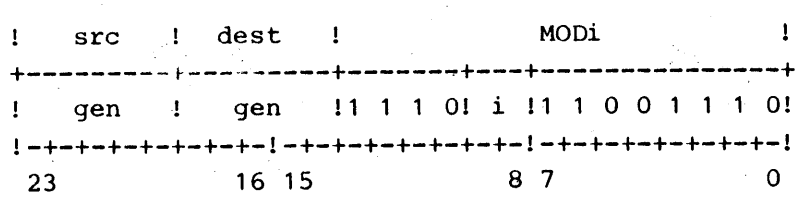
Modulus

Syntax: **MODi** **src,** **dest**

gen	gen
read.i	rmw.i

MODB
MODW
MODD



The MODi instruction places the value dest modulo src in the dest operand location. The modulus is the remainder after division is performed as per the DIVi instruction (q.v.). It is computed as

```
dest - ((dest DIV src) * src)
```

where dest DIV src is rounded to the next integer less than or equal to the exact quotient. The result of a MODi instruction always has the sign of the src operand (i.e., the divisor) unless the result is zero, which is always positive. Compare the REMi instruction (q.v.).

The src and dest operands are interpreted as signed integers.

Flags Affected: None.

Traps: Divide by Zero Trap (DVZ) is activated if src equals zero.

Example:

```
MODB  4(SB), 8(SB)                    CE B8 D6 04 08
```

This example computes the modulus of the operands specified by 4(SB) and 8(SB) and places the result in the byte at 8(SB).

The action of this instruction for four different cases is illustrated below:

	Operands	Operand Values: Hex (Dec) Before	After
Case 1:	4(SB)	0A (+10)	0A (+10)
	8(SB)	1F (+31)	01 (+1)
Case 2:	4(SB)	F6 (-10)	F6 (-10)
	8(SB)	1F (+31)	F7 (-9)
Case 3:	4(SB)	F6 (-10)	F6 (-10)
	8(SB)	E1 (-31)	FF (-1)
Case 4:	4(SB)	0A (+10)	0A (+10)
	8(SB)	E1 (-31)	09 (+9)

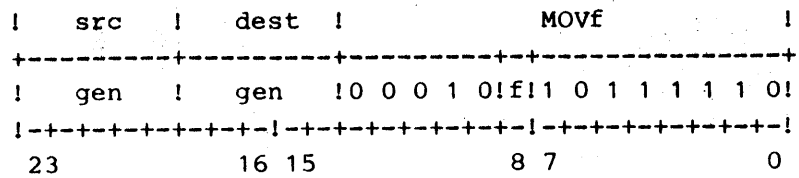
Move Floating Point

Syntax: **MOVf** **src,** **dest**

gen gen

read.f write.f

MOVF
MOVL



The MOVF instruction copies the src operand to the dest operand location.

Flags Affected: None.

Traps: Undefined Instruction Trap (UND) is activated if the F bit in the CFG register is clear.

Example:

```
MOVF  F0, 8(SB)                    BE 85 06 08
```

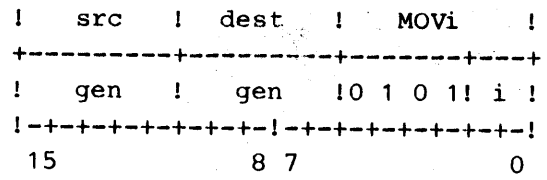
This example moves the single-precision number in register F0 to the operand at address 8(SB).

The instruction is illustrated below:

Operands	Operand Value: Hex (Dec) Before	After
F0	3F800000 (+1.0)	3F800000 (+1.0)
8(SB)	AAAAAAAA	3F800000 (+1.0)

Syntax: **MOVi** **src,** **dest**
gen gen
read.i write.i

MOVB
MOVW
MOVD



The MOVi instruction copies the src operand to the dest operand location.

Flags Affected: None.

Traps: None.

Example:

```
MOVD  R0, 8(SB)                    97 06 08
```

This example copies the contents of register R0 to the double-word at the address specified by 8(SB).

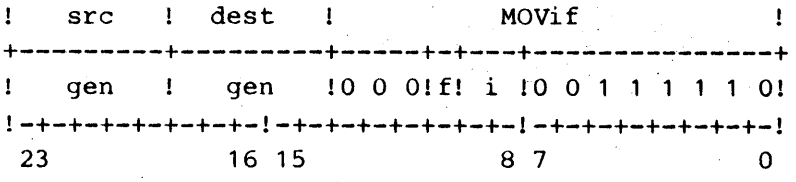
The instruction is illustrated below.

Operands	Operand Values: Hex Before	After
R0	12345678	12345678
8(SB)	AAAAAAAA	12345678

Move Converting Integer to Floating Point

Syntax: **MOVif** **src**, **dest**

	gen	gen	MOVBF MOVBL
			MOVWF MOVWL
	read.i	write.f	MOVDF MOVDL



The MOVif instruction converts the integer operand src to a single- or double-precision floating-point number and places the result in the dest operand location.

Rounding, if required, is controlled by the Rounding Mode bits in the FSR.

Flags Affected: No PSR flags. One FSR flag may be affected:
IF is set on an inexact result; unaffected otherwise.
See the Trap(FPU) discussion below.

Traps: Undefined Instruction Trap (UND) is activated if the F bit in the CFG register is clear.

