Chapter 6

EXCEPTION PROCESSING

Exceptions processed in an NS16000-based system are of three general types.

A Reset is performed whenever an RST signal is received by the processor. It immediately terminates all other processing and re-initializes the processor state.

Interrupt service is performed by an NS16000 CPU upon receipt of an interrupt request on either of two input pins:

INT, on which maskable interrupts are requested, and

NMI, on which non-maskable interrupts are requested.

Traps comprise a group of special interrupts which are triggered as a result either of exceptional conditions encountered in executing an instruction (e.g., attempted division by zero) or of specific instructions whose purpose is to cause a trap to occur (e.g., the Supervisor Call instruction).

6.1 Reset

Resetting an NS16000-based system has the following effects:

1. All implemented bits of the PC, PSR and FSR registers are cleared.

2. In the MSR, the NT, FT, BEN, TS, TU and ERC fields are cleared.

All other register contents are undefined. Program execution begins at address zero. (In memory-managed systems, this is physical address zero.)

6.2 General Interrupt/Trap Sequence

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

1. Adjustment of Registers.
   Depending on the source of the interrupt or trap, the CPU may update 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.

2. Saving Processor Status.
   The PSR copy is pushed onto the Interrupt Stack as a 16-bit value.

3. Vector Acquisition.
   A vector number is either read from a source external to the CPU or is supplied internally.

4. Service Call.
   The vector number is used as an index into the Interrupt Dispatch Table, whose base address is taken from the CPU Interrupt Base (INTBASE) Register. See Figure 6-1. A 32-bit external procedure descriptor is read from this table entry, and an external procedure call is performed using it. The MOD Register (16 bits) and Program Counter (32 bits) are pushed onto the Interrupt Stack as part of this step.

This process is illustrated in Figure 6-2.

Full sequences of events in processing interrupts and traps may be found as follows:

Maskable and Non-Maskable Interrupts:	Section 6.8.1
Abort Trap:	Section 6.8.4
Trace Trap:	Section 6.8.3
Other Traps:	Section 6.8.2

6.3 Interrupt/Trap Return

To return control to an interrupted program, one of two instructions is used. The RETT (Return from Trap) instruction restores the PSR, MOD, PC and SB registers to their previous contents and, since traps are often used deliberately as a call mechanism for Supervisor Mode procedures, it also discards a specified number of bytes from the interrupted program's stack as surplus parameter space. RETT (Return from Trap) is used to return from any trap or interrupt except Maskable interrupts in Vectored mode (Section 6.4). For this, the RETI (Return from Interrupt) instruction is used, which has the additional function of informing any external Interrupt Control Units that interrupt service has completed.

6.4 Maskable Interrupts

Maskable interrupt requests are received on a CPU pin called INT in the current implementation. 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 INT, NMI or Abort request, and is restored to its original setting upon return from the interrupt service routine via the RETT or RETI instruction.

