_s}	4–19	$\overline{NMI}$ Setup Time	Before R.E., BCLK	18		15		14		ns
t_{NMI_h}	4–19	$\overline{NMI}$ Hold Time	After R.E., BCLK	1		1		1		ns
t_{SD_s}	4–16	$\overline{SDN}$ Setup Time	Before R.E., BCLK	12		10		9		ns
t_{SD_h}	4–16	$\overline{SDN}$ Hold Time	After R.E., BCLK	1		1		1		ns
t_{FSSR_s}	4–17	$\overline{FSSR}$ Setup Time	Before R.E., BCLK	12		10		9		ns
t_{FSSR_h}	4–17	$\overline{FSSR}$ Hold Time	After R.E., BCLK	1		1		1		ns
t_{SYNC_s}	4–25	$\overline{SYNC}$ Setup Time	Before R.E., CLK	10		8		7		ns
t_{SYNC_h}	4–25	$\overline{SYNC}$ Hold Time	After R.E., CLK	1		1		1		ns
t_{CIA_s}	4–18	CIA0–6 Setup Time	Before R.E., BCLK	12		10		9		ns
t_{CIA_h}	4–18	CIA0–6 Hold Time	After R.E., BCLK	1		1		1		ns
t_{INVS_s}	4–18	$\overline{INVSET}$ Setup Time	Before R.E., BCLK	12		11		9		ns
t_{INVS_h}	4–18	$\overline{INVSET}$ Hold Time	After R.E., BCLK	1		1		1		ns
t_{INVI_s}	4–18	$\overline{INVIC}$ Setup Time	Before R.E., BCLK	12		10		9		ns
t_{INVI_h}	4–18	$\overline{INVIC}$ Hold Time	After R.E., BCLK	1		1		1		ns
t_{INVD_s}	4–18	$\overline{INVDC}$ Setup Time	Before R.E., BCLK	12		10		9		ns
t_{INVD_h}	4–18	$\overline{INVDC}$ Hold Time	After R.E., BCLK	1		1		1		ns
t_{DBG_s}	4–20	$\overline{DBG}$ Setup Time	Before R.E., BCLK	12		10		9		ns
t_{DBG_h}	4–20	$\overline{DBG}$ Hold Time	After R.E., BCLK	1		1		1		ns

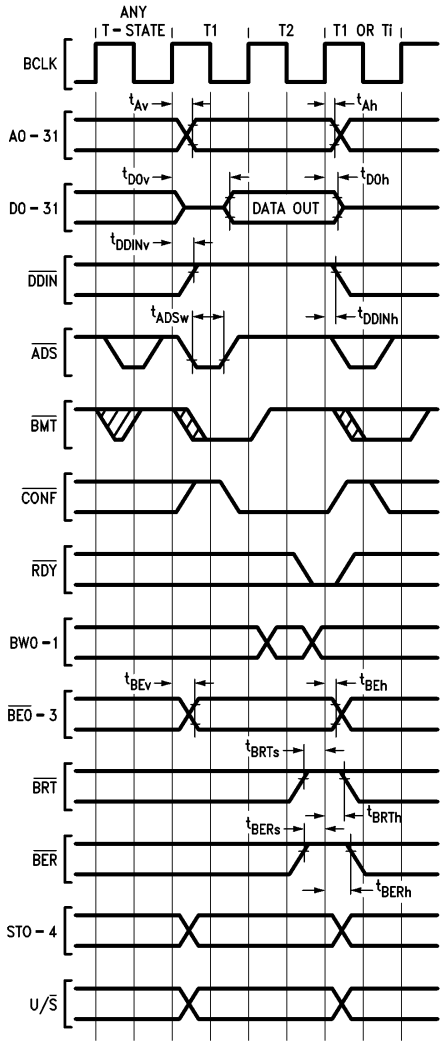
4.0 Device Specifications (Continued)

4.4.3 Timing Diagrams



TL/EE/9354–43

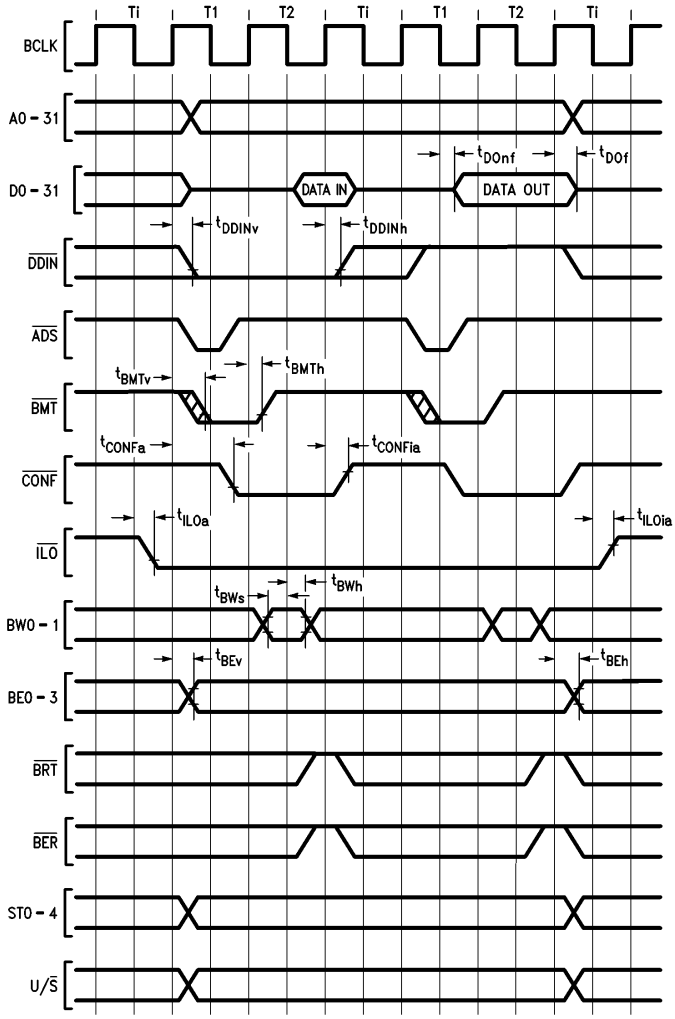
FIGURE 4-5. Basic Read Cycle Timing



TL/EE/9354–44

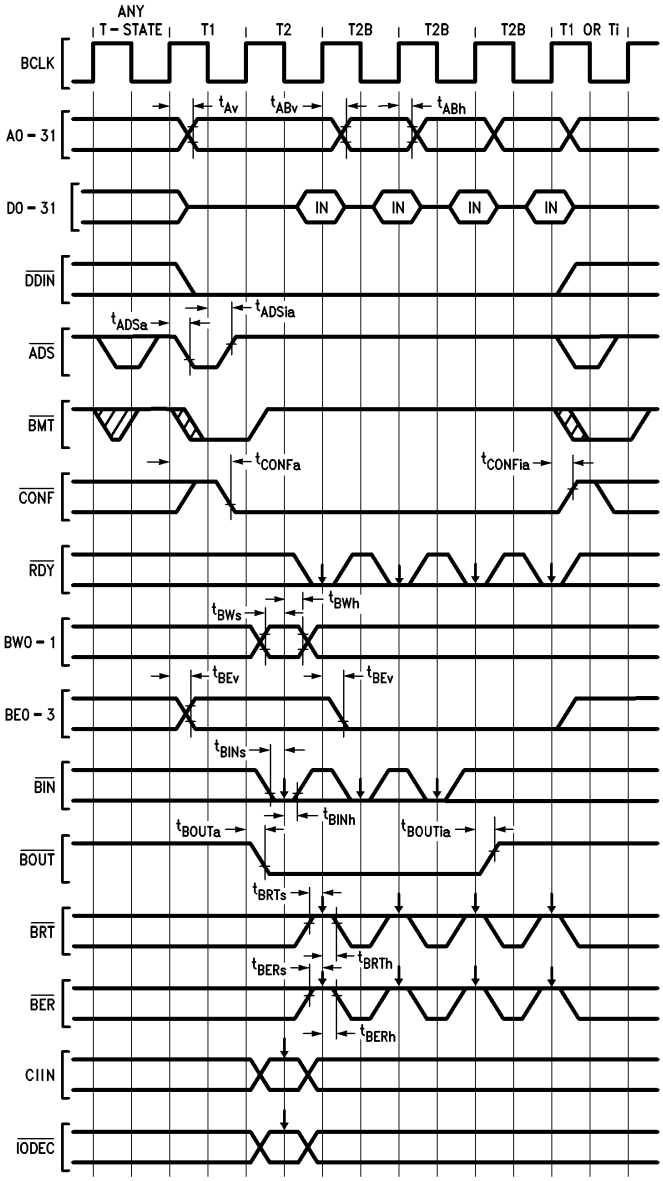
Note: An Idle State is always inserted before a Write Cycle when the Write immediately follows a confirmed Read Cycle.