Floating-Point Trap (FPU) is activated if a floating-point exception is detected. See Section 3.3. Particularly relevant is the Inexact Result trap if it is enabled in the FSR. It can occur in the MOVDF form, because in this case there are fewer significant bits in dest than in src. The smallest integer values for which this will happen are +16,277,217 (01000001 Hex) and -16,277,217 (FEFFFFFF Hex).

Examples:

```
1.  MOVBF   2, F0                   3E 04 A0 02
2.  MOVDL   16(SB), F2              3E 83 D0 10
```

Example 1 converts the integer constant 2 to a single-precision number and copies the number to the register F0.

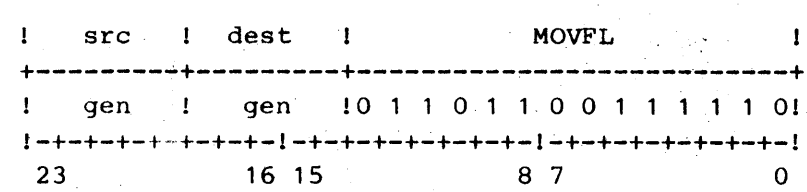
Example 2 converts the double-word integer at the address specified by 16(SB) to a double-precision number and places the number into the register pair (F2,F3).

These instructions are illustrated below:

	Operands	Operand Values: Hex (Dec) Before	After
Ex. 1:	2 (immediate)	02 (+2)	--
	F0	AAAAAAAA	40000000 (+2.0)
Ex. 2:	16(SB)	20401110 (+541069584)	20401110 (+541069584)
	(F2,F3)	AAAAAAAAAAAAAAAA	41C0200888000000 (+541069584.0)

Move Floating to Long Floating

Syntax: **MOVFL** **src,** **dest**
gen gen
read.F write.L



The MOVFL instruction converts the src operand to double-precision format and places the result in the dest operand location.

Flags Affected: None.

Traps: Undefined Instruction Trap (UND) is activated if the F bit in the CFG register is clear.

Floating-Point Trap (FPU) is activated if a floating-point exception is detected. See Section 3.3.

Example:

```
MOVFL  8(SB), F0                    3E 1B D0 08
```

This example converts the single-precision number at the address specified by 8(SB) to a double-precision number and places the number in the register pair (F0,F1).

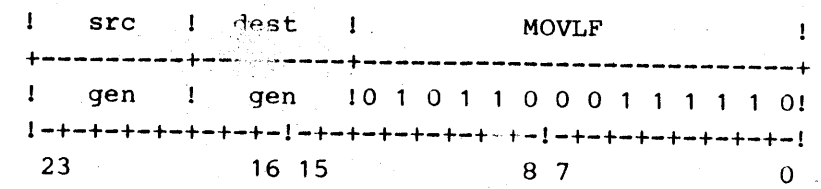
The instruction is illustrated below:

Operands	Operand Values: Hex (Dec) Before	After
8(SB)	3F800000 (+1.0)	3F800000 (+1.0)
F0	AAAAAAAAAAAAAAAA	3FF0000000000000 (+1.0)

MOVLF

Move Long Floating to Floating

Syntax: **MOVLF** **src,** **dest**
gen gen
read.L write.F



The MOVLF instruction converts the src operand to a single-precision number and places the result in the dest operand location.

Rounding is performed, if necessary, according to the rounding mode selected in the FSR. See Section 3.3 for details of rounding modes.

Flags Affected: No PSR flags. Two FSR flags may be affected:
UF is set if an underflow occurs; unaffected otherwise.
IF is set on an inexact result; unaffected otherwise.
See Section 3.3 for floating-point exception definitions.

Traps: Undefined Instruction Trap (UND) is activated if the F bit in the CFG register is clear.

Floating-Point Trap (FPU) is activated if a floating-point exception is detected. See Section 3.3. Particularly relevant cases are:

Overflow, which occurs if the src operand is too great in absolute value to be represented as a single-precision number,

Underflow, which, if enabled in the FSR, occurs if the src operand is too small in absolute value to be represented as a normalized single-precision number, and

Inexact Result, which, if enabled in the FSR, occurs if a loss of precision occurs in the conversion.

Move Long Floating to Floating (continued)

Example:

```
MOVLF   F0, 12(SB)                    3E 96 06 0C
```

This example converts the double-precision number in register pair (F0,F1) to a single-precision number and places the result at address 12(SB).

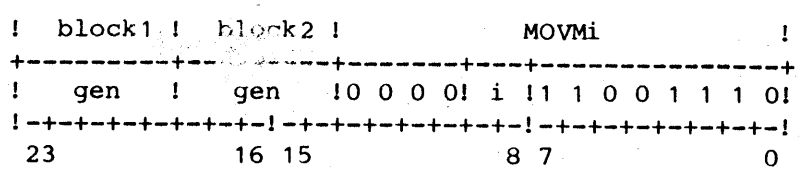
The instruction is illustrated below:

Operands	Operand Values: Hex (Dec) Before	After
F0	3FF0000000000000 (+1.0)	3FF0000000000000 (+1.0)
12(SB)	AAAAAAAA	3F800000 (+1.0)

MOVMi
Move Multiple

Syntax: **MOVMi** **block1**, **block2**, **length**
gen addr, gen addr, disp

MOVMB
MOVMW
MOVMD



The MOVMi instruction copies the contents of block1 to block2. The instruction copies consecutive integers from block1 to consecutive integer locations in block2.