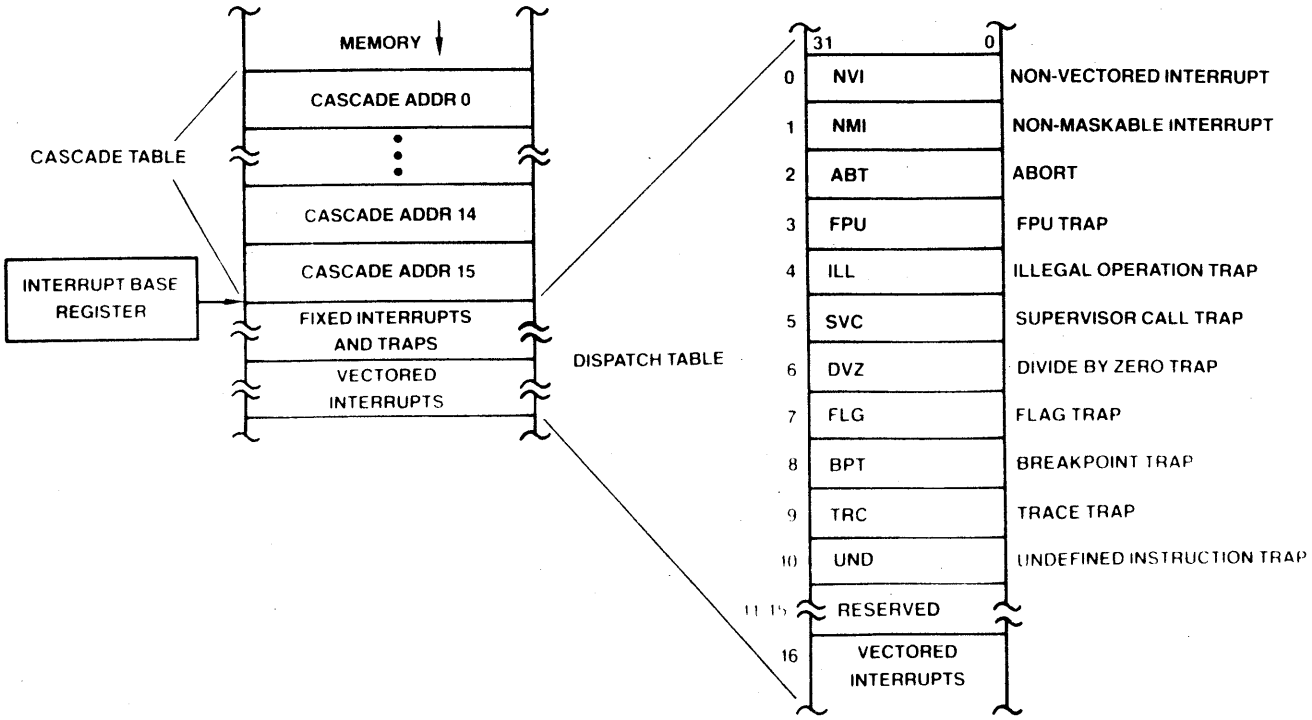

Figure 6-1 Interrupt Dispatch and Cascade Tables

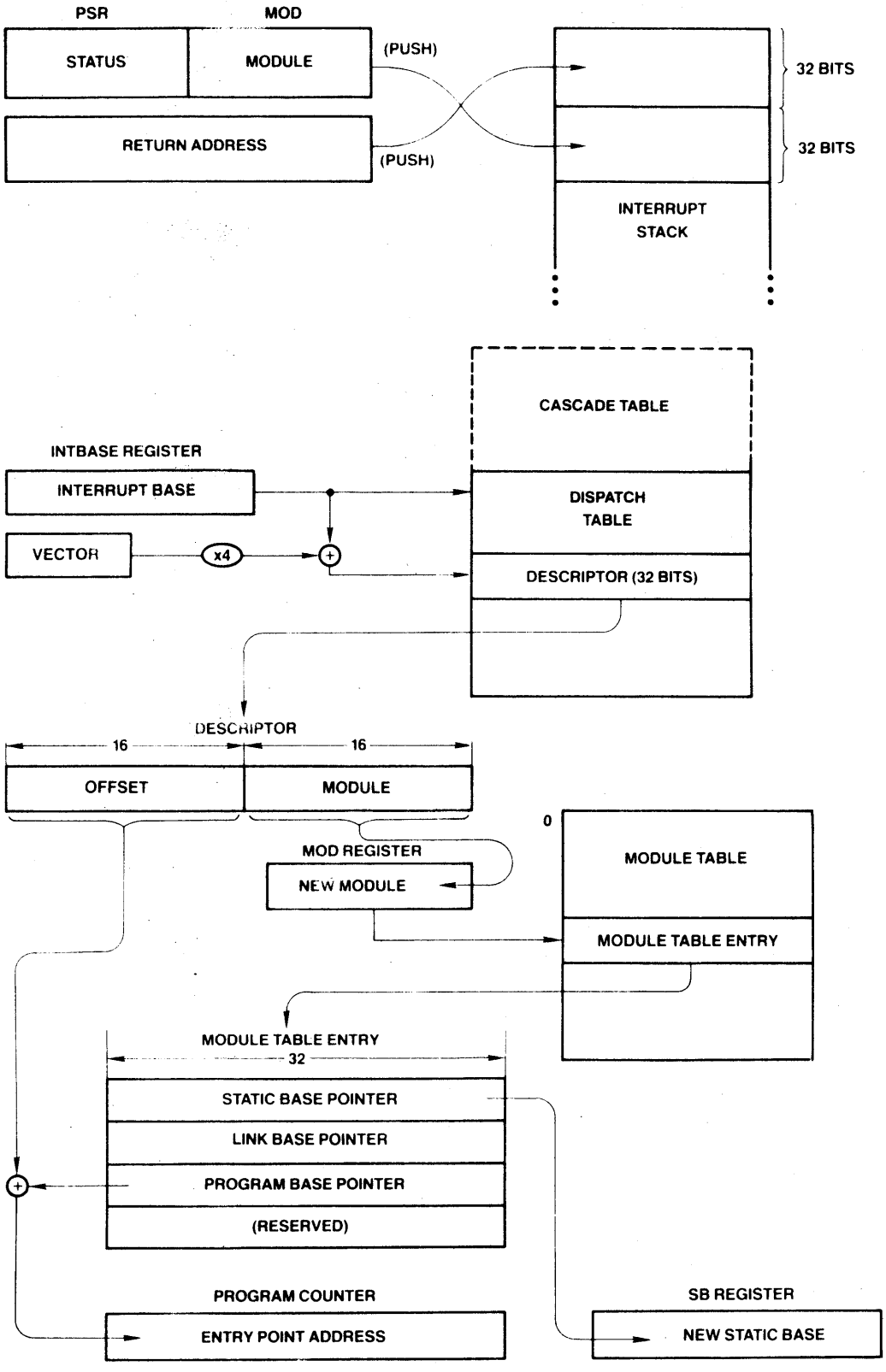

Figure 6-2 Interrupt/ Trap Service Routine Calling Sequence

Maskable interrupt service requested on the INT pin may be performed according to one of two interrupt modes. The interrupt mode is selected via the SETCFG instruction as either Non-Vectored (CFG register bit I = 0) or Vectored (CFG register bit I = 1). The RETT instruction must be used to return from maskable interrupts in Non-Vectored mode, and the RETI instruction must be used to return from maskable interrupts in Vectored mode.

6.4.1 Non-Vectored Mode

In Non-Vectored mode, an interrupt request on the INT pin causes the CPU to read a byte from address 00FFFE00 (Hex), but it ignores any value supplied and uses instead a default vector of zero. This mode is useful for small systems in which hardware interrupt prioritization is unnecessary.

6.4.2 Vectored Mode: Non-Cascaded Case

In Vectored mode, the CPU uses an NS16202 Interrupt Control Unit (ICU) to prioritize up to 16 interrupt requests (see Figure 6-3). Upon receipt of an interrupt request on the INT pin, the CPU reads a one-byte vector number from address 00FFFE00, asserting a status line to the ICU which indicates Interrupt Acknowledge as the reason for this access. This vector is then used as an index into the Interrupt Dispatch Table in order to find the external procedure descriptor (Section 2.7.3) for the proper interrupt service procedure. The service procedure eventually returns via the Return from Interrupt (RETI) instruction, which performs another one-byte read from address 00FFFE00, this time setting the status line to indicate End of Interrupt, informing the ICU that it may re-prioritize any interrupt requests still pending. The ICU provides the same vector number which was presented previously when the interrupt occurred, and the CPU uses this to determine whether it needs also to inform a cascaded ICU of the end of the service procedure (see Section 6.4.3).

In a system with only one ICU (16 levels of interrupt), the vectors provided must be in the range 0 through 127; that is, they must be positive numbers in eight bits. By providing a negative vector number, an ICU flags the interrupt source as being a cascaded ICU (see Section 6.4.3).