FIGURE 4-6. Write Cycle Timing



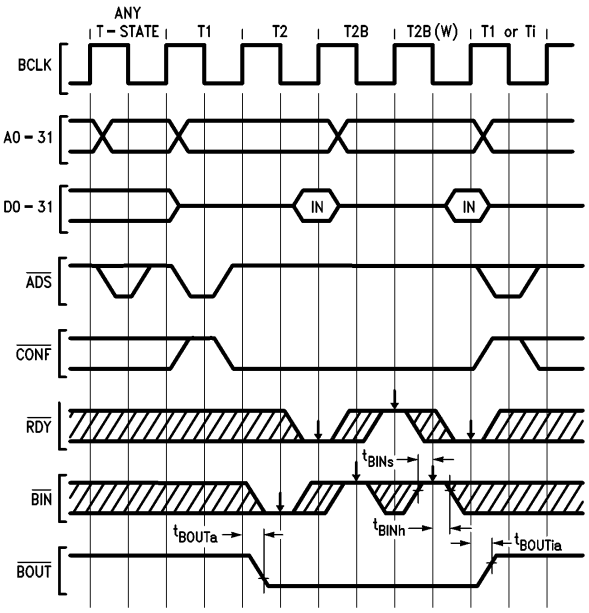
TL/EE/9354–45

FIGURE 4-7. Interlocked Read and Write Cycles



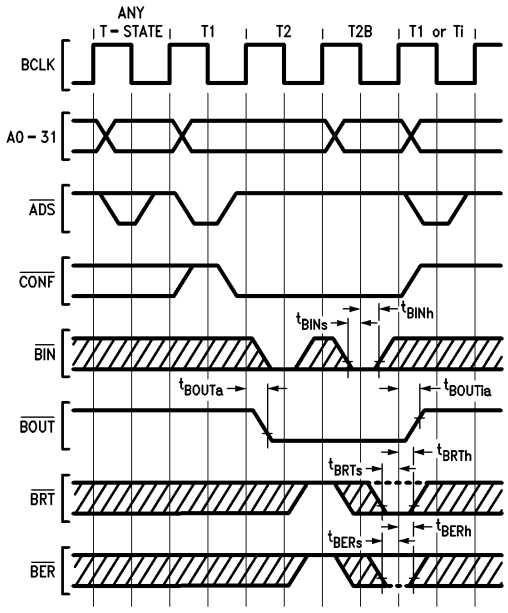
TL/EE/9354–46

FIGURE 4-8. Burst Read Cycles



TL/EE/9354–47

FIGURE 4-9. External Termination of Burst Cycles

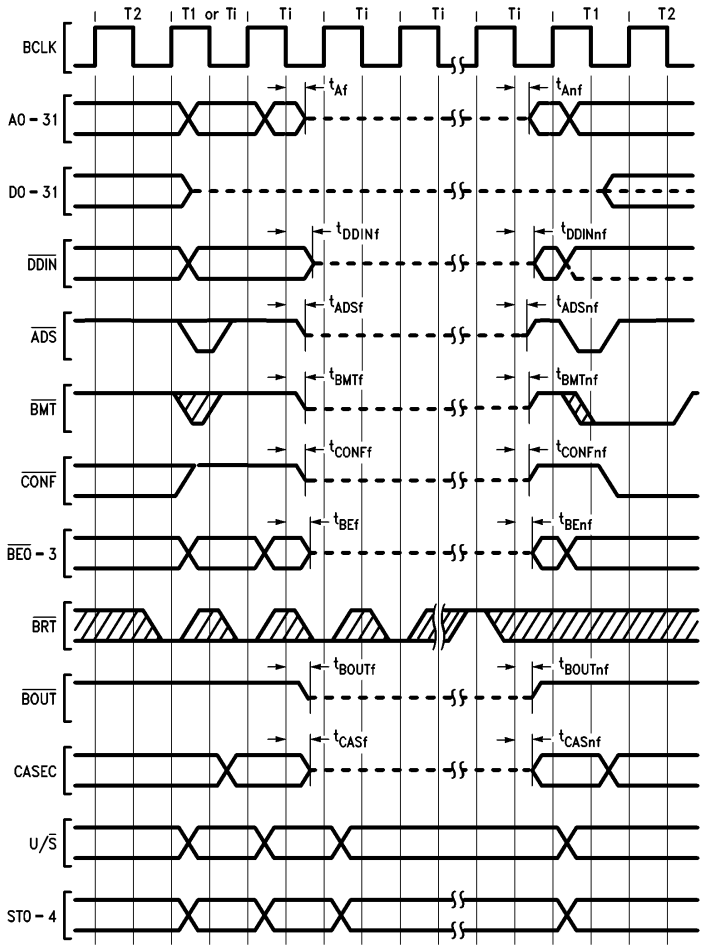


TL/EE/9354–48

FIGURE 4-10. Bus Error or Retry During Burst Cycles

Note: Two idle state are always inserted by the CPU following the assertion of $\overline{BRT}$.

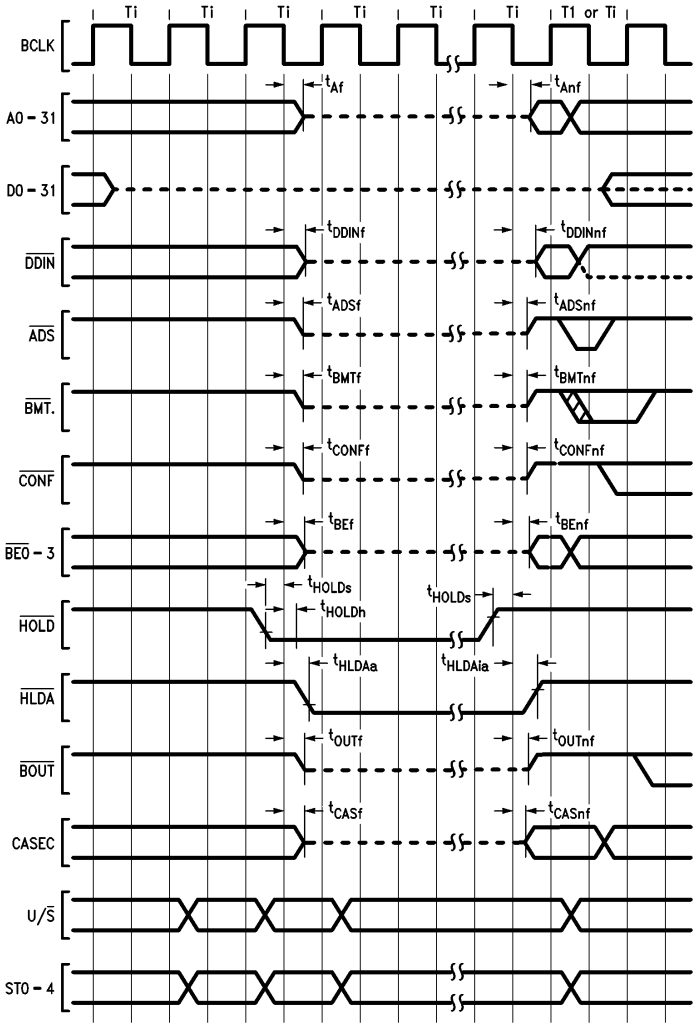
4.0 Device Specifications (Continued)



TL/EE/9354–49

FIGURE 4-11. Extended Retry Timing

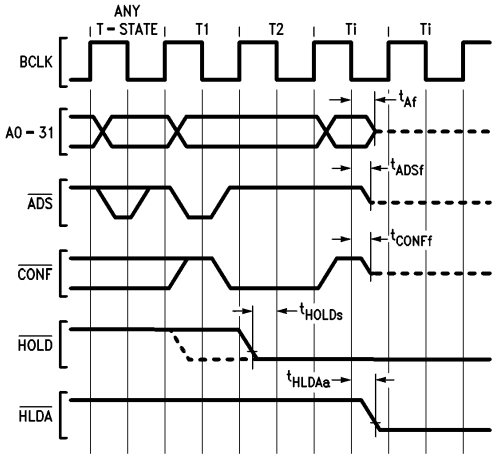
4.0 Device Specifications (Continued)



TL/EE/9354–50

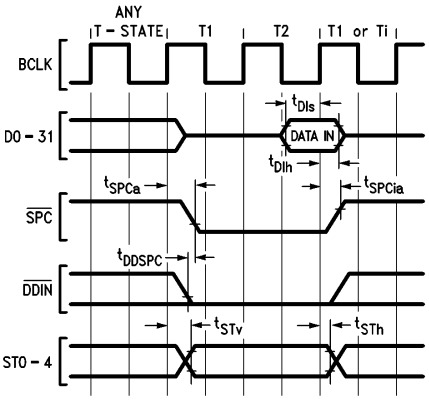
FIGURE 4-12. Hold Timing (Bus Initially Idle)

4.0 Device Specifications (Continued)



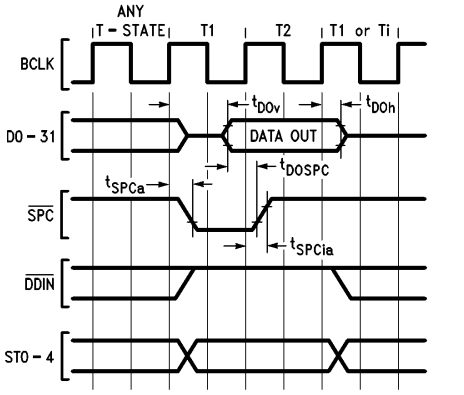
TL/EE/9354–51

FIGURE 4-13. $\overline{\text{HOLD}}$ Acknowledge Timing (Bus Initially Not Idle)



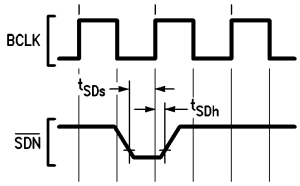
TL/EE/9354–52

FIGURE 4-14. Slave Processor Read Timing



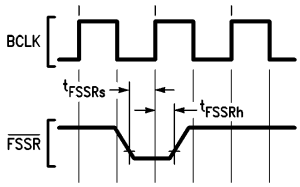
TL/EE/9354–53

FIGURE 4-15. Slave Processor Write Timing



TL/EE/9354–54

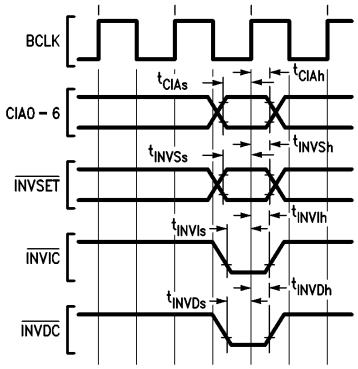
FIGURE 4-16. Slave Processor Done



TL/EE/9354–55

FIGURE 4-17. $\overline{\text{FSSR}}$ Signal Timing

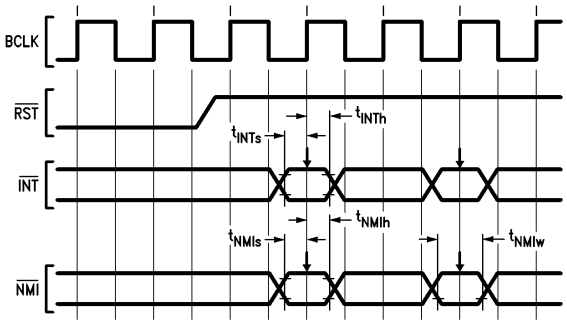
4.0 Device Specifications (Continued)



TL/EE/9354–56

FIGURE 4-18. Cache Invalidation Request

Note 1: CIA0–6 and INVSET are only relevant when INVIC and/or INVDC are asserted.

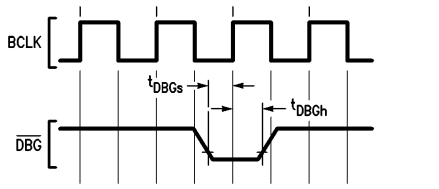


TL/EE/9354–57

FIGURE 4-19. INT and NMI Signals Sampling

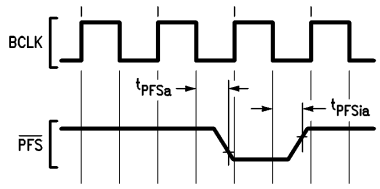
Note 1: INT and NMI are sampled on every other rising edge of BCLK, starting with the second rising edge of BCLK after RST goes high.

Note 2: INT is level sensitive, and once asserted, it should not be deasserted until it is acknowledged.