In assembly language, the length operand is specified as the number of integers in each block. In the machine instruction, however, the length operand is encoded according to the formula

(num - 1) * i

where num is the number of integers in each block, and i is the number of bytes per integer. A block may not be greater than 16 bytes in length.

Flags Affected: None.

Traps: None.

Example:

```
MOVMW 10(R0), 16(R1), 4          CE 41 42 0A 10 06
```

This instruction copies four word-long integers from the block starting at the address specified by 10(R0) to the block starting at the address specified by 16(R1).

The instruction is illustrated below:

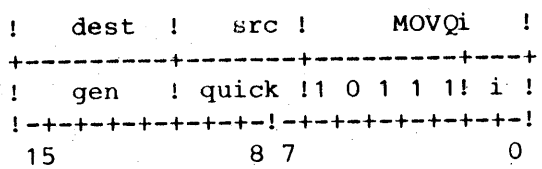
Operands	Operand Values: Hex Before	After
R0	00002000	00002000
R1	0000F000	0000F000
0000200A *	1FBE 10A9 8729 6511	1FBE 10A9 8729 6511
0000F010 **	AAAA AAAA AAAA AAAA	1FBE 10A9 8729 6511

* The address of the first block, as specified by 10(R0).
** The address of the second block, as specified by 16(R1).

Move Quick Integer

Syntax: **MOVQi** **src,** **dest**
quick gen
write.i

MOVQB
MOVQW
MOVQD



The MOVQi instruction copies the src operand to the dest operand location. Before the copy operation, src is sign-extended to the length of dest.

Flags Affected: None.

Traps: None.

Example:

```
MOVQW  7, TOS                    DD BB
```

This example pushes the quick value 7 as a word onto the top of the stack. The high-order bits of the result are zero-filled due to sign-extension.

The instruction is illustrated below:

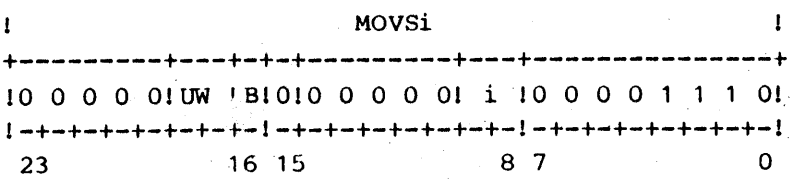
Operands	Operand Values: Hex Before	After
7 (quick)	0007 *	--
SP	0000FFEE	0000FFEC
Stack:		
0000FFEC	xxxx	0007
0000FFEE	AAAA	AAAA

* This value is the internal representation of the Quick value 7, after sign-extension to Word length. The operand is encoded within the instruction as binary 0111.

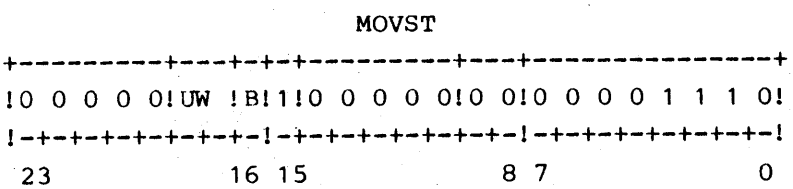
MOVSi
MOVST
Move String

Syntax: **MOVSi options**

MOVSB
MOVSW
MOVSD
MOVST



Syntax: **MOVST options**



Operands of the MOVSi and MOVST instructions are specified in General-Purpose Registers:

R0 - Number of string elements to be processed.
R1 - Address of current String 1 element.
R2 - Address of current String 2 element.
R3 - Address of translation table (MOVST form only).
R4 - Match value (with Until Match or While Match option only).

The MOVSi instruction copies consecutive elements of String 1 (address in R1) to consecutive element locations in String 2 (address in R2). After an element is copied, the instruction sets register R1 to the address of the next element to copy, sets register R2 to the address of the next location to receive an element, and sets R0 to the number of elements remaining to be copied. See Section 3.7 for the exact sequences followed by String instructions.

The MOVST instruction copies one-byte elements from String 1, after translation, to String 2. The translated value to be copied is found by adding the current element from String 1 as an unsigned integer to the translation table address found in register R3. The instruction copies elements and sets registers as described above. See Section 3.7 for details of string translation.

Options may be specified by listing the letters B (Backward), U (Until Match) and W (While Match) as operands. The U and W options are mutually exclusive. See Section 3.7 for details of the options available in String instructions.

Move String (continued)

In the machine instruction, the options are encoded in the B and UW fields as follows:

B field =	0	Forward direction.
	1	Backward direction.
UW field =	00	Neither Until Match nor While Match.
	01	While Match.
	10	(reserved)
	11	Until Match.

String instructions are interruptible. See Section 3.7.

Flags Affected: F is set if the U or W option is specified and the corresponding Until/While condition is met, otherwise it is cleared.

Traps: None.

Example:

```
MOVST                                   0E 80 00
```

This example moves byte-long integers from the first string, after translation, to the second string.