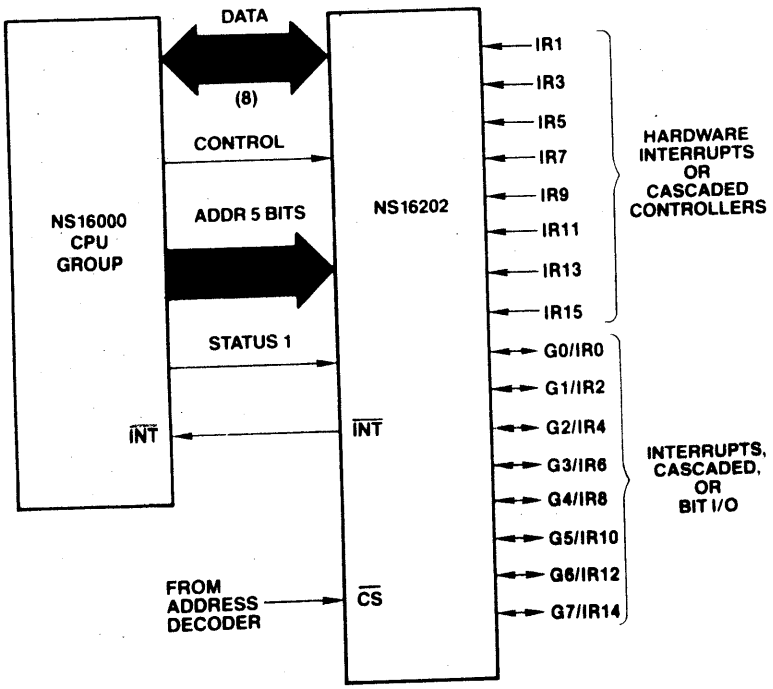

Figure 6-3 Interrupt Control Unit Connections (16 Levels)

6.4.3 Vectored Mode: Cascaded Case

In order to allow up to 256 levels of interrupt, provision is made both in the CPU and in the NS16202 Interrupt Control Unit (ICU) to transparently support cascading. Figure 6-4 shows a typical cascaded configuration. 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 INT pin.

In a system which uses cascading, two additional initializations must be performed:

1. For each cascaded ICU in the system, the master ICU must be informed of the line number (0 to 15) on which it receives the cascaded requests.

2. A Cascade Table must be established in memory. The Cascade Table is located in a negative direction from the location indicated by the Interrupt Base (INTBASE) register. Its entries contain the 32-bit addresses of the Vector Registers of each of up to 16 cascaded ICUs.

Figure 6-1 illustrates the position of the Cascade Table. To find the Cascade Table entry for any given 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 for that ICU.

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. The CPU reads a byte from the address given by the Cascade Address, which it uses as the final vector number. This vector number is interpreted by the CPU as an unsigned integer, and can therefore be in the range 0 through 255.

In returning from a cascaded interrupt, the service procedure executes the Return from Interrupt (RETI) instruction, as it would for any interrupt in Vectored mode. When the CPU performs the End of Interrupt read cycle, 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 read cycle, informing the cascaded ICU of the completion of the service routines. The byte read from the cascaded ICU is discarded.

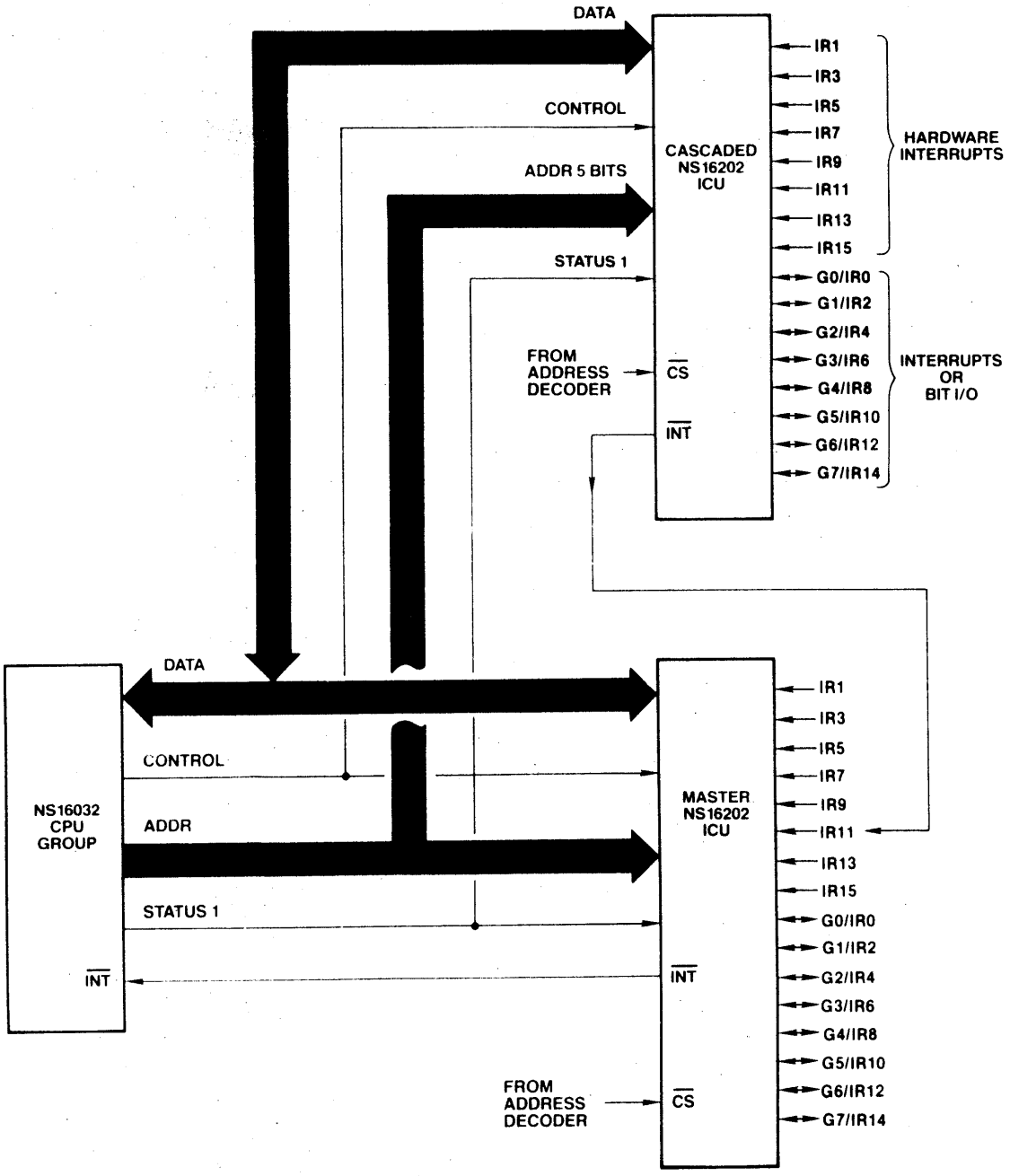

Figure 6-4 Cascaded Interrupt Control Unit Connections

6.5 Non-Maskable Interrupt

The Non-Maskable Interrupt is triggered whenever a falling edge is detected on a CPU pin called NMI. The CPU reads a byte from address 00FFFF00 (Hex) when processing of this interrupt actually begins. Note that this address differs from the address read for Maskable interrupts. The vector value used for the Non-Maskable interrupt is 1, regardless of the value read.