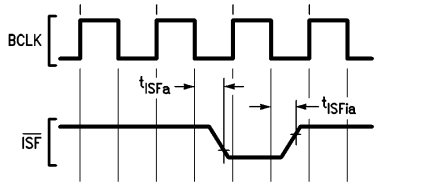
TL/EE/9354–58

FIGURE 4-20. Debug Trap Request



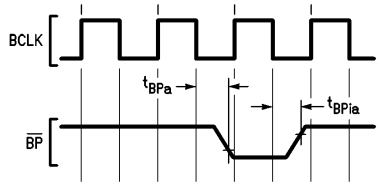
TL/EE/9354–59

FIGURE 4-21. PFS Signal Timing



TL/EE/9354–60

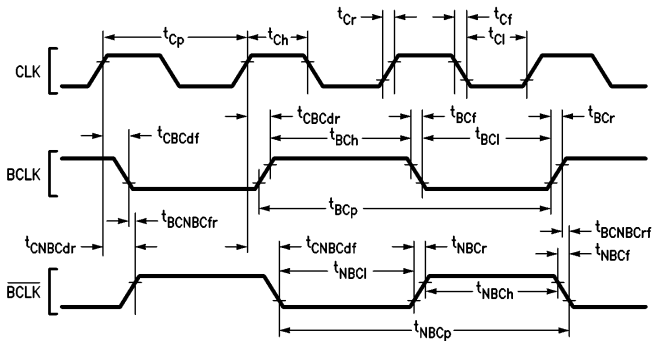
FIGURE 4-22. ISF Signal Timing



TL/EE/9354–61

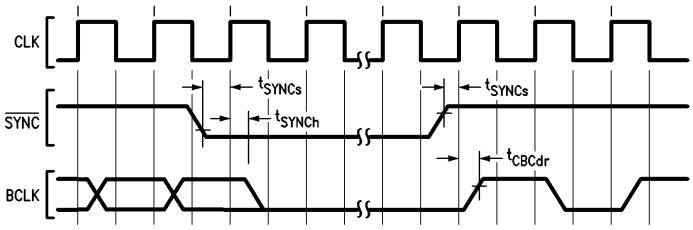
FIGURE 4-23. Break Point Signal Timing

4.0 Device Specifications (Continued)



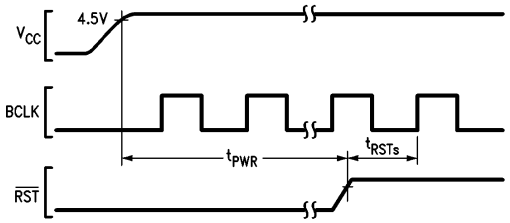
TL/EE/9354–62

FIGURE 4-24. Clock Waveforms



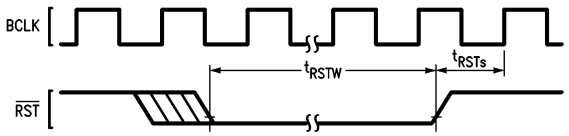
TL/EE/9354–63

FIGURE 4-25. Bus Clock Synchronization



TL/EE/9354–64

FIGURE 4-26. Power-On Reset



TL/EE/9354–65

FIGURE 4-27. Non-Power-On Reset

Appendix A: 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 = Custom Type Field
D = 1 (Double Word)
Q = 0 (Quad Word)

op = Operation Code
Valid encodings shown with each format.

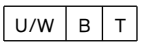
gen, gen 1, gen 2 = General Addressing Mode Field
See Section 2.2 for encodings.

reg = General Purpose 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)

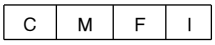
short = Short Immediate value. May contain:
quick: Signed 4-bit value, in MOVQ, ADDQ, CMPQ, ACB.
cond: Condition Code (above), in Scond.
areg: CPU Dedicated Register, in LPR, SPR.
0000 = UPSR
0001 = DCR
0010 = BPC
0011 = DSR
0100 = CAR
0101–0111 = (Reserved)
1000 = FP
1001 = SP
1010 = SB
1011 = USP
1100 = CFG
1101 = PSR
1110 = INTBASE
1111 = MOD

Options: in String Instructions

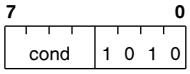


T = Translated
B = Backward
U/W = 00: None
01: While Match
11: Until Match

Configuration bits, in SETCFG Instruction:

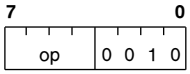


mreg: MMU Register number, in LMR, SMR.
0000 = } Trap (UND)
•
•
•
0111 = }
1000 = Reserved
1001 = MCR
1010 = MSR
1011 = TEAR
1100 = PTB0
1101 = PTB1
1110 = IVAR0
1111 = IVAR1



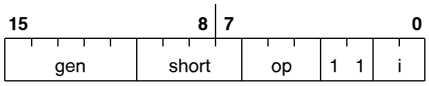
Format 0

Bcond (BR)



Format 1

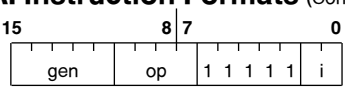
BSR	-0000	ENTER	-1000
RET	-0001	EXIT	-1001
CXP	-0010	NOP	-1010
RXP	-0011	WAIT	-1011
RETT	-0100	DIA	-1100
RETI	-0101	FLAG	-1101
SAVE	-0110	SVC	-1110
RESTORE	-0111	BPT	-1111



Format 2

ADDQ	-000	ACB	-100
CMPQ	-001	MOVQ	-101
SPR	-010	LPR	-110
Scond	-011		

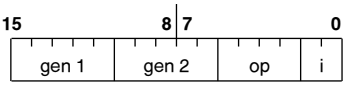
Appendix A: Instruction Formats (Continued)



Format 3

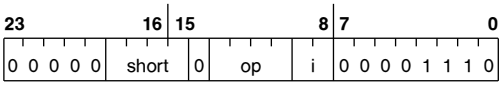
CXPD	-0000	ADJSP	-1010
BICPSR	-0010	JSR	-1100
JUMP	-0100	CASE	-1110
BISPSR	-0110		

Trap (UND) on XXX1, 1000



Format 4

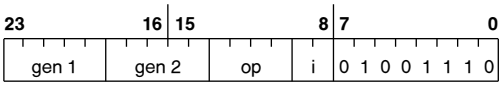
ADD	-0000	SUB	-1000
CMP	-0001	ADDR	-1001
BIC	-0010	AND	-1010
ADDC	-0100	SUBC	-1100
MOV	-0101	TBIT	-1101
OR	-0110	XOR	-1110



Format 5

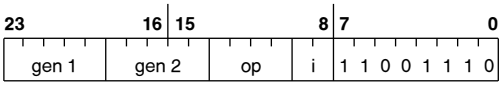
MOVS	-0000	SETCFG	-0010
CMPS	-0001	SKPS	-0011

Trap (UND) on 1XXX, 01XX



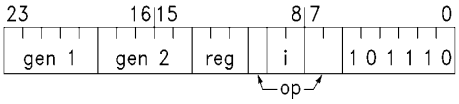
Format 6

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

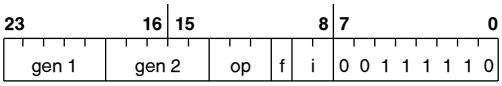
MOVM	-0000	MUL	-1000
CMPM	-0001	MEI	-1001
INSS	-0010	Trap (UND)	-1010
EXTS	-0011	DEI	-1011
MOVXBW	-0100	QUO	-1100
MOVZBW	-0101	REM	-1101
MOVZiD	-0110	MOD	-1110
MOVXiD	-0111	DIV	-1111



TL/EE/9354–66

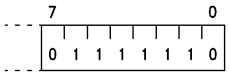
Format 8

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

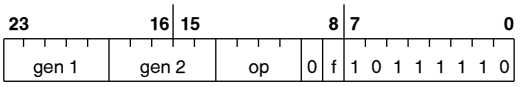
MOVif	-000	ROUND	-100
LFSR	-001	TRUNC	-101
MOVLF	-010	SFSR	-110
MOVFL	-011	FLOOR	-111



TL/EE/9354–67

Format 10

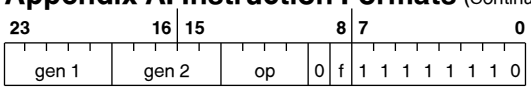
Trap (UND) Always



Format 11

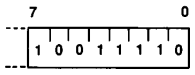
ADDf	-0000	DIVf	-1000
MOVf	-0001	Note 1	-1001
CMPf	-0010	Note 3	-1010
Note 3	-0011	Note 1	-1011
SUBf	-0100	MULf	-1100
NEGf	-0101	ABSf	-1101
Note 2	-0110	Note 2	-1110
Note 1	-0111	Note 1	-1111

Appendix A: Instruction Formats (Continued)



Format 12

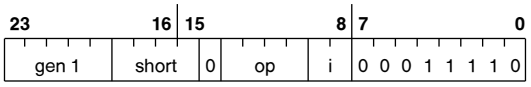
Note 2	-0000	Note 2	-1000
SQRTf	-0001	Note 1	-1001
POLYf	-0010	MACf	-1010
DOTf	-0011	Note 1	-1011
SCALBf	-0100	Note 2	-1100
LOGBf	-0101	Note 1	-1101
Note 2	-0110	Note 2	-1110
Note 1	-0111	Note 1	-1111



TL/EE/9354–68

Format 13

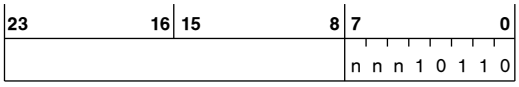
Trap (UND) Always



Format 14

RDVAL	-0000	LMR	-0010
WRVAL	-0001	SMR	-0011
		CINV	-1001

Trap (UND) on 01XX, 1000, 101X, 11XX

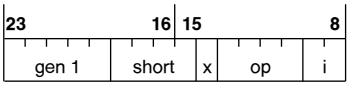


Format 15

(Custom Slave)

nnn **Operation Word Format**

000

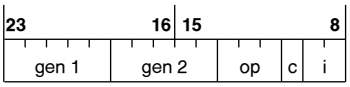


Format 15.0

LCR	-0010
SCR	-0011

Trap (UND) on all others

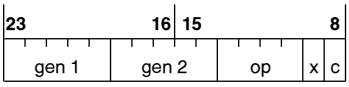
001



Format 15.1

CCV3	-000	CCV2	-100
LCSR	-001	CCV1	-101
CCV5	-010	SCSR	-110
CCV4	-011	CCV0	-111

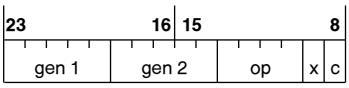
101



Format 15.5

CCAL0	-0000	CCAL3	-1000
CMOV0	-0001	CMOV3	-1001
CCMP0	-0010	Note 3	-1010
CCMP1	-0011	Note 1	-1011
CCAL1	-0100	CCAL2	-1100
CMOV2	-0101	CMOV1	-1101
Note 2	-0110	Note 2	-1110
Note 1	-0111	Note 1	-1111

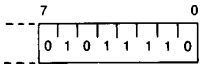
111



Format 15.7

Note 2	-0000	Note 2	-1000
Note 1	-0001	Note 1	-1001
Note 3	-0010	Note 3	-1010
Note 3	-0011	Note 1	-1011
Note 2	-0100	Note 2	-1100
Note 1	-0101	Note 1	-1101
Note 2	-0110	Note 2	-1110
Note 1	-0111	Note 1	-1111

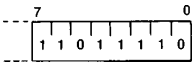
If nnn = 010, 011, 100, 110 then Trap (UND) Always.