The instruction is illustrated below:

Operands	Operand Values: Hex (De.) Before	After
R0	00000020 (+32)	00000000 (0)
R1	00002000	00002021
R2	0000F000	0000F021
R3	00010000	00010000
UPSR	nzfxxltc	nz0xxltc

Translation Table Contents

```
10000   00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
        16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
```

String Contents Before

```
2000    1E 04 05 1C 0A 14 0C 0B 09 07 1F 0F 17 01 00 11
        1F 1D 1A 09 01 12 14 0E 1E 0A 00 03 09 06 16 18

F000    AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA
        AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA
```

String Contents After

```
2000    1E 04 05 1C 0A 14 0C 0B 09 07 1F 0F 17 01 00 11
        1F 1D 1A 09 01 12 14 0E 1E 0A 00 03 09 06 16 18

F000    30 04 05 28 10 20 12 11 09 07 31 15 23 01 00 17
        31 29 26 09 01 18 20 14 30 10 00 03 09 06 22 24
```

This case of the above example translates 32 binary integers (in the range 0-31) into binary coded decimal (BCD) values. Each integer is read from String 1 (address given by R1) and used as an offset into the translation table at address 10000 (Hex). The BCD value found at that address in the translation table is then copied to the current location in String 2 (address in R2).

Move Value from Supervisor to User Space

Syntax: **MOVSUi** **src,** **dest**

MOVSUB
MOVSUW
MOVSUD

	src	dest	MOVSUi
	gen	gen	0 0 1 1 i 1 0 1 0 1 1 1 0
	addr	addr	

23 16 15 8 7 0

The MOVSUi instruction copies the src operand in Supervisor space to the dest operand location in User space. User Mode protection is applied to the User space access.

Flags Affected: None.

Traps: Illegal Instruction Trap (ILL) is activated if the U flag is set.

Example:

```
MOVSUB  5(SP), 9(SB)                AE 8C CE 05 09
```

This example copies the byte at the operand address 5(SP) in supervisor space to the operand address 9(SB) in user space.

The instruction is illustrated below:

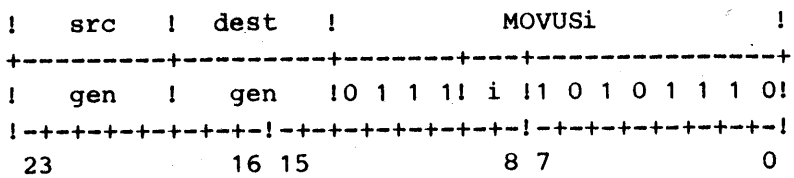
	Operand Values: Hex	
Operands	Before	After
5(SP) Supervisor	10	10
9(SB) User	AA	10

MOVUSi

Move Value from User to Supervisor Space

Syntax: **MOVUSi** **src,** **dest**
gen gen
addr addr

MOVUSB
MOVUSW
MOVUSD



The MOVUSi instruction moves the src operand in User space to the dest operand location in Supervisor space. User Mode protection is applied to the User space access.

Flags Affected: None.

Traps: Illegal Instruction Trap (ILL) is activated if the U flag is set.

Example:

```
MOVUSB  9(SB), 5(SP)                AE 5C D6 09 05
```

This example moves the byte at the address specified by 9(SB) in user space to the address specified by 5(SP) in supervisor space.

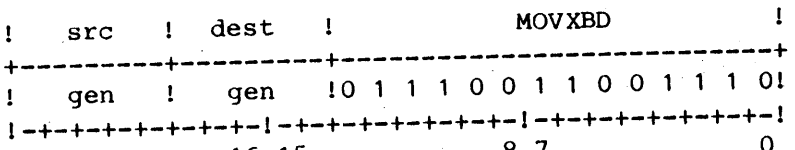
The instruction is illustrated below:

	Operand Values: Hex	
Operands	Before	After
9(SB) User	10	10
5(SP) Supervisor	AA	10

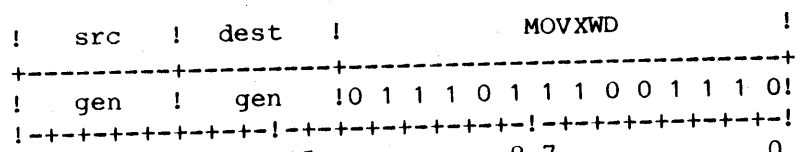
Move with Sign-Extension

MOVXBD
MOVXWD

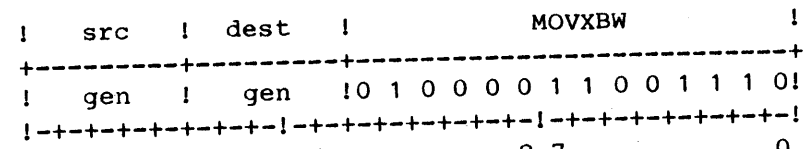
Syntax: **MOVXBD** **src,** **dest**
gen gen
read.B write.D



Syntax: **MOVXWD** **src,** **dest**
gen gen
read.W write.D



Syntax: **MOVXBW** **src,** **dest**
gen gen
read.B write.W



The MOVX instructions convert signed integers to any greater length while preserving their signed values through sign-extension.

The MOVXBD instruction copies the byte-length src operand to the low-order byte of the double-word dest operand and extends the src operand's sign bit through the remaining high-order bits of the dest operand.

The MOVXWD instruction copies the word-length src operand to the low-order word of the double-word dest operand and extends the src operand's sign bit through the remaining high-order bits of the dest operand.

The MOVXBW instruction copies the byte-length src operand to the low-order byte of the word dest operand and extends the src operand's sign bit through the remaining high-order bits of the dest operand.