The service procedure returns from the Non-Maskable interrupt using the Return from Trap (RETT) instruction. No data transfers occur for interrupt control.

For the full sequence used in processing the Non-Maskable Interrupt, see Section 6.8.1.

6.6 Traps

A trap is an internally-generated interrupt request caused as a direct and immediate result of the execution of an instruciton. Traps occurring in an NS16000-based system are:

Trap (ABT): An exceptional condition was detected by memory management.

Trap (FPU): An exceptional condition was detected during the execution of a Floating-Point instruction. In systems incorporating a Custom Slave Processor, this trap can also be caused by an exceptional condition arising there. See Section 3.3 for the conditions which cause this trap.

Trap (ILL): Illegal operation. A privileged operation was attempted while the CPU was in User Mode (PSR U bit = 1). See Section 2.8 for the conditions which cause this trap.

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 instead for Floating-Point division by zero.) See Section 3.1 for further details.

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. See below.

Trap (UND): An undefined instruction was encountered.

The Return Address pushed by any trap except TRC is the address of the first byte of the instruction during which the trap occurred. Traps do not disable interrupts, as they are not associated with external events.

A special case is the Trace trap, Trap (TRC), which is enabled by setting the T bit in the Processor Status Register (PSR). At the beginning of each instruction, the T bit is copied into the PSR P (Trace "Pending") bit. If the P bit and T bit are both set at the end of an instruction, the Trace Trap is activated. If any other trap or interrupt request is also pending, 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, ensuring that one and only one Trace Trap will occur per instruction, and also ensuring that the Return Address pushed during a Trace Trap is always the address of the next instruction to be traced.

6.7 Prioritization

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

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

Traps at priority level 1 do not occur simultaneously. The first trap occurring is the trap which is serviced.

6.8 Interrupt/Trap Sequences: Detailed Flow

Figure 6-5 defines a common sequence called "Service", which is followed by the CPU in handling all interrupts and traps. Upon detecting any interrupt request or trap condition, the CPU first performs a sequence dependent upon the type of interrupt or trap. This sequence will include pushing the Processor Status Register and establishing a Vector and a Return Address. The CPU then performs the Service sequence.

For the sequence followed in processing either Maskable or Non-maskable interrupts, see Section 6.8.1. For the Abort trap, see Section 6.8.4. For the Trace Trap, see Section 6.8.3 and for all other traps see Section 6.8.2.

6.8.1 Maskable/Non-Maskable Interrupt Sequence

This sequence is performed by the CPU when the NMI pin receives a falling edge, or the INT pin becomes active with the PSR I bit set. The interrupt sequence begins either at the next instruction boundary or, in a String instruction, at the next interruptible point during its execution.

1. If a String instruction is being interrupted and has 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.

Service (Vector, Return Address):
1. Push the MOD Register onto the Interrupt Stack as a 16-bit value. (The PSR has already been pushed as a 16-bit value.) 2. Push the Return Address onto the Interrupt Stack as a 32-bit value. 3. Read the 32-bit external procedure descriptor from the Interrupt Dispatch Table: address is Vector*4 + INTBASE. 4. Move the Module field of the descriptor into the MOD Register. 5. Read the new Static Base pointer from the memory address contained in MOD, placing it into the SB Register. 6. Read the Program Base pointer from memory address MOD+8, and add to it the Offset field from the descriptor, placing the result in the Program Counter.

Figure 6-5 Common Interrupt/Trap Service Sequence

2. Copy the Processor Status Register (PSR) into a temporary register, then clear PSR bits S, U, T, P and I.

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

4. If the interrupt is Non-Maskable:

   a. Read a byte from address 00FFFF00 (Hex). Discard the byte read.

   b. Set "Vector" to 1.

   c. Go to Step 9.

5. If the interrupt is Non-Vectored:

   a. Read a byte from address 00FFFE00 (Hex). Discard the byte read.

   b. Set "Vector" to 0.

   c. Go to Step 9.

6. Here the interrupt is being serviced in Vectored mode. Read "Byte" from address 00FFFE00 (Hex).

7. If "Byte" $\geq$ 0, then set "Vector" to "Byte" and go to Step 9.

8. If "Byte" is in the range -16 through -1, then the interrupt source is a cascaded ICU. (More negative values are reserved for future use.) Perform the following:

   a. Read the 32-bit Cascade Address from memory location INTBASE + 4*Byte.

   b. Read "Vector" from the address given by the Cascade Address.