TL/EE/9354–69

Format 16

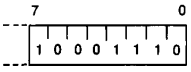
Trap (UND) Always



TL/EE/9354–70

Format 17

Trap (UND) Always

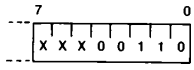


TL/EE/9354–71

Appendix A: Instruction Formats (Continued)

Format 18

Trap (UND) Always

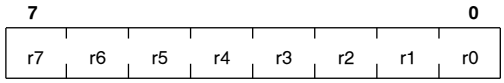


TL/EE/9354–72

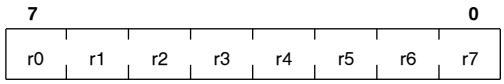
Format 19

Trap (UND) Always

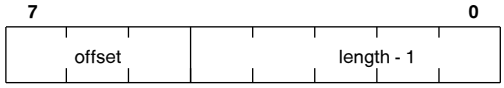
Implied Immediate Encodings:



Register Mark, Appended to SAVE, ENTER



Register Mark, Appended to RESTORE, EXIT



Offset/Length Modifier Appended to INSS, EXTS

Note 1: Opcode not defined; CPU treats like MOV_f or $CMOV_c$. First operand has access class of read; second operand has access class of write; f or c field selects 32- or 64-bit data.

Note 2: Opcode not defined; CPU treats like ADD_f or $CCAL_c$. First operand has access class of read;, second operand has access class of read-modify-write; f or c field selects 32- or 64-bit data.

Note 3: Opcode not defined; CPU treats like CMP_f or $CCMP_c$. First operand has access class of read;, second operand has access class of read; f or c field selects 32- or 64-bit data.

Appendix B. Compatibility Issues

The NS32532 is compatible with the Series 32000 architecture implemented by the NS32032, NS32332, and previous microprocessors in the family. Compatibility means that within certain limited constraints, programs that execute on one of the earlier Series 32000 microprocessors will produce identical results when executed on the NS32532. Compatibility applies to privileged operating systems programs, as well as to non-privileged applications programs. This appendix explains both the restrictions on compatibility with previous Series 32000 microprocessors and the extensions to the architecture that are implemented by the NS32532.

B.1 RESTRICTIONS ON COMPATIBILITY

If the following restrictions are observed, then a program that executes on an earlier Series 32000 microprocessor will produce identical results when executed on the NS32532 in an appropriately configured system:

1. The program is not time-dependent. For example, the program should not use instruction loops to control real-time delays.
2. The program does not use any encodings of instructions, operands, addresses, or control fields identified to be reserved or undefined. For example, if the count operand's value for an LSHi instruction is not within the range specified by the *Series 32000 Instruction Set Reference Manual,* then the results produced by the NS32532 may differ from those of the NS32032.
3. Either the program does not depend on the use of a Memory Management Unit (MMU), or it is written for operation with the NS32382 MMU and does not use the bus-error or debugging features of the NS32382.
4. The program does not depend on the detection of bus errors according to the implementation of the NS32332. For example, the NS32532 distinguishes between restartable and nonrestartable bus errors by transferring control to the appropriate bus-error exception service procedure through one of two distinct entries in the Interrupt Dispatch Table. In contrast, the NS32332 uses a single entry in the Interrupt Dispatch Table for all bus errors.
5. The program does not modify itself. Refer to Section B.4 for more information.
6. The program does not depend on the execution of certain complex instructions to be non-interruptible. Refer to Section B.5 on. ''Memory-Mapped I/O'' for more information.
7. The program does not use the custom slave instructions CATSTO and CATST1, as they are not supported by the NS32532 and will result in a Trap (UND) when their execution is attempted.

B.2 ARCHITECTURE EXTENSIONS

The NS32532 implements the following extensions of the Series 32000 architecture using previously reserved control bits, instruction encodings, and memory locations. Extensions implemented earlier in the NS32332, such as 32-bit addressing, are not listed.

1. The DC, LDC, IC, and LIC bits in the CFG register have been defined to control the on-chip Instruction and Data Caches. The DE-bit in the CFG register has been defined to enable Direct-Exception Mode.
2. The V-flag in the PSR register has been defined to enable the Integer-Overflow Trap.
3. The DCR, BPC, DSR, and CAR registers have been defined to control debugging features. Access to these registers has been added to the definition of the LPR and SPR instructions.
4. Access to the CFG and SP1 registers has been added to the definition of the LPR and SPR instructions.
5. The CINV instruction has been defined to invalidate control of the on-chip Instruction and Data Caches.
6. Direct-Exception Mode has been added to support faster interrupt service time and systems without module tables.
7. A new entry has been added to the Interrupt Dispatch Table for supporting vectors to distinguish between restartable and nonrestartable bus errors. Two additional entries support Trap (OVF) and Trap (DBG).
8. Restrictions have been eliminated for recovery from Trap (ABT) for operands with access class of write that cross page boundaries. Restrictions still exist however, for the operands of the MOVMi instruction.

B.3 INTEGER OVERFLOW TRAP

A new trap condition is recognized for integer arithmetic overflow. Trap (OVF) is enabled by the V-flag in the PSR. This new trap is important because detection of integer overflow conditions is required for certain programming languages, such as ADA, and the PSR flags do not indicate the occurrence of overflow for ASHi, DIVi and MULi instructions.

Appendix C. Instruction Set Extensions (Continued)

C.1 PROCESSOR SERVICE INSTRUCTIONS

The CFG register, User Stack Pointer (SP1), and Debug Registers can be loaded and stored using privileged forms of the LPRi and SPRi instructions.

When the SETCFG instruction is executed, the CFG register bits 0 through 3 are loaded from the instruction's short field, bits 4 through 7 are forced to 1, and bits 8 through 12 are forced to 0.

The contents of the on-chip Instruction Cache and Data Cache can be invalidated by executing the privileged instruction CINV. While executing the CINV instruction, the CPU generates 2 slave bus cycles on the system interface to display the first 3 bytes of the instruction and the source operand. External circuitry can thereby detect the execution of the CINV instruction for use in monitoring the contents of the on-chip caches.

C.2 MEMORY MANAGEMENT INSTRUCTIONS

The NS32532 on-chip MMU does not implement the BAR, BDR, BEAR, and BMR registers of the NS32382. These registers are used in the NS32382 to support bus error and debugging features. When an attempt is made to access one of these 4 registers by executing an LMR or SMR instruction, a Trap (UND) occurs. More generally, a Trap (UND) occurs whenever an attempt is made to execute an LMR or SMR instruction and the most-significant bit of the short-field is 0.

While executing an LMR instruction, the CPU generates 2 slave bus cycles on the system interface to display the first 3 bytes of the instruction and the source operand. External circuitry can thereby detect the execution of an LMR instruction for use in monitoring the contents of the on-chip Translation Lookaside Buffer.

Like the NS32382 MMU, the F-flag in the PSR is set and no Trap (ABT) occurs when a RDVAL or WRVAL instruction is executed and the Protection Level in the Level-1 Page Table Entry indicates that the access is not allowed. In the NS32082 MMU, an abort occurs when the Level-1 PTE is invalid, regardless of the Protection Level.

C.3 INSTRUCTION DEFINITIONS

This section provides a description of the operations and encodings of the new NS32532 privileged instructions.

Load and Store Processor Registers

Syntax:	**LPRi**	**procreg,**	**src**
		short	gen
			read.i
	SPRi	**procreg**	**dest**
		short	gen
			write.i

The LPRi and SPRi instructions can be used to load and store the User Stack Pointer (USP or SP1), the Configuration Register (CFG) and the Debug Registers in addition to the Processor Registers supported by the previous Series 32000 CPUs. Access to these registers is privileged.

Figure C-1 and Table C-1 show the instruction formats and the new 'short' field encodings for LPRi and SPRi.

Flags Affected: No flags affected by loading or storing the USP, CFG, or Debug Registers.

Traps: Illegal Instruction Trap (ILL) occurs if an attempt is made to load or store the USP, CFG or Debug Registers while the U-flag is 1.

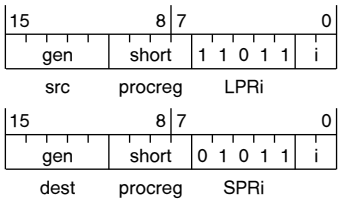


FIGURE C-1. LPRi/SPRi Instruction Formats

TABLE C-1. LPRi/SPRi New 'Short' Field Encodings

Register	procreg	short field
Debug Condition Register	DCR	0001
Breakpoint Program Counter	BPC	0010
Debug Status Register	DSR	0011
Compare Address Register	CAR	0100
User Stack Pointer	USP	1011
Configuration Register	CFG	1100

Cache Invalidate

Syntax:	**CINV**	**[options], src**	
		gen	
		read. D	

The CINV instruction invalidates the contents of locations in the on-chip Instruction Cache and Data Cache. The instruction can be used to invalidate either the entire contents of the on-chip caches or only a 16-byte block. In the latter case, the 28 most-significant bits of the source operand specify the physical address of the aligned 16-byte block; the 4 least-significant bits of the source operand are ignored. If the specified block is not located in the on-chip caches, then the instruction has no effect. If the entire cache contents is to be invalidated, then the source operand is read, but its value is ignored.

Options are specified by listing the letters A (invalidate All), I (Instruction Cache), and D (Data Cache). If neither the I nor D option is specified, the instruction has no effect.

In the instruction encoding, the options are represented in the A, I, and D fields as follows:

A: 0—invalidate only a 16-byte block
1—invalidate the entire cache

I: 0—do not affect the Instruction Cache
1—invalidate the Instruction Cache

D: 0—do not affect the Data Cache
1—invalidate the Data Cache

Flags Affected: None

Traps: Illegal Operation Trap (ILL) occurs if an attempt is made to execute this instruction while the U-flag is 1.

Examples:

1. CINV [A, D, I], R3 1E A7 1B

2. CINV [I], R3 1E 27 19

Example 1 invalidates the entire Instruction Cache and Data Cache.