The src and dest operands are interpreted as signed integers.

Flags Affected: None.

Traps: None.

Example:

```
MOVXBW  2(SB), R0                CE 10 D0 02
```

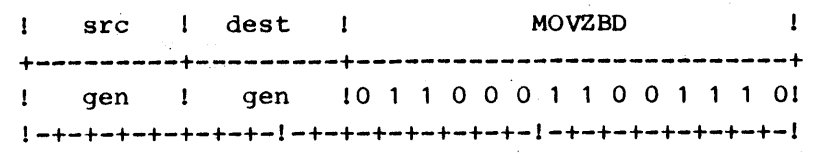
This example copies the byte at the address specified by 2(SB) to the low-order byte of register R0 and extends the sign bit of the byte through the next eight bits of R0. The instruction affects the low-order word of R0 only.

The instruction (for two cases) is illustrated below:

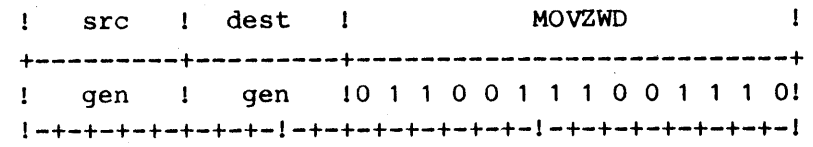
	Operands	Operand Values: Hex (Dec) Before	After
Case 1:	2(SB)	F0 (-16)	F0 (-16)
	R0	AAAAAAAA	AAAAFFF0 (-16)
Case 2:	2(SB)	70 (+112)	70 (+112)
	R0	AAAAAAAA	AAAA0070 (+112)

Move with Zero-Extension

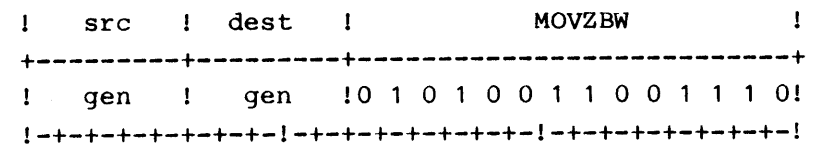
Syntax: **MOVZBD** **src,** **dest** MOVZBD
gen gen MOVZWD
read.B write.D MOVZBW



Syntax: **MOVZWD** **src,** **dest**
gen gen
read.W write.D



Syntax: **MOVZBW** **src,** **dest**
gen gen
read.B write.W



The MOVZ instructions convert unsigned integers to any greater length while preserving their unsigned values through zero-extension.

The MOVZBD instruction copies the byte-length src operand to the low-order byte of the double-word dest operand and zero-fills the remaining high-order bits of the dest operand.

The MOVZWD instruction copies the word-length src operand to the low-order word of the double-word dest operand and zero-fills the remaining high-order bits of the dest operand.

The MOVZBW instruction copies the src operand to the low-order byte of the dest operand and zero-fills the remaining eight bits of dest.

The src and dest operands are interpreted as unsigned integers.

Flags Affected: None.

Traps: None.

Example:

```
MOVZBW  -4(FP), R0                    CE 14 C0 7C
```

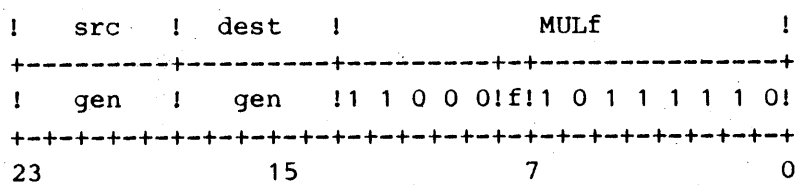
This example copies the byte at the address specified by -4(FP) to the low-order byte of register R0 and sets the next eight bits of register R0 to zero. The instruction affects only the low-order word of R0.

The instruction is illustrated below:

Operands	Operand Values: Hex (Unsigned Dec) Before	After
-4(FP)	FF (+255)	FF (+255)
R0	AAAAAAAA	AAAA00FF (+255)

Multiply Floating

Syntax: **MULf** src, **dest** MULF
gen gen MULL
read.f rmw.f



The MULf instruction multiplies the src and dest operands and places the result in the dest operand location. Results for normalized and zero operands are given in the table below. The symbols "+n" and "-n" represent non-zero normalized numbers, positive and negative, respectively. The symbols "+z" and "-z" represent positive and negative zero, respectively.

dest: / src	+n	-n	+z	-z
+n	*	*	+z	-z
-n	*	*	-z	+z
+z	+z	-z	+z	-z
-z	-z	+z	-z	+z

* The result in these cases is the product of the two operands.

Flags Affected: No PSR flags. Two FSR flags may be affected:
UF is set if an underflow occurs; unaffected otherwise.
IF is set on an inexact result; unaffected otherwise.
See Section 3.3 for floating-point exception definitions.

Traps: Undefined Instruction Trap (UND) is activated if the F bit in the CFG register is clear.

Floating-Point Trap (FPU) is activated if a floating-point exception is detected. See Section 3.3.

Examples:

1.	MULF F0, F7	BE F1 01
2.	MULL -8(FP), 8(SB)	BE B0 C6 78 08

Example 1 multiplies the single-precision numbers in registers F0 and F7 and places the result in register F7.