9. Perform Service (Vector, Return Address), Figure 6-5.

6.8.2 Trap Sequence: All Except Trace and Abort

1. Restore the currently selected Stack Pointer to its original contents at the beginning of the trapped instruction.

2. Set "Vector" to the value corresponding to the trap type.

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

3. Copy the Processor Status Register (PSR) into a temporary register, then clear PSR bits S, U, P and T.

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

5. Set "Return Address" to the address of the first byte of the trapped instruction.

6. Perform Service (Vector, Return Address), Figure 6-5

6.8.3 Trace Trap Sequence

1. In the Processor Status Register (PSR), clear the P bit.

2. Copy the PSR into a temporary register, then clear PSR bits S, U and T.

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

4. Set "Vector" to 9.

5. Set "Return Address" to the address of the next instruction.

6. Perform Service (Vector, Return Address), Figure 6-5.

6.8.4 Abort Trap Sequence

1. Restore the currently selected Stack Pointer to its original contents at the beginning of the aborted instruction.

2. Clear the PSR P bit.

3. Copy the PSR into a temporary register, then clear PSR bits S, U, T and I.

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

5. Set "Vector" to 2.

6. Set "Return Address" to the address of the first byte of the aborted instruction.

7. Perform Service (Vector, Return Address), Figure 6-5.

APPENDIX A

INSTRUCTION SET LISTED BY FUNCTIONAL GROUPS

Instruction	Mnemonic Forms	Index
INTEGER		
Arithmetic		
Add	ADDB, ADDW, ADDD	ADDi
Add Quick	ADDQB, ADDQW, ADDQD	ADDQi
Add with Carry	ADDCB, ADDCW, ADDCD	ADDCi
Subtract	SUBB, SUBW, SUBD	SUBi
Subtract with Carry [Borrow]	SUBCB, SUBCW, SUBCD	SUBCi
Negate	NEGB, NEGW, NEGD	NEGi
Absolute Value	ABSB, ABSW, ABSD	ABSi
Multiply	MULB, MULW, MULD	MULi
Multiply Extended Integer	MEIB, MEIW, MEID	MEIi
Divide	DIVB, DIVW, DIVD	DIVi
Modulus	MODB, MODW, MODD	MODi
Quotient	QUOB, QUOW, QUOD	QUOi
Remainder	REMB, REMW, REMD	REMi
Divide Extended Integer	DEIB, DEIW, DEID	DEIi
Movement and Conversion		
Move	MOVB, MOVW, MOVD	MOVi
Move Quick	MOVQB, MOVQW, MOVQD	MOVQi
Move with Sign-Extension	MOVXBD,MOVXWD,MOVXBW	MOVXii
Move with Zero-Extension	MOVZBD,MOVZWD,MOVZBW	MOVZii
Comparison		
Compare	CMPB, CMPW, CMPD	CMPi
Compare Quick	CMPQB, CMPQW, CMPQD	CMPQi
PACKED DECIMAL		
Add Packed Decimal	ADDPB, ADDPW, ADDPD	ADDPi
Subtract Packed Decimal	SUBPB, SUBPW, SUBPD	SUBPi

Instruction	Mnemonic Forms	Index
FLOATING POINT		
Add Floating	ADDF, ADDL	ADDf
Subtract Floating	SUBF, SUBL	SUBf
Multiply Floating	MULF, MULL	MULf
Divide Floating	DIVF, DIVL	DIVf
Negate Floating	NEGF, NEGL	NEGf
Absolute Value Floating	ABSF, ABSL	ABSf
Compare Floating	CMPF, CMPL	CMPf
Move Floating	MOVF, MOVL	MOVf
Move Long Floating to Floating	MOVLF	
Move Floating to Long Floating	MOVFL	MOVFL
Move Integer to Floating	MOVBF, MOVWF, MOVDF, MOVBL, MOVWL, MOVDL	MOVif
Round Floating to Integer	ROUNDFB, ROUNDFW, ROUNDFD, ROUNDLB, ROUNDLW, ROUNDLD	ROUNDfi
Truncate Floating to Integer	TRUNCFB, TRUNCFW, TRUNCFD, TRUNCLB, TRUNCLW, TRUNCLD	TRUNCfi
Floor Floating to Integer	FLOORFB, FLOORFW, FLOORFD, FLOORLB, FLOORLW, FLOORLD	FLOORfi
Load FSR	LFSR	LFSR
Store FSR	SFSR	SFSR
LOGICAL		
Arithmetic		
Logical AND	ANDB, ANDW, ANDD	ANDi
Logical OR	ORB, ORW, ORD	ORi
Bit Clear	BICB, BICW, BICD	BICi
Exclusive OR	XORB, XORW, XORD	XORi
Complement	COMB, COMW, COMD	COMi
Shift		
Arithmetic Shift	ASHB, ASHW, ASHD	ASHi
Logical Shift	LSHB, LSHW, LSHD	LSHi
Rotate	ROTB, ROTW, ROTD	ROTi
Boolean		
Complement Boolean	NOTB, NOTW, NOTD	NOTi
Save Condition as Boolean	ScondB, ScondW, ScondD	Scondi

Instruction	Mnemonic Forms	Index
BIT		
Test Bit	TBITB, TBITW, TBITD	TBITi
Set Bit	SBITB, SBITW, SBITD, SBITIB, SBITIW, SBITID	SBITi, SBITIi
Clear Bit	CBITB, CBITW, CBITD, CBITIB, CBITIW, CBITID	CBITi, CBITIi
Invert Bit	IBITB, IBITW, IBITD	IBITi
Find First Set Bit	FFSB, FFSW, FFSD	FFSi
Convert to Bit Pointer	CVTP	CVTP
BIT FIELD		
Extract Field	EXTB, EXTW, EXTD	EXTi
Extract Field Short	EXTSB, EXTSW, EXTSD	EXTSi
Insert Field	INSB, INSW, INSD	INSi
Insert Field Short	INSSB, INSSW, INSSD	INSSi
STRING		
Move String	MOVSB, MOVSW, MOVSD	MOVSi
Move String, Translating	MOVST	MOVST
Compare Strings	CMPSB, CMPSW, CMPSD	CMPSi
Compare Strings, Translating	CMPST	CMPST
Skip String	SKPSB, SKPSW, SKPSD	SKPSi
Skip String, Translating	SKPST	SKPST
BLOCK		
Move Multiple	MOVMB, MOVMW, MOVMD	MOVMi
Compare Multiple	CMPMB, CMPMW, CMPMD	CMPMi
ARRAY		
Bounds Check	CHECKB, CHECKW, CHECKD	CHECKi
Calculate Index	INDEXB, INDEXW, INDEXD	INDEXi

Instruction	Mnemonic Forms	Index
PROCESSOR CONTROL		
Branches		
Jump	JUMP	JUMP
Conditional Branch	Bcond	Bcond
Unconditional Branch	BR	BR
Case Branch (Multiway)	CASEB, CASEW, CASED	CASEi
Add, Compare and Branch	ACBB, ACBW, ACBD	ACBi
Local Procedure Calls/Returns		
Jump to Subroutine	JSR	JSR
Branch to Subroutine	BSR	BSR
Return from Subroutine	RET	RET
External Procedure Calls/Returns		
Call External Procedure	CXP	CXP
Call External Procedure with Descriptor	CXPD	CXPD
Return from External Procedure	RXP	RXP
Explicit Trap Instructions		
Breakpoint Trap	BPT	BPT
Trap on Flag (conditional)	FLAG	FLAG
Supervisor Call Trap	SVC	SVC
Trap/Interrupt Returns		
Return from Trap*	RETT	RETT
Return from Interrupt*	RETI	RETI

* Privileged instruction.