Example 2 invalidates the 16-byte block whose physical address in the Instruction Cache is contained in R3.

Appendix C. Instruction Set Extensions (Continued)

C.1 PROCESSOR SERVICE INSTRUCTIONS

The CFG register, User Stack Pointer (SP1), and Debug Registers can be loaded and stored using privileged forms of the LPRi and SPRi instructions.

When the SETCFG instruction is executed, the CFG register bits 0 through 3 are loaded from the instruction's short field, bits 4 through 7 are forced to 1, and bits 8 through 12 are forced to 0.

The contents of the on-chip Instruction Cache and Data Cache can be invalidated by executing the privileged instruction CINV. While executing the CINV instruction, the CPU generates 2 slave bus cycles on the system interface to display the first 3 bytes of the instruction and the source operand. External circuitry can thereby detect the execution of the CINV instruction for use in monitoring the contents of the on-chip caches.

C.2 MEMORY MANAGEMENT INSTRUCTIONS

The NS32532 on-chip MMU does not implement the BAR, BDR, BEAR, and BMR registers of the NS32382. These registers are used in the NS32382 to support bus error and debugging features. When an attempt is made to access one of these 4 registers by executing an LMR or SMR instruction, a Trap (UND) occurs. More generally, a Trap (UND) occurs whenever an attempt is made to execute an LMR or SMR instruction and the most-significant bit of the short-field is 0.

While executing an LMR instruction, the CPU generates 2 slave bus cycles on the system interface to display the first 3 bytes of the instruction and the source operand. External circuitry can thereby detect the execution of an LMR instruction for use in monitoring the contents of the on-chip Translation Lookaside Buffer.

Like the NS32382 MMU, the F-flag in the PSR is set and no Trap (ABT) occurs when a RDVAL or WRVAL instruction is executed and the Protection Level in the Level-1 Page Table Entry indicates that the access is not allowed. In the NS32082 MMU, an abort occurs when the Level-1 PTE is invalid, regardless of the Protection Level.

C.3 INSTRUCTION DEFINITIONS

This section provides a description of the operations and encodings of the new NS32532 privileged instructions.

Load and Store Processor Registers

Syntax:	**LPRi**	**procreg,**	**src**
		short	gen
			read.i
	SPRi	**procreg**	**dest**
		short	gen
			write.i

The LPRi and SPRi instructions can be used to load and store the User Stack Pointer (USP or SP1), the Configuration Register (CFG) and the Debug Registers in addition to the Processor Registers supported by the previous Series 32000 CPUs. Access to these registers is privileged.

Figure C-1 and Table C-1 show the instruction formats and the new 'short' field encodings for LPRi and SPRi.

Flags Affected: No flags affected by loading or storing the USP, CFG, or Debug Registers.

Traps: Illegal Instruction Trap (ILL) occurs if an attempt is made to load or store the USP, CFG or Debug Registers while the U-flag is 1.

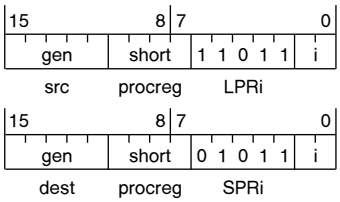


FIGURE C-1. LPRi/SPRi Instruction Formats

TABLE C-1. LPRi/SPRi New 'Short' Field Encodings

Register	procreg	short field
Debug Condition Register	DCR	0001
Breakpoint Program Counter	BPC	0010
Debug Status Register	DSR	0011
Compare Address Register	CAR	0100
User Stack Pointer	USP	1011
Configuration Register	CFG	1100

Cache Invalidate

Syntax:	**CINV**	**options, src**	
			gen
			read. D

The CINV instruction invalidates the contents of locations in the on-chip Instruction Cache and Data Cache. The instruction can be used to invalidate either the entire contents of the on-chip caches or only a 16-byte block. In the latter case, the 28 most-significant bits of the source operand specify the physical address of the aligned 16-byte block; the 4 least-significant bits of the source operand are ignored. If the specified block is not located in the on-chip caches, then the instruction has no effect. If the entire cache contents is to be invalidated, then the source operand is read, but its value is ignored.

Options are specified by listing the letters A (invalidate All), I (Instruction Cache), and D (Data Cache). If neither the I nor D option is specified, the instruction has no effect.

In the instruction encoding, the options are represented in the A, I, and D fields as follows:

A: 0—invalidate only a 16-byte block
1—invalidate the entire cache

I: 0—do not affect the Instruction Cache
1—invalidate the Instruction Cache

D: 0—do not affect the Data Cache
1—invalidate the Data Cache

Flags Affected: None

Traps: Illegal Operation Trap (ILL) occurs if an attempt is made to execute this instruction while the U-flag is 1.

Examples:

1. CINV A, D, I, R3 1E A7 1B

2. CINV I, R3 1E 27 19

Example 1 invalidates the entire Instruction Cache and Data Cache.

Example 2 invalidates the 16-byte block whose physical address in the Instruction Cache is contained in R3.

Appendix C. Instruction Set Extensions (Continued)

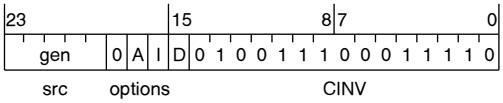


FIGURE C-2. CINV Instruction Format

Load and Store Memory Management Register

Syntax:	**LMR**	**mmreg,**	**src**
		short	gen read.D
	SMR	**mmureg,**	**dest**
		short	gen write.D

The LMR and SMR instruction load and store the on-chip MMU registers as 32-bit quantities to and from any general operand. For reasons of system security, these instructions are privileged. In order to be executable, they must also be enabled by setting the M bit in the CFG register.

The instruction formats as well as the 'short' field encodings are shown in *Figure C-3* and Table C-2 respectively.

It is to be noted that the IVAR0 and IVAR1 registers are write-only, and as such, they can only be loaded by the LMR instruction.

Flags Affected: none

Traps: Undefined Instruction Trap (UND) occurs if an attempt is made to execute this instruction while either of the following conditions is true:

1. The M-bit in the CFG register is 0.
2. The U-Flag in the PSR is 0 and the most-significant bit of the short field is 0.

Illegal Instruction Trap (ILL) occurs if an attempt is made to execute this instruction while the M-bit in the CFG register and the U-flag in the PSR are both 1.

TABLE C-2. LMR/SMR 'Short' Field Encodings

Register	mmureg	short field
Memory Management Control Reg	MCR	1001
Memory Management Status Reg	MSR	1010
Translation Exception Address Reg	TEAR	1011
Page Table Base Register 0	PTB0	1100
Page Table Base Register 1	PTB1	1101
Invalidate Virtual Address 0	IVAR0	1110
Invalidate Virtual Address 1	IVAR1	1111

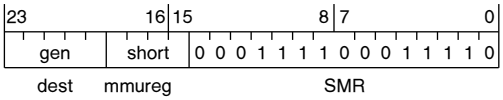
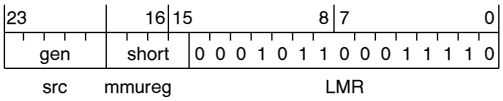


FIGURE C-3. LMR/SMR Instruction Formats



Appendix D. Instruction Execution Times

The NS32532 achieves its performance by using an advanced implementation incorporating a 4-stage Instruction Pipeline, a Memory Management Unit, an Instruction Cache and a Data Cache into a single integrated circuit.

As a consequence of this advanced implementation, the performance evaluation for the NS32532 is more complex than for the previous microprocessors in the Series 32000 family. In fact, it is no longer possible to determine the execution time for an instruction using only a set of tables for operations and addressing modes. Rather, it is necessary to consider dependencies between the various instructions executing in the pipeline, as well as the occurrence of misses for the on-chip caches.

The following sections explain the method to evaluate the performance of the NS32532 by calculating various timing parameters for an instruction sequence. Due to the high degree of parallelism in the NS32532, the evaluation techniques presented here include some simplifications and approximations.

D.1 INTERNAL ORGANIZATION AND INSTRUCTION EXECUTION

The NS32532 is organized internally as 8 functional units as shown in *Figure 1*. The functional units operate in parallel to execute instructions in the 4-stage pipeline. The structure of this pipeline is shown in *Figure 3-2*. The Instruction Fetch and Instruction Decode pipeline stages are implemented in the loader along with the 8-byte instruction queue and the buffer for a decoded instruction. The Address Calculation pipeline stage is implemented in the address unit. The Execute pipeline stage is implemented in the Execution Unit along with the write data buffer that holds up to two results directed to memory.

The Address Unit and Execution Unit can process instructions at a peak rate of 2 clock cycles per instruction, enabling a sustained pipeline throughput at 30 MHz of 15 MIPS (million instructions per second) for sequences of register-to-register, immediate-to-register, register-to-memory and memory-to-register instructions. Nevertheless, the execution of instructions in the pipeline is reduced from the peak throughput of 2 cycles by the following causes of delay:

1. Complex operations, like division, require more than 2 cycles in the Execution Unit, and complex addressing modes, like memory relative, require more than 2 cycles in the Address Unit.

2. Dependencies between instructions can limit the flow through the pipeline. A data dependency can arise when the result of one instruction is the source of a following instruction. Control dependencies arise when branching instructions are executed. Section D.3 describes the types of instruction dependencies that impact performance and explains how to calculate the pipeline delays.
3. Cache and TLB misses can cause the flow of instructions through the pipeline to be delayed, as can non-aligned references. Section D.4 explains the performance impact for these forms of storage delays.

The effective time T_{eff} needed to execute an instruction is given by the following formula:

$$T_{eff} = T_e + T_d + T_s$$

T_e is the execution time in the pipeline in the absence of data dependencies between instructions and storage delays, T_d is the delay due to data dependencies, and T_s is the effect of storage delays.

D.2 BASIC EXECUTION TIMES

Instruction flow in sequence through the pipeline stages implemented by the Loader, Address Unit, and Execution Unit. In almost all cases, the Loader is at least as fast at decoding an instruction as the Address Unit is at processing the instruction. Consequently, the effects of the Loader can be ignored when analyzing the smooth flow of instructions in the pipeline, and it is only necessary to consider the times for the Address Unit and Execution Unit. The time required by the Loader to fetch and decode instructions is significant only when there are control dependencies between instructions or Instruction Cache misses, both of which are explained later.

The time for the pipeline to advance from one instruction to the next is typically determined by the maximum time of the Address Unit and Execution Unit to complete processing of the instruction on which they are operating. For example, if the Execution Unit is completing instruction *n* in 2 cycles and the Address Unit is completing instruction *n+1* in 4 cycles, then the pipeline will advance in 4 cycles. For certain instructions, such as RESTORE, the Address Unit waits until the Execution Unit has completed the instruction before proceeding to the next instruction. When such an instruction is in the Execution Unit, the time for the pipeline to advance is equal to the sum of the time for the Execution Unit to complete instruction *n* and the time for the Address Unit to complete instruction *n+1*. The processing times for the Loader, Address Unit, and Execution Unit are explained below.