Example 2 multiplies the double-precision numbers at addresses -8(FP) and 8(SB) and places the double-precision result at address 8(SB).

Thse instructions are illustrated below:

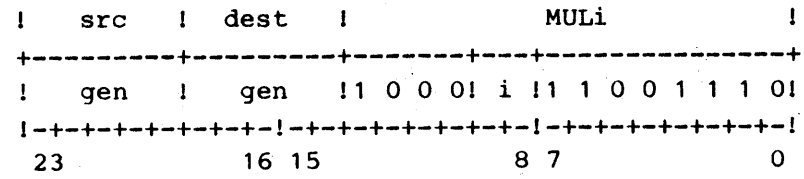
	Operands	Operand Values: Hex (Dec) Before	After
Ex. 1:	F0	42250000 (+41.25)	42250000 (+41.25)
	F7	40A00000 (+5.0)	434E4000 (+206.25)
Ex. 2:	-8(FP)	409F440000000000 (+2001.0)	409F440000000000 (+2001.0)
	8(SB)	40F3218E00000000 (+78360.875)	41A2B128DDC00000 (+156800110.875)

Multiply

Syntax: **MULi** **src,** **dest**

MULB
MULW

	src	dest
	gen	gen
	read.i	rmw.i



The MULi instruction multiplies the src and dest operands and places the product in the dest operand location. If the product is longer than dest, the high-order bits are truncated.

The src and dest operands are interpreted as signed integers.

Flags Affected: None.

Traps: None.

Examples:

1.	MULW 5, R0	CE 21 A0 00 05
2.	MULD 4(-4(FP)), 3(SB)	CE A3 86 7C 04 03

Example 1 multiplies the constant 5 and the low-order word of register R0 and places the result in the low-order word of register R0.

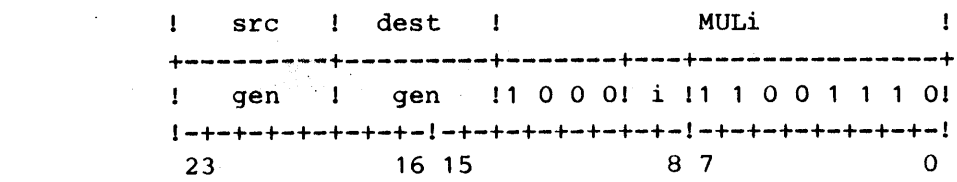
Example 2 multiplies the double-word at the memory address specified by 4(-4(FP)) by the double-word at the address specified by 3(SB). The instruction places the result in the double-word at address 3(SB).

These instructions are illustrated below:

	Operands	Operand Values: Hex (Dec) Before	After
Ex. 1:	5 (immediate)	0005 (+5)	--
	R0	BBBB000A (+10)	BBBB0032 (+50)
Ex. 2:	4(-4(FP))	FFFFFFFE (-2)	FFFFFFFE (-2)
	3(SB)	FFFFFFF0 (-16)	00000020 (+32)

Multiply

Syntax: **MULi** **src,** **dest** MULB
gen gen MULW
read.i rmw.i



The MULi instruction multiplies the src and dest operands and places the product in the dest operand location. If the product is longer than dest, the high-order bits are truncated.

The src and dest operands are interpreted as signed integers.

Flags Affected: None.

Traps: None.

Examples:

1. `MULW 5, R0` — `CE 21 A0 00 05`
2. `MULD 4(-4(FP)), 3(SB)` — `CE A3 86 7C 04 03`

Example 1 multiplies the constant 5 and the low-order word of register R0 and places the result in the low-order word of register R0.

Example 2 multiplies the double-word at the memory address specified by 4(-4(FP)) by the double-word at the address specified by 3(SB). The instruction places the result in the double-word at address 3(SB).

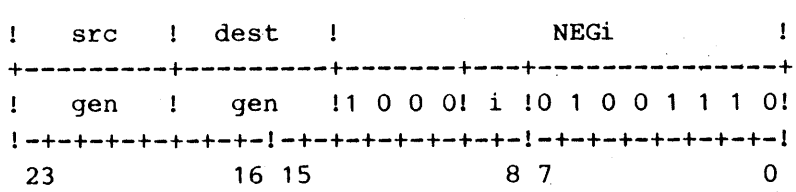
These instructions are illustrated below:

	Operands	Operand Values: Hex (Dec) Before	After
Ex. 1:	5 (immediate)	0005 (+5)	--
	R0	BBBB000A (+10)	BBBB0032 (+50)
Ex. 2:	4(-4(FP))	FFFFFFFE (-2)	FFFFFFFE (-2)
	3(SB)	FFFFFFF0 (-16)	00000020 (+32)

Negate

Syntax: **NEGi** src, **dest**

gen	gen	NEGB
read.i	write.i	NEGW
		NEGD



The NEGi instruction negates (takes the two's complement of) the src operand by subtracting src from zero and places the result in the dest operand location.

Flags Affected: C is set on a borrow from subtraction, cleared if no borrow. (A borrow will always occur in this instruction unless the src operand is zero.)

F is set on an overflow from subtraction, cleared if no overflow. This condition will occur if the src operand is the most negative number that can be represented in the operand length specified by the programmer. This value for bytes is -128 (Hex 80); for words it is -32768 (Hex 8000) and for double-words it is -2,147,483,648 (Hex 80000000). These values have no corresponding positive values in the same operand length. The result returned on an overflow is the original src operand.