Instruction	Mnemonic Forms	Index
PROCESSOR SERVICE		
Effective Address		
Calculate Effective Address	ADDR	ADDR
Load External Procedure Descriptor (alternate mnemonic for ADDR)	LXPD	LXPD
Context Instructions		
Save General Purpose Registers	SAVE	SAVE
Restore General Purpose Registers	RESTORE	RESTORE
Enter New Procedure Context	ENTER	ENTER
Exit Procedure Context	EXIT	EXIT
Register/Stack Manipulation		
Adjust Stack Pointer	ADJSPB, ADJSPW, ADJSPD	ADJSPi
Bit Clear in PSR*	BICPSRB, BICPSRW	BICPSRB BICPSRW
Bit Set in PSR*	BISPSRB, BISPSRW	BISPSRB BISPSRW
Load Processor Register*	LPRB, LPRW, LPRD	LPRi
Store Processor Register*	SPRB, SPRW, SPRD	SPRi
Set Configuration Register*	SETCFG	SETCFG
Miscellaneous		
No Operation	NOP	NOP
Wait for Interrupt	WAIT	WAIT
Diagnose	DIA	DIA
* Privileged, or having privileged forms.		
MEMORY MANAGEMENT		
Load Memory Management Register	LMR	LMR
Store Memory Management Register	SMR	SMR
Validate Address for Reading	RDVAL	RDVAL
Validate Address for Writing	WRVAL	WRVAL
Move Value from Supervisor to User Space	MOVSUB, MOVSUW, MOVSUD	MOVSUi
Move Value from User to Supervisor Space	MOVUSB, MOVUSW, MOVUSD	MOVUSi

APPENDIX B

NS16032 INSTRUCTION EXECUTION TIMES

B.1 Assumptions

The entire instruction, with all displacements and immediate operands, is assumed to be present in the instruction queue when needed.

Interference from instruction prefetches, which is very dependent upon the preceding instruction(s), is ignored. This assumption will tend to affect the timing estimate in an optimistic direction.

It is assumed that all memory operand transfers are completed before the next instruction begins execution. In the case of an operand of access class rmw in memory, this is pessimistic, as the Write transfer occurs in parallel with the execution of the next instruction.

It is assumed that there is no overlap between the fetch of an operand and the following sequences of microcode. This is pessimistic, as the fetch of Operand A will generally occur in parallel with the effective address calculation of Operand B, and the fetch of Operand B will occur in parallel with the execution phase of the instruction. See Section 4.3 for the definition of operands A and B.

Where possible, the values of operands are taken into consideration when they affect instruction timing, and a range of times is given. Where this is not done, the worst case is assumed.

B.2 Definitions

TEA - The time required to calculate an operand's Effective Address. For a Register or Immediate operand, this includes the fetch of that operand.

TMMU - The extra clock cycle required for translation of memory addresses if an MMU is present.

TOPB - The time needed to read or write a memory byte.

TOPW - The time needed to read or write a memory word.

TOPD - The time needed to read or write a memory double-word.

TOPi - The time needed to read or write a memory operand, where the operand size is given by the operation length of the instruction. It is always equivalent to either TOPB, TOPW or TOPD.

TCY - Internal processing overhead, in clock cycles.

L - Internal processing whose duration depends on the operation length (Section 4.1). The number of clock cycles is derived by multiplying this value by the number of bytes in the operation length.

B.3 Equations

```
TMMU -  If an MMU is present then TMMU=1
        else TMMU=0

TOPB -  If operand is in a register or is immediate then TOPB=0
        else TOPB = 3 + TMMU

TOPW -  If operand is in a register or is immediate then TOPW=0
        else if word-aligned (even address) then TOPW = 3 + TMMU
             else TOPW = 7 + 2 * TMMU

TOPD -  If operand is in a register or is immediate then TOPD=0
        else if word-aligned (even address) then TOPD = 7 + 2 * TMMU
             else TOPD = 11 + 3 * TMMU

TOPi -  If operand is in a register or is immediate then TOPi=0
        else if i=byte then TOPi = TOPB
        else if i=word then TOPi = TOPW
        else (i=double-word) then TOPi = TOPD

TCY  -  TCY = 1

L    -  If i (operation length) = byte then L = 1
        else if i = word then L = 2
        else (i = double-word) L = 4

TEA  -  If REGISTER addressing then TEA = 2
        if IMMEDIATE or ABSOLUTE addressing then TEA = 4
        if REGISTER RELATIVE or MEMORY SPACE addressing then TEA = 5
        if MEMORY RELATIVE addressing then TEA = 7 + TOPD
        if TOP OF STACK addressing then
             if access class = write then TEA = 4
             if access class = read then TEA = 2
             else TEA = 3
        if EXTERNAL addressing then TEA = 11 + 2 * TOPD
        if SCALED INDEXED addressing then TEA = TI1 + TI2

             where TI1 depends on scale factor:
                   if byte indexing TI1 = 5
                   if word indexing TI1 = 7
                   if double-word indexing TI1 = 8
                   if quad-word indexing TI1 = 10

             and   TI2 = TEA of the basemode except:
                         if basemode is REGISTER then TI2 = 5
                         if basemode is TOP OF STACK then TI2 = 4
```

B.4 Calculation of Total Execution Time (TEX)

TEX is obtained by performing the following steps:

Find the desired instruction in the table.

Calculate the values for TEA, TOPB, etc. using the numbers in the table and the equations given on the preceding page.

The result derived by adding together these values is the execution time (TEX) in clock cycles.

B.5 Notes on Table Use

Values in the TEA column indicate the number of effective addresses to be calculated. If the value in this column is less than the number of general operands in the instruction, this is because one or both operands are in registers and that instruction has an optimized form which eliminates TEA for such operands.

In the L column, multiply the entry by the operation length in bytes (1, 2 or 4).

In the TCY column, special notations sometimes appear:

n1-n2 means n1 minimum, n2 maximum.

n1%n2 means that the instruction flushes the instruction queue after n1 clock cycles and non-sequentially fetches the next instruction. The value n2, indicating the total number of clock cycles in internally executing the instruction (including n1), is not generally useful. The most accurate technique for determining such timing depends on the size and alignment of the Basic Instruction portion of the next instruction, plus Index Bytes. If this portion can be read in one memory cycle, then the execution time is n1+10 (including the memory cycle). If more memory cycles are required, the value is n1+5+4*m, where m is the number of memory cycles required.