D.2.1 Loader Timing

The Loader can process an instruction field on each clock cycle, where a *field* is one of the following:

- An opcode of 1 to 3 bytes including addressing mode specifiers.
- Up to 2 index bytes, if scaled index addressing mode is used.
- A displacement.
- An immediate value of 8, 16 or 32 bits.

The Loader requires additional time in the following cases:

- 1 additional cycle when 2 consecutive double-word fields begin at an odd address.
- 2 cycles in total to process a double-precision floating-point immediate value.

D.2.2 Address Unit Timing

The processing time of the Address Unit depends on the instruction's operation and the number and type of its general addressing modes. The basic time for most instructions is 2 cycles. A relatively small number of instructions require an additional address unit time, as shown in the timing tables in Section D.5.5. Non-pipelined floating-point instructions as well as Custom-Slave instructions require an additional 3 cycles plus 2 cycles for each quad-word operand in memory.

For instructions with 2 general addressing modes, 2 additional cycles are required when both addressing modes refer to memory. Certain general addressing modes require an additional processing time, as shown in Table D-1. For example, the instruction **MOVD 4(8(FP)), TOS** requires 7 cycles in the Address Unit; 2 cycles for the basic time, an additional 2 cycles because both modes refer to memory, and an additional 3 cycles for Memory Relative addressing mode.

TABLE D-1. Additional Address Unit Processing Time for Complex Addressing Modes

Mode	Additional Cycles
Memory Relative	3
External	8
Scaled Indexing	2

D.2.3 Execution Unit Timing

The Execution Unit processing times for the various NS32532 instructions are provided in Section D.5.5. Certain operations cause a break in the instruction flow through the pipeline.

Some of these operation simply stop the Address Unit, while others flush the instruction queue as well. The information on how to evaluate the penalty resulting from instruction flow breaks is provided in the following sections.

D.3 INSTRUCTION DEPENDENCIES

Interactions between instructions in the pipeline can cause delays. Two types of interactions can arise, as described below.

D.3.1 Data Dependencies

In certain circumstances the flow of instructions in the pipeline will be delayed when the result of an instruction is used as the source of a succeeding instruction. Such interlocks are automatically detected by the microprocessor and handled with complete transparency to software.

D.3.1.1 Register Interlocks

When an instruction uses a base register that is the destination of either of the previous 2 instructions, a delay occurs. The delay is 3 cycles when, as in the following example, the base register is modified by the immediately preceding instruction. Modifications of the Stack Pointer resulting from the use of TOS addressing mode do not cause any delay. Also, there is no delay for a data dependency when the instruction that modifies the register is one for which the Address Unit stops.

```
  n: ADDD R1,R0     ; modify R0
n+1: MOVD 4(R0),R2  ; R0 is base register,
                      delay 3 cycles
```

The delay is 1 cycle when the register is modified 2 instructions before its use as a base register, as shown in this example.

```
  n: ADDD R1,R0     ; modify R0
n+1: MOVD 4(SP),R3  ; R0 not used
n+2: MOVD 4(R0),R2  ; R0 is base register,
                      delay 1 cycle
```

When an instruction uses an index register that is the destination of the previous instruction, a delay of 1 cycle occurs, as shown in the example below. If the register is modified 2 or more instructions prior to its use as an index register, then no delay occurs.

```
  n: ADDD R1,R0     ; modify R0
n+1: MOVD 4(SP)[R0:B],R2
                    ; R0 is index register,
                      delay 1 cycle
```

Bypass circuitry in the Execution Unit generally avoids delay when a register modified by one instruction is used as the source operand of the following instruction, as in the following example.

```
  n: ADDD R1,R0     ; modify R0
n+1: MOVD R0,R2     ; R0 is source register,
                      no delay
```

For the uncommon case where the operand in the source register is larger than the destination of the previous instruction, a delay of 2 cycles occurs. Here is an example.

```
  n: ADDB R1,R0     ; modify byte in R0
n+1: MOVD R0,R2     ; R0 dw source operand,
                      2 cycle delay
```

Note: The Address Unit does not make any differentiation between CPU and FPU registers. Therefore, register interlocks can occur between integer and floating-point instructions.

D.3.1.2 Memory Interlocks

When an instruction reads a source operand (or address for effective address calculation) from memory that depends on the destination of either of the previous 2 instructions, a delay occurs. The CPU detects a dependency between a read and a write reference in the following cases, which include some false dependencies in addition to all actual dependencies:

- Either reference crosses a double-word boundary
- Address bits 0 through 11 are equal
- Address bits 2 through 11 are equal and either reference is for a word
- Address bits 2 through 11 are equal and either reference is for a double-word

The delay for a memeory interlock is 4 cycles when, as in the following example, the memory location is modified by the immediately preceding instruction.

```
  n: ADDQD 1,4(SP)  ; modify 4(SP)
n+1: CMPD 10,4(SP)  ; read, 4(SP),
                      4 cycle delay
```

The delay is 2 cycles when the memory location is modified 2 instructions before its use as a source operand or effective address, as shown in this example.

```
  n: ADDQD 1,4(SP)  ; modify 4(SP)
n+1: MOVD R0,R1     ; no reference to 4(SP)
n+2: CMPD 10, 4(SP); read 4(SP),
                      2 cycles delay
```

Certain sequences of read and write references can cause a delay of 1 cycle although there is no data dependency between the references. This arises because the Data Cache is occupied for 2 cycles on write references. In the absence of data dependencies, read references are given priority over write references. Therefore, this delay only occurs when an instruction with destination in memory is followed 2 instructions later by an instruction that refers to memory (read or write) and 3 instructions later by an instruction that reads from memory. Here is an example:

```
  n: MOVD R0,4(SP)  ; memory write
n+1: MOVD R6,R7     ; any instruction
n+2: MOVD 8(SP),R0  ; memory read or write
n+3: MOVD 12(SP),R1; memory read
                      delayed 1 cycle
```

D.3.2 Control Dependencies

The flow of instructions through the pipeline is delayed when the address from which to fetch an instruction depends on a previous instruction, such as when a conditional branch is excuted. The Loader includes special circuitry to handle branch instructions (ACB, BR, Bcond, and BSR) that serves to reduce such delays. When a branch instruction is decoded, the Loader calculates the destination address and selects between the sequential and non-sequential instruction streams. The non-sequential stream is selected for unconditional branches. For conditional branches the selection is based on the branch's direction (forward or backward) as well as the tested condition. The branch is predicted taken in any of the following cases.

- The branch is backward.
- The tested condition is either NE or LE.

Measurements have shown that the correct stream is selected for 64% of conditional branches and 71% of total branches.

If the Loader selects the non-sequential stream, then the destination address is transferred to the Instruction Cache. For conditional branches, the Loader saves the address of the alternate stream (the one not selected). When a conditional branch instruction reaches the Execution Unit, the condition is resolved, and the Execution Unit signals the Loader whether or not the branch was taken. If the branch had been incorrectly predicted, the Instruction Cache begins fetching instructions from the correct stream.

The delay for handling a branch instruction depends on whether the branch is taken and whether it is predicted correctly. Unconditional branches have the same delay as correctly predicted, taken conditional branches.

Another form of delay occurs when 2 consecutive conditional branch instructions are executed. This delay of 2 cycles arises from contention for the register that holds the alternate stream address in the Loader.

Control dependencies also arise when JUMP, RET, and other non-branch instructions alter the sequential execution of instructions.

D.4 STORAGE DELAYS

The flow of instructions in the pipeline can be delayed by off-chip memory references that result from misses in the on-chip storage buffers and by misalignment of instructions and operands. These considerations are explained in the following sections. The delays reported assume no wait states on the external bus and no interference between instruction and data references.

D.4.1 Instruction Cache Misses

An Instruction Cache miss causes a 5 cycle gap in the fetching of instructions. When the miss occurs for a non-sequential instruction fetch, the pipeline is idle for the entire gap, so the delay is 5 cycles. When the miss occurs for a sequential fetch, the pipeline is not idle for the entire gap because instructions that have been prefetched ahead and buffered can be executed. The delay for misses on non-sequential instruction fetches can be estimated to be approximately half the gap, or 2.5 cycles.

D.4.2 Data Cache Misses

A Data Cache miss causes a delay of 2 cycles. When a burst read cycle is used to fill the cache block, then 3 additional cycles are required to update the Data Cache. In case a burst cycle is used and either of the 2 instructions following the instruction that caused the miss also reads from memory, then an additional delay occurs: 3 cycle delay when the instruction that reads from memory immediately follows the miss, and 2 cycle delay when the memory read occurs 2 instructions after the miss.

D.4.3 TLB Misses

There is a delay for the MMU to translate a virtual address whenever there is a TLB miss for an instruction fetch, data read or data write and whenever the M-bit in the Page Table Entry (PTE) must be set for a data write that hits in the TLB. The delay for the MMU to handle a TLB miss is 15 cycles when no update to the PTEs is necessary. When only the Level-1 PTE must be updated, the delay is 17 cycles; when only the Level-2 PTE must be updated, the delay is 22 cycles. When both PTEs must be updated, the delay is 24 cycles.

D.4.4 Instruction and Operand Alignment

When a data reference (either read or write) crosses a double-word boundary, there is a delay of 2 cycles.

When the opcode for a non-sequential instruction crosses a double-word boundary, there is a delay of 1 cycle. No delay occurs in the same situation for a sequential instruction. There is also a delay of 2 cycles when an instruction fetch is located on a different page from the previous fetch and there is a hit in the Instruction Cache. This delay, which is due to the time required to translate the new page's address, also occurs following any serializing operation.

D.5 EXECUTION TIME CALCULATIONS

This section provides the necessary information to calculate the T_e portion of the effective time required by the CPU to execute an instruction.

The effects of data dependencies and storage delays are not taken into account in the evaluation of T_e, rather, they should be separately evaluated through a careful examination of the instruction sequence.

The following assumptions are made:

— The entire instruction, with displacements and immediate operands, is present in the instruction queue when needed.

— All memory operands are available to the Execution Unit and Address Unit when needed.

— Memory writes are performed at full speed through the write buffer.

— Where possible, the values of operands are taken into consideration when they affect instruction timing, and a range of times is given. When this is not done, the worst case is assumed.

D.5.1 Definitions

T_{eu} Time required by the Execution Unit to execute an instruction.

T_{au} Total processing time in the Address Unit.

T_{ad} Extra time needed by the Address Unit, in addition to the basic time, to process more complex cases. T_{ad} can be evaluated as follows:

$T_{ad} = T_x + T_{y1} + T_{y2}$

$T_x = 2$ if the instruction has two general operands and both of them are in memory.