Integer borrow and overflow conditions are defined in Section 3.1.

Traps: None.

Examples:

```
1.  NEGB  R5, R6                    4E A0 29
2.  NEGW  4(SB), 6(SB)              4E A1 D6 04 06
```

Example 1 negates the low-order byte of register R5 and places the result in the low-order byte of register R6. The remaining bytes of registers R5 and R6 are neither used nor affected.

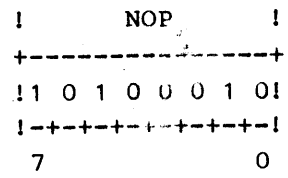
Example 2 negates the word at the memory address specified by 4(SB) and places the word result at the memory address specified by 6(SB).

These instructions are illustrated below:

	Operands	Operand Values: Hex (Dec) Before	After
Ex. 1:	R5	AAAAAAFF (-1)	AAAAAAFF (-1)
	R6	BBBBBBBB	BBBBBB01 (+1)
	UPSR	nzfxxltc	nz0xxlt1
Ex. 2:	4(SB)	0041 (+65)	0041 (+65)
	6(SB)	xxxx	FFBF (-65)
	UPSR	nzfxxltc	nz0xxlt1

No Operation

Syntax: **NOP**



The NOP instruction passes control to the next sequential instruction. No operation is performed.

Flags Affected: None.

Traps: None.

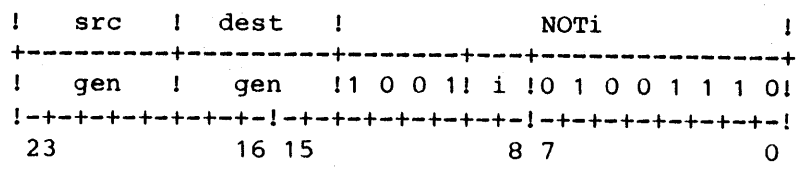
Example:

```
NOP                                    A2
```

NOTi
Complement Boolean

Syntax: **NOTi** **src,** **dest**
gen gen
read.i write.i

NOTB
NOTW



The NOTi instruction complements (inverts) the Boolean value of the src operand and places the result in the dest operand location. The complement of a Boolean value is that value with its least-significant bit complemented. The Boolean value "True" (the integer value 1) thus becomes "False" (the integer value 0) and vice versa. Boolean values are described in Section 3.4.

Flags Affected: None.

Traps: None.

Examples:

```
1.  NOTB  R0, R0                4E 24 00
2.  NOTW  10(R1), TOS           4E E5 4D 0A
```

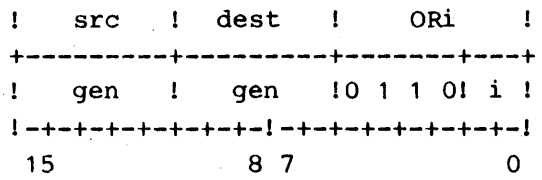
Example 1 complements the Boolean value in the low-order byte of register R0. The remaining bytes of R0 are neither used nor affected.

Example 2 complements the one-word Boolean value at memory address 10(R1) and pushes the result as a word onto the top of the stack.

These instructions are illustrated below:

	Operands	Operand Values: Before	Hex (Boolean) After
Ex. 1:	R0	AAAAAA01 (True)	AAAAAA00 (False)
Ex. 2:	10(R1)	AAAA0000 (False)	AAAA0000 (False)
	Stack:		
	0000FFE0	xxxx	0001 (True)
	0000FFE2	AAAA	AAAA
	SP	0000FFE2	0000FFE0

Syntax: **ORi** **src,** **dest** ORB
gen gen ORW
read.i rmw.i ORD



The ORi instruction performs a bit-wise logical OR operation between the src and dest operands and places the result in the dest operand location.

The instruction ORs each bit in src with the corresponding dest bit. If two corresponding bits are both "0", the dest bit is set to "0"; otherwise, the dest bit is set to "1".

Flags Affected: None.

Traps: None.

Example:

```
ORB  -6(FP), 11(SB)                98 C6 7A 0B
```

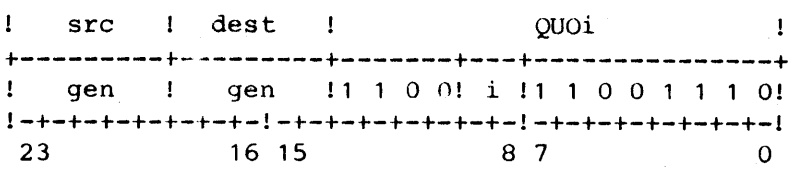
This example ORs the bytes at memory addresses -6(FP) and 11(SB), placing the result at memory address 11(SB).

The instruction is illustrated below:

Operands	Operand Values: Binary Before	After
-6(FP)	11011000	11011000
11(SB)	00001011	11011011

Syntax: **QUOi** **src,** **dest**
gen gen
read.i rmw.i

QUOB
QUOW
QUOD



The QUOi instruction divides the dest operand by the src operand, returning as the result the nearest integer whose absolute value is less than or equal to the absolute value of the exact quotient. The result is placed in the dest operand location.

The src and dest operands are interpreted as signed integers.

Flags Affected: None.

Traps: Divide by Zero trap (DVZ) is activated if src equals zero.