In the Notes column, notations held within angle brackets <> indicate alternatives in the form of the instruction which affect the execution time. A table entry which is affected by the form of the instruction may have multiple values, separated by slashes, corresponding to the alternatives. The notations are:

<M>	Memory form
<R>	Register form
<MM>	Memory-to-Memory form
<RM>	Register-to-Memory form
<MR>	Memory-to-Register form
<RR>	Register-to-Register form
x	either Register or Memory

B.6 Example of Table Use

Calculate TEX for the instruction:

```
CMPW  R0,TOS
```

Operand A is in a register; Operand B is in memory. This means that we must use the table values corresponding to the <xM> case as given in the Notes column (<xM> means "anything to memory").

Only the TEA, TOPi and TCY columns have values assigned for the CMPi instruction. Therefore, they are the only ones that need to be calculated to find TEX. The blank columns are irrelevant to this instruction.

The TEA column contains 2 for the <xM> case. This means that effective address times have to be calculated for both operands. (For the <MR> case, the Register operand would have required no TEA time, therefore only the Memory operand TEA would have been necessary.) From the equations:

```
TEA (Register mode)  =  2,
TEA (Top of Stack mode, read access class)  =  2,

Total TEA = 2+2 = 4.
```

The TOPi column represents potential operand transfers to or from memory. For a Compare instruction, each operand is read once, for a total of two operand transfers.

```
TOPi (Word, Register) = 0,
TOPi (Word, TOS) = TOPW = 3  (assuming aligned, no MMU)
Total TOPi = 3
```

TCY is the time required for internal operation within the CPU. The TCY value for this case is 3.

TEX = TEA + TOPi + TCY = 4 + 3 + 3 = 10 machine cycles.

If the CPU is running at 10 MHz then a machine cycle (clock cycle) is 100 ns. Therefore, this instruction would take 10x100 ns., or 1.0 microseconds, to execute.

Table B-1 Basic and Memory Management Instructions

This table does not include the timings for Floating-Point instructions. See Section B.7 and Table B-2.

MNEMONIC	TEA	TOPB	TOPW	TOPD	TOPi	TCY	L	NOTES
ABSi	2	-	-	-	2	9/8	-	src<0 / src>=0
ACBi	1	-	-	-	2	16/15%20	-	<M>, no branch / branch
ACBi	-	-	-	-	-	18/17%22	-	<R>, no branch / branch
ADDi	2/1/0	-	-	-	3/1/0	3/4/4	-	<xM>/<MR>/<RR>
ADDCi	2/1/0	-	-	-	3/1/0	3/4/4	-	<xM>/<MR>/<RR>
ADDPi	2	-	-	-	3	16/18	-	no carry / carry
ADDQi	1/0	-	-	-	1/0	6/4	-	<M>/<R>
ADDR	2/1	-	-	1/0	-	2/3	-	<xM>/<xR>
ADJSPi	1	-	-	-	1	6	-	
ANDi	2/1/0	-	-	-	3/1/0	3/4/4	-	<xM>/<MR>/<RR>
ASHi	2	1	-	-	2	14-45	-	
Bcond	-	-	-	-	-	7/6%10	-	no branch / branch
BICi	2/1/0	-	-	-	3/1/0	3/4/4	-	<xM>/<MR>/<RR>
BICPSRB	1	1	-	-	-	18%22	-	
BICPSRW	1	-	1	-	-	30%34	-	

Table B-1 (Cont.)

MNEMONIC	TEA	TOPB	TOPW	TOPD	TOPi	TCY	L	NOTES
BISPSRB	1	1	-	-	-	18%22	-	
BISPSRW	1	-	-	-	-	30%34	-	
BPT	-	-	4	3	-	40	-	
BR	1	-	-	-	1	4%9	-	
BSR	-	-	-	1	-	6%16	-	
CASEi	1	-	-	-	1	4%9	-	
CBITi	2/1	2/0	-	-	1	15/7	-	<xM>/<xR>
CBITIi	2/1	2/0	-	-	1	15/7	-	<xM>/<xR>
CHECKi	2	-	-	-	3	7/10/11	-	high / low / ok
CMPi	2/1/0	-	-	-	2/1/0	3	-	<xM>/<MR>/<RR>
CMPMi	2	-	-	-	2*n	9*n+24	-	n = # of elements in block
CMPQi	1/0	-	-	-	1/0	3	-	<M>/<R>
CMPSi	-	-	-	-	2*n	35*n+53	-	n = # of elements, not Translated
CMPST	-	n	-	-	2*n	38*n+53	-	Translated
COMi	2	-	-	-	2	7	-	
CVTP	2	-	-	1	-	7	-	
CXP	-	-	3	4	-	16%21	-	

Table B-1 (Cont.)

MNEMONIC	TEA	TOPi	TOPW	TOPD	TOPi	TCY	L	NOTES
CXPD	1	-	3	1	-	13%18	-	
DEIi	2/1	-	-	-	5/1	38/31	16	<xM>/<xR>
DIA	-	-	-	-	-	3%7	-	
DIVi	2	-	-	-	3	58-68	16	
ENTER	-	-	-	n+1	-	4*n+18	-	n = # of general registers saved
EXIT	-	-	-	n+1	-	5*n+17	-	n = # of general registers restored
EXTi	2	-	-	1	1	19-29	-	field in memory
EXTi	2	-	-	-	1	17-51	-	field in register
EXTSi	2	-	-	1	1	26-36	-	
FFSi	2	2	-	-	1	24-28	24	
FLAG	-	-	0/4	0/3	-	6/44	-	no trap / trap
IBITi	2/1	2/0	-	-	1	17/9	-	<xM>/<xR>
INDEXi	2	-	-	-	2	25	16	
INSi	2	-	-	2	1	29-39	-	field in memory
INSi	1	-	-	-	1	28-96	-	field in register
INSSi	2	-	-	2	1	39-49	-	
JSR	1	-	-	1	1	5%15	-	

Table B-1 (Cont.)