0 otherwise.

T_{y1} and T_{y2} are related to operands 1 and 2 respectively. Their values are given below.

$T_{y(1, 2)} = 3$ if Memory Relative

8 if External

2 if Scaled Indexing

0 if any other addressing mode

The following parameters are only used for floating-point execution time calculations.

T_{anp} Additional Address Unit time needed to process floating-point instructions in non-pipelined mode. (Section D.2.2).

T_{anp} may be totally hidden for pipelined instructions. For non-pipelined instructions it can be calculated as follows:

$T_{anp} = 3 + 2 *$ (Number of 64-bit operands in memory)

T_{tcs} Time required to transfer ID and Opcode, if no operand needs to be transferred to the slave. Otherwise, it is the time needed to transfer the last 32 bits of operand data to the slave. In the latter case the transfer of ID and Opcode as well as any operand data except the last 32 bits is included in the Execution Unit timing.

T_{tsc} Time required by the CPU to complete the floating-point instruction upon receiving the DONE signal from the slave. This includes the time to process the DONE signal itself in addition to the time needed to read the result (if any) from the slave.

I This parameter is related to the floating-point operand size as follows:

Standard floating (32 bits): I = 0

Long floating (64 bits): I = 1

D.5.2 Notes on Table Use

1. In the T_{eu} column the notation n1 → n2 means n1 minimum, n2 maximum.
2. In the notes column, notations held within angle brackets <> indicate alternatives in the operand addressing modes which affect the execution time. A table entry which is affected by the operand addressing may have multiple values, corresponding to the alternatives. This addressing notations are:
   - <I> Immediate
   - <R> CPU register
   - <M> Memory
   - <F> FPU register, either 32 or 64 bits
   - <m> Memory, except Top of Stack
   - <T> Top of Stack
   - <x> Any addressing mode
   - <ab> a and b represent the addressing modes of operands 1 and 2 respectively. Both of them can be any addressing mode. (e.g., <MR> means memory to CPU register).
3. The notation 'Break K' provides pipeline status information after executing the instruction to which 'Break K' applies. The value of K is interpreted as follows:
   - K = 0 The Address Unit was stopped by the instruction but the pipeline was not flushed. The Address Unit can start processing the next instruction immediately.
   - K > 0 The pipeline was flushed by the instruction. The Address Unit must wait for K cycles before it can start processing the next instruction.
   - K < 0 The Address Unit was stopped at the beginning of the instruction but it was restarted |K| cycles before the end of it. The Address Unit can start processing the next instruction |K| cycles before the end of the instruction to which 'Break K' applies.
4. Some instructions must wait for pending writes to complete before being able to execute. The number of cycles that these instructions must wait for, is between 6 and 7 for the first operand in the write buffer and 2 for the second operand, if any.
5. The CBITIi and SBITIi instructions will execute a RMW access after waiting for pending writes. The extra time required for the RMW access is only 3 cycles since the read portion is overlapped with the time in the Execution Unit.
6. The keyword defined for the Bcond instruction have the following meaning:
   - BTPC Branch Taken, Predicted Correctly
   - BTPI Branch Taken, Predicted Incorrectly
   - BNTPC Branch Not Taken, Predicted Correctly
   - BNTPI Branch Not Taken, Predicted Incorrectly

D.5.3 T_{eff} Evaluation

The T_e portion of the effective execution time for a certain instruction in an instruction sequence is obtained by performing the following steps:

1. Label the current and previous instruction in the sequence with n and n−1 respectively.
2. Obtain from the tables the values of T_{eu} and T_{au} for instruction n and T_{eu} for instruction n−1.
3. For floating-point instructions, obtain the values of T_{tcs} and T_{tsc}.
4. Use the following formula to determine the execution time T_e.

   $T_e = T_{dpf}(n) + \text{func}\ (T_{au}(n), T_{eu}(n-1), T_{flt}(n-1), \text{Break}\ (n-1)) + T_{eu}(n) + T_{flt}(n)$

   T_{dpf} is the delay incurred before an instruction can begin execution. It must be considered only when the floating-point pipelined mode is enabled.

   For a non-floating-point instruction, it represents the time needed to complete all the instructions in the FIFO. For a floating-point instruction, it is only relevant if the FIFO is full, and represents the time to complete the first instruction in the FIFO.

   func provides the amount of processing time in the Address Unit that cannot be hidden. Its definition is given below.

func = 0	if $T_{au}(n) \le (T_{eu}(n-1) + T_{flt}(n-1))$ AND NOT Break (n−1)
$T_{au}(n) - T_{eu}(n-1)$	if $T_{au}(n) > (T_{eu}(n-1) + T_{flt}(n-1))$ AND NOT Break (n−1)
$T_{au}(n) + K$	if $(T_{au}(n) + K) > 0$ AND Break (n−1)
0	if $(T_{au}(n) + K) \le 0$ AND Break (n−1)

   K is the value associated with Break (n−1).

T_{flt} only applies to floating-point instructions and is always 0 for other instructions. It is evaluated as follows:

if pipelined mode is disabled, then

$T_{flt} = t_{tcs} + T_{tsc} + T_{fpu}$

else

Tflt = 0 if group A instruction.

max (T_{prv}, T_{tcs}) + T_{tsc} if group B instruction.

T_{fpu} is the execution time in the Floating-Point Unit. T_{prv} is the time needed by the CPU and FPU to complete all the floating-point instructions in the FIFO.

5. Calculate the total execution time T_{eff} by using the following formula:

$$T_{eff} = T_e + T_d + T_s$$

Where T_d and T_s are dependent on the instruction sequence, and can be obtained using the information provided in Section D.4.

D.5.4 Instruction Timing Example

This section presents a simple instruction timing example for a procedure that recursively evaluates the Fibonacci function. In this example there are no data dependencies or storage buffer misses; only the basic instruction execution times in the pipeline, control dependencies, and instruction alignment are considered.

The following is the source of the procedure in C.

```
unsigned fib(x)
int   x;
{
      if (x > 2)
              return (fib(x-1) + fib(x-2));
      else
              return(1);
}
```

The assembly code for the procedure with comments indicating the execution time is shown below. The procedure requires 26 cycles to execute when the actual parameter is less than or equal to 2 (branch taken) and 99 cycles when the actual parameter is equal to 3 (recursive calls).

```
_fib:   movd    r3,tos     ; 2 cycles
        movd    r4,tos     ; 2 cycles
        movd    r1,r3      ; 2 cycles
        cmpqd   $(2),r3    ; 2 cycles
        bge     .L1        ; 2 cycles,  Break 4 If Branch Taken
        movd    r3,r1      ; 2 cycles
        addqd   $(-2),r1   ; 2 cycles
        bsr     _fib       ; 3 cycles
        movd    r0,r4      ; 4 cycles + 4 Cycles due to RET
        movd    r3,r1      ; 2 cycles
        addqd   $(-1),r1   ; 2 cycles
        bsr     _fib       ; 3 cycles
        addd    r4,r0      ; 4 cycles + 1 cycle alignment + 4 cycles due to RET
        movd    tos,r4     ; 2 cycles
        movd    tos,r3     ; 2 cycles
        ret     $(0)       ; 4 cycles, break 4
        .align 4
_L1:    movqd   $(1),r0    ; 4 cycles + 4 cycles due to BGE
        movd    tos,r4     ; 2 cycles
        movd    tos,r3     ; 2 cycles
        ret     $(0)       ; 4 cycles, Break 4
```

Appendix D. Instruction Execution Times (Continued)

D.5.5 Execution Timing Tables

The following tables provide the execution timing information for all the NS32532 instructions. The table for the floating-point instructions provides only the CPU portion of the total execution time. The FPU execution times can be found in the NS32381 and NS32580 datasheets.

D.5.5.1 Basic and Memory Management Instructions

Mnemonic	T_{eu}	T_{au}	Notes
ABSi	5	2 + T_{ad}	
ACBi	5	2 + T_{ad}	If incorrect prediction then Break 1
ADDi	2	2 + T_{ad}	
ADDCi	2	2 + T_{ad}	
ADDPi	9	2 + T_{ad}	
ADDQi	2	2 + T_{ad}	
ADDR	2	4 + T_{ad}	
ADJSPi	5 3	2 + T_{ad} 2 + T_{ad}	i = B, W Break 0 i = D Break 0
ANDi	2	2 + T_{ad}	
ASHi	9	2 + T_{ad}	
B_{COND}	2 ⟶ 3 2 2 2	2 2 2 2	BTPC BTPI Break 2 BNTPC BNTPI Break 2 (see Note 5 in Section D.5.2)
BICi	2	2 + T_{ad}	
BICPSRi	6	2 + T_{ad}	Wait for pending writes. Break 5
BISPSRi	6	2 + T_{ad}	Wait for pending writes. Break 5
BPT	30 21	2 2	Modular Direct Break 5
BR	2 ⟶ 3	2	
BSR	2 ⟶ 3	3 + T_{ad}	
CASEi	7	2 + T_{ad}	Break 5
CBITi	10 14	2 2 + T_{ad}	<R> <M> Break 0
CBITIi	18	2 + T_{ad}	<M> Wait for pending writes. Execute interlocked RMW access. Break 5

Mnemonic	T_{eu}	T_{au}	Notes
CHECKi	10	2 + T_{ad}	Break −3. If SRC is out of bounds and the V bit in the PSR is set, then add trap time.
CINV	10	2 + T_{ad}	Wait for pending writes. Break 5
CMPi	2	2 + T_{ad}	
CMPMi	6 + 8 * n		n = number of elements. Break 0
CMPQi	2	2 + T_{ad}	
CMPSi	7 + 13 * n	2 + T_{ad}	n = number of elements. Break 0
CMPST	6 + 20 * n	2 + T_{ad}	n = number of elements. Break 0
COMi	2	2 + T_{ad}	
CVTP	5	4 + T_{ad}	
CXP	17	13	Break 5
CXPD	21	11 + T_{ad}	Break 5
DEIi	28 + 4 * i	5 + T_{ad}	i = 0/4/12 for B/W/D. Break 0
DIA	3	2	Break 5
DIVi	(30 ⟶ 40) + 4 * i	2 + T_{ad}	i = 0/4/12 for B/W/D
ENTER	15 + 2 * n	3	n = number of registers saved. Break 0
EXIT	8 + 2 * n	2	n = number of registers restored
EXTi	12 13	8 8 + T_{ad}	<R> <M> Break −3
EXSi	11 14	6 6 + T_{ad}	<R> <M> Break −3

Appendix D. Instruction Execution Times (Continued)

D.5.5.1 Basic and Memory Management Instructions (Continued)

Mnemonic	T_{eu}	T_{au}	Notes
FFSi	11 + 3 * i	2 + T_{ad}	i = number of bytes
FLAG	4 32 21	2 2 2	No trap Trap, Modular Trap, Direct If trap then: {wait for pending writes; Break 5}
IBITi	10 14	2 2 + T_{ad}	<R> <M> Break 0
INDEXi	43	5 + T_{ad}	
INSi	15 18	8 8 + T_{ad}	<R> <M>
INSSi	14 19	6 6 + T_{ad}	<R> <M> Break 0
JSR	3	9 + T_{ad}	Break 5
JUMP	3	4 + T_{ad}	Break 5
LMR	11	2 + T_{ad}	Wait for pending writes. Break 5
LPRi	6 5 7	2 + T_{ad} 2 + T_{ad} 2 + T_{ad}	CPU Reg = FP, SP, USP, SP, MOD. Break 0 CPU Reg = CFG, INTBASE, DSR, BPC, UPSR. Wait for pending writes. Break 5 CPU Reg = DCR, PSR CAR. Wait for pending writes. Break 5
LSHi	3	2 + T_{ad}	
MEIi	13 + 2 * i	5 + T_{ad}	i = 0/4/12 for B/W/D. Break 0
MODi	(34 → 49) + 4 * i	2 + T_{ad}	i = 0/4/12 for B/W/D
MOVi	2	2 + T_{ad}	
MOVMi	5 + 4 * n	2 + T_{ad}	n = number of elements. Break 0
MOVQi	2	2 + T_{ad}	
MOVSi	 12 + 4 * n 14 + 8 * n	 2 + T_{ad} 2 + T_{ad}	n = number of elements. No options. B, W and/or U Options in effect. Break 0
MOVST	16 + 9 * n	2 + T_{ad}	n = number of elements. Break 0

Mnemonic	T_{eu}	T_{au}	Notes
MOVSVi	9	2 + T_{ad}	Wait for pending writes. Break 5
MOVUSi	11	2 + T_{ad}	Wait for pending writes. Break 5
MOVXii	2	2 + T_{ad}	
MOVZii	2	2 + T_{ad}	
MULi	13 + 2 * i 24	2 + T_{ad} 2 + T_{ad}	i = 0/4/12 for B/W/D. General case. If MULD and 0 ≤ SRC ≤ 255
NEGi	2	2 + T_{ad}	
NOP	2	2	
NOTi	3	2 + T_{ad}	
ORi	2	2 + T_{ad}	
QUOi	(30 → 40) + 4 * i	2 + T_{ad}	i = 0/4/12 for B/W/D
RDVAL	10	2 + T_{ad}	Wait for pending writes. Break 5
REMi	(32 → 42) + 4 * i	2 + T_{ad}	i = 0/4/12 for B/W/D
RESTORE	7 + 2 * n	2	n = number of registers restored. Break 0
RET	4	3	Break 4
RETI	19 13 29 22	5 5 5 5	Noncascaded, Modular Noncascaded, Direct Cascaded, Modular Cascaded, Direct Wait for pending writes. Break 5
RETT	14 8	5 5	Modular Direct Wait for pending writes. Break 5
ROTi	7	2 + T_{ad}	
RXP	8	5	Break 5
SCONDi	3	2 + T_{ad}	
SAVE	8 + 2 * n	2	n = number of registers. Break 0
SBITi	10 14	2 2 + T_{ad}	<R> <M> Break 0

D.5.5.1 Basic and Memory Management Instructions (Continued)

Mnemonic	T_{eu}	T_{au}	Notes
SBITIi	10 18	2 2 + T_{ad}	<R> <M> Wait for pending writes. Execute interlocked RMW access. Break 5
SETCFG	6	2	Break 5
SKPSi	8 + 6 * n	2 + T_{ad}	n = number of elements. Break 0
SKPST	6 + 20 * n	2 + T_{ad}	n = number of elements. Break 0
SMR	7	2 + T_{ad}	Wait for pending writes. Break 5

Mnemonic	T_{eu}	T_{au}	Notes
SPRi	5 3	2 + T_{ad} 2 + T_{ad}	CPU Reg = PSR, CAR CPU Reg = all others
SUBi	2	2 + T_{ad}	
SUBCi	2	2 + T_{ad}	
SUBPi	6	2 + T_{ad}	
SVC	32 21	2 2	Modular Direct Wait for pending writes. Break 5
TBITi	8 11	2 2 + T_{ad}	<R> <M> Break 0
WAIT	3	2	Wait for pending writes. Wait for interrupt
WRVAL	10	2 + T_{ad}	Wait for pending writes. Break 5
XORi	2	2 + T_{ad}	

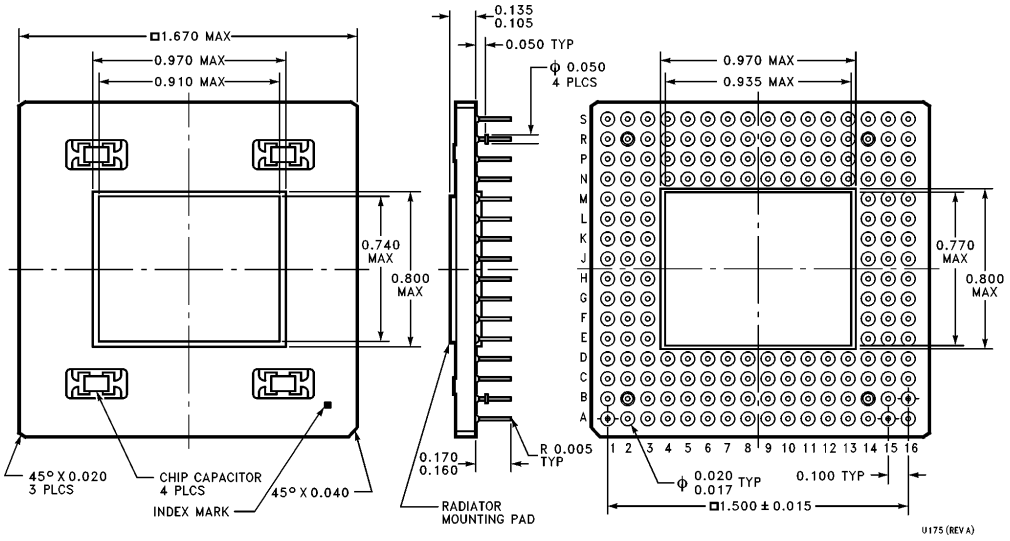
Appendix D. Instruction Execution Times (Continued)

D.5.5.2 Floating-Point Instructions, CPU Portion

Mnemonic	T_{eu}	T_{au}	T_{tcs}	T_{tsc}	Group	Notes
MOVf, NEGf, ABSf, SQRTf, LOGBf	2 4 + 3 * I 6 + 3 * I 6 + 3 * I 11 + 4 * I 13 + 7 * I	2 + T_{anp} 2 + T_{anp} + T_{ad} 2 + T_{anp} 2 + T_{anp} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad}	2 2 2 2 2 2	1 1 1 1 3 + 2 * I 3 + 2 * I	A A B B B B	<FF> <MF> <IF> <TF> <FM> Break − (1 + I) <MM>, <IM> Break − (1 + I)
ADDf, SUBf, MULf, DIVf, SCALBf	2 4 + 3 * I 6 + 3 * I 6 + 3 * I 17 + 7 * I 19 + 10 * I	2 + T_{anp} 2 + T_{anp} 2 + T_{anp} 2 + T_{anp} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad}	2 2 2 2 2 2	1 1 1 1 3 + 2 * I 3 + 2 * I	A A B B B B	<FF> <MF> <IF> <TF> <FM> Break − (1 + I) <MM>, <IM> Break − (1 + I)
ROUNDfi, TRUNCfi, FLOORfi	11 11 + 4 * I 13 13 + 7 * I	2 + T_{anp} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad}	2 2 2 2	3 + 2 * I 3 + 2 * I 3 + 2 * I 3 + 2 * I	B B B B	<FR> Break − 1 <FM> Break − (1 + I) <MR>, <IR> Break − 1 <MM>, <IM> Break − (1 + I)
CMPf	18 20 + 3 * I 23 + 3 * I 25 + 6 * I	2 + T_{anp} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad}	2 2 2 2		B B B B	<FF> <MF> <FM> <MM>, <IM>, <MI>, <II> Break 3
POLYf, DOTf, MACf	2 4 + 3 * I 6 + 3 * I 11 + 4 * I 13 + 7 * I	2 + T_{anp} 2 + T_{anp} + T_{ad} 2 + T_{anp} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad}	2 2 2 2 2	1 1 1 1 1	A A B A B	<FF> <MF> <IF>, <TF> <FM> Break − (1 + I) <MM>, <MI>, <IM>, <II> Break − (1 + I)
MOVif	6 13 6 + 3 * I 13 + 7 * I	2 + T_{anp} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad}	2 2 2 2	1 1 	B B B B	<RF> <RM> Break − 1 <MF>, <IF>, <TF> <MM>, <IM> Break − (1 + I)
LFSR	6 6 + 3 * I 6 + 3 * I 6 + 3 * I	2 + T_{anp} 2 + T_{anp} + T_{ad} 2 + T_{anp} 2 + T_{anp}	2 2 2 2	1 1 1 1	B B B B	<R> <M> <I> <T>
SFSR	11	2 + T_{anp} + T_{ad}	2	3	B	Break − 1
MOVFL	4 6 15 17	2 + T_{anp} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad}	2 2 2 2	1 1 	B B B B	<FF> <MF>, <IF>, <TF> <FM> Break 0 <MM>, <IM> Break 0
MOVLF	4 9 15 20	2 + T_{anp} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad} 2 + T_{anp} + T_{ad}	2 2 2 2	1 1 	B B B B	<FF> <MF>, <IF>, <TF> <FM> Break 0 <MM>, <IM> Break 0

Physical Dimensions inches (millimeters)

Lit. # 114272



Hermetic Pin Grid Array (U)
Order Number NS32532-20, NS32532-25 or NS32532-30
NS Package Number U175A

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
1111 West Bardin Road
Arlington, TX 76017
Tel: 1(800) 272-9959
Fax: 1(800) 737-7018

National Semiconductor Europe
Fax: (+49) 0-180-530 85 86
Email: cnjwge@tevm2.nsc.com
Deutsch Tel: (+49) 0-180-530 85 85
English Tel: (+49) 0-180-532 78 32
Français Tel: (+49) 0-180-532 93 58
Italiano Tel: (+49) 0-180-534 16 80

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductor Japan Ltd.
Tel: 81-043-299-2309
Fax: 81-043-299-2408

National does not assume any responsibility for use of any circuitry described, no circuit patent licenses are implied and National reserves the right at any time without notice to change said circuitry and specifications.