MNEMONIC	TEA	TR	TOPR	TOPD	TOPi	TCY	L	NOTES
JUMP	1	-	-	-	-	2%6	-	
LMR	1	-	-	-	1	30%34	-	
LPRi	1	-	-	-	1	19-33	-	
LSHi	2	1	-	-	2	14-45	-	
MEIi	2	-	-	-	4	23	16	
MODi	2	-	-	-	3	54-73	16	
MOVi	2/1/0	-	-	-	2/1/0	1/3/3	-	<xM>/<MR>/<RR>
MOVMi	2	-	-	-	2*n	3*n+20	-	n = # of elements in block
MOVQi	1/0	-	-	-	1/0	2/3	-	<M>/<R>
MOVSi	-	-	-	-	2*n	13*n+18	-	n = # of elements, no options
MOVSi	-	-	-	-	2*n	24*n+54	-	B, W and/or U option in effect
MOVST	-	n	-	-	2*n	27*n+54	-	Translated
MOVSUi	2	-	-	-	2	33%37	-	
MOVUSi	2	-	-	-	2	33%37	-	
MOVXBD	2	1	-	1	-	6	-	
MOVXBW	2	1	1	-	-	6	-	
MOVXWD	2	-	1	1	-	6	-	

Table B-1 (Cont.)

MNEMONIC	TOPA	TOPB	TOPW	TOPD	TOPi	TCY	L	NOTES
MOVZBD	2	1	-	1	-	5	-	
MOVZBW	2	1	1	-	-	5	-	
MOVZWD	2	-	1	1	-	5	-	
MULi	2	-	-	-	3	15	16	
NEGi	2	-	-	-	2	5	-	
NOP	-	-	-	-	-	3	-	
NOTi	2	-	-	-	2	5	-	
ORi	2/1/0	-	-	-	3/1/0	3/4/4	-	<xM>/<MR>/<RR>
QUOi	2	-	-	-	3	49-55	16	
RDVAL	1	1	-	-	-	21	-	
REMi	2	-	-	-	3	57-62	16	
RESTORE	-	-	-	n	-	5*n+12	-	n = # of general registers restored
RET	-	-	-	1	-	2%8	-	
RETI	-	1	3	3	-	39%45	-	
RETT	-	-	2	2	-	35%41	-	
ROTi	2	1	-	-	2	14-45	-	
RXP	-	-	1	2	-	2%6	-	

Table B-1 (Cont.)

MNEMONIC	TEA	TOPB	TOPW	TOPD	TOPi	TCY	L	NOTES
Scondi	1	-	-	-	1	9/10	-	False / True
SAVE	-	-		n	-	4*n+13	-	n = # of general registers saved
SBITi	2/1	2/0		-	1	15/7	-	<xM>/<xR>
SBITIi	2/1	2/0		-	1	15/7	-	<xM>/<xR>
SETCFG	-	-		-	-	15	-	
SKPSi	-	-	-	-	n	27*n+51	-	n = # of elements, not Translated
SKPST	-	n	-	-	n	30*n+51	-	Translated
SMR	1	-		-	1	25	-	
SPRi	1	-	-	-	1	21-27	-	
SUBi	2/1/0	-	-	-	3/1/0	3/4/4	-	<xM>/<MR>/<RR>
SUBCi	2/1/0	-		-	3/1/0	3/4/4	-	<xM>/<MR>/<RR>
SUBPi	2	-	-	-	3	16/18	-	no carry / carry
SVC	-	-	4	3	-	40	-	
TBITi	2/1	1/0	-	-	1	14/4	-	<xM>/<xR>
WAIT	-	-	-	-	-	6-?	-	? = until an interrupt/reset
WRVAL	1	1	-	-	-	21	-	
XORi	2/1/0	-	-	-	3/1/0	3/4/4	-	<xM>/<MR>/<RR>

B.7 Floating-Point Execution Times

Table B-2 gives execution timing information for Floating-Point instructions. Some additional timing definitions are used, as given below.

f - The floating-point operation length.

Standard Floating (32 bits): f=1
Long Floating (64 bits): f=2

Tf - The time required to transfer 32 bits of a floating-point value to or from the NS16081 Floating-Point Unit.

Tf = 4 always.

Ti - The time required to transfer an integer value to or from the NS16081 Floating-Point Unit.

Byte:	Ti = 2
Word:	Ti = 2
Double-Word:	Ti = 4

Table B-2 Floating-Point Instruction Execution Times

INSTRUCTION	CASE	TEA	TOPD	TOPi	Ti	Tf	TCY
MOVf	<MM>	1	3	-	-	3	23
	<RR>	-	-	-	-	-	27
	<MR>	1	2	-	-	2	23
	<RM>	-	1	-	-	1	27
ADDf, SUBf	<MM>	2	3f	-	-	3f	70
	<RR>	-	-	-	-	-	74
	<MR>	1	f	-	-	f	70
	<RM>	1	2f	-	-	2f	70
MULf	<MM>	2	3f	-	-	3f	30+14f
	<RR>	-	-	-	-	-	34+14f
	<MR>	1	f	-	-	f	30+14f
	<RM>	1	2f	-	-	2f	30+14f
DIVf	<MM>	2	3f	-	-	3f	55+30f
	<RR>	-	-	-	-	-	59+30f
	<MR>	1	f	-	-	f	55+30f
	<RM>	1	2f	-	-	2f	55+30f
ABSf, NEGf	<MM>	1	2f	-	-	2f	20
	<RR>	-	-	-	-	-	24
	<MR>	1	f	-	-	f	20
	<RM>	-	f	-	-	f	24
CMPf	<MM>	2	2f	-	-	2f	45
	<RR>	-	-	-	-	-	49
	<MR>	1	f	-	-	f	45
	<RM>	1	f	-	-	f	45
MOVLF	<MM>	1	3	-	-	3	23
	<RR>	-	-	-	-	-	27
	<MR>	1	2	-	-	2	23
	<RM>	-	1	-	-	1	27
MOVFL	<MM>	1	3	-	-	3	22
	<RR>	-	-	-	-	-	26
	<MR>	1	1	-	-	1	22
	<RM>	-	2	-	-	2	26
MOVif	<MM>	1	f	1	1	f	53
	<MR>	1	-	1	1	-	53
ROUNDfi, TRUNCfi, FLOORfi	<MM>	1	f	1	1	f	53
	<RM>	-	-	1	1	-	66
SFSR	<M>	-	1	-	-	1	13
LFSR	<M>	1	1	-	-	1	18