Examples:

1.	QUOB R0, R7	CE F0 01
2.	QUOW 4(SB), 8(SB)	CE B1 D6 04 08

Example 1 divides the low-order byte of register R7 by the low-order byte of register R0, placing the result in the low-order byte of register R7.

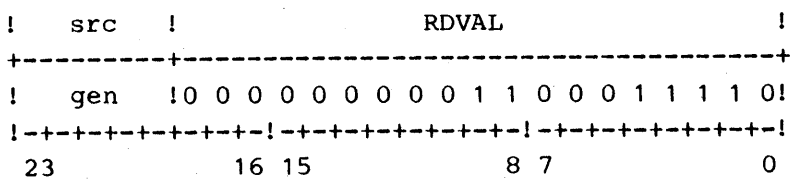
Example 2 divides the word at address 8(SB) by the word at address 4(SB), placing the one-word result at address 8(SB).

These instructions are illustrated below:

	Operands	Operand Values: Hex (Dec) Before	After
Ex. 1:	R0	AAAAAA05 (+5)	AAAAAA05 (+5)
	R7	BBBBBB22 (+34)	BBBBBB06 (+6)
Ex. 2:	4(SB)	0010 (+16)	0010 (+16)
	8(SB)	FFE1 (-31)	FFFF (-1)

Validate Address for Reading

Syntax: **RDVAL** **loc**
gen
addr



The RDVAL instruction checks the protection level assigned to the user-mode virtual memory address specified as loc. If the address is allowed to be read while the CPU is in user mode, the F flag in the PSR is cleared. If the address is not allowed to be read, the F flag in the PSR is set. An address which is protected against reading is also protected against writing, and is therefore inaccessible for any use by a user-mode program.

NOTE: Although the final effective address of loc is interpreted as a user-mode virtual address, any memory references required in order to calculate that effective address are interpreted as using supervisor-mode addresses. This will occur in using the Memory Relative and External addressing modes for loc.

Flags Affected: F is set if loc is inaccessible in user mode, cleared otherwise.

Traps: Undefined Instruction Trap (UND) is activated if the M bit in the CFG register is clear.

Illegal Operation Trap (ILL) is activated if this instruction is attempted while the PSR U bit is set.

Abort Trap (ABT) is activated if the Level 1 page table entry for loc is invalid (V bit = 0). No trap is issued for an invalid Level 2 page table entry, and the Protection Level (PL) field is assumed to be present regardless of the state of the V bit.

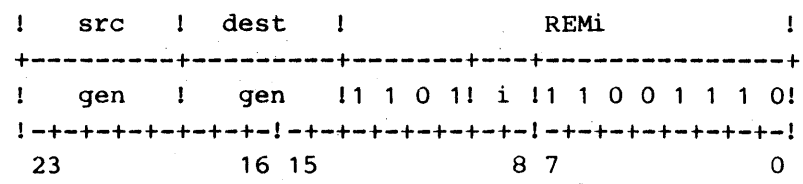
Example:

```
RDVAL 512(R0)                    1E 03 40 82 00
```

The above example checks the protection level assigned to the address 512(R0) and sets or clears the F flag to indicate the result.

REMi
Remainder

Syntax: **REMi** **src,** **dest** REMB
gen gen REMW
read.i rmw.i REMD



The REMi instruction places the remainder from dividing dest by src into the dest operand location. The remainder is computed as

dest - src * (dest QUO src) ,

where dest QUO src is the result of dividing dest by src as per the QUOi instruction (q.v.).

The result of a REMi instruction always has the sign of the dest operand (i.e. the dividend) unless the result is zero, which is always positive. Compare the MODi instruction (q.v.).

The src and dest operands are interpreted as signed integers.

Flags Affected: None.

Traps: Divide by Zero trap (DVZ) is activated if src equals zero.

Remainder (continued)

Example:

```
REMB  4(SB), 8(SB)                    CE B4 D6 04 08
```

This example computes the remainder from dividing the one-byte operand at address 8(SB) by the one-byte operand at address 4(SB) and places the result as a byte at address 8(SB).

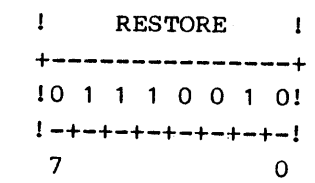
The action of the above instruction for four different cases is illustrated below:

	Operands	Operand Values: Hex (Dec) Before	After
Case 1:	4(SB)	0F (+15)	0F (+15)
	8(SB)	21 (+33)	03 (+3)
Case 2:	4(SB)	F1 (-15)	F1 (-15)
	8(SB)	21 (+33)	03 (+3)
Case 3:	4(SB)	F1 (-15)	F1 (-15)
	8(SB)	DF (-33)	FD (-3)
Case 4:	4(SB)	0F (+15)	0F (+15)
	8(SB)	DF (-33)	FD (-3)

RESTORE

Restore General Purpose Registers

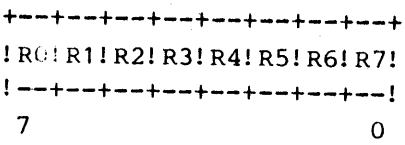
Syntax: **RESTORE reglist**
imm



The RESTORE Registers instruction restores from the current stack the general purpose registers specified by reglist.

In assembly language, the reglist operand is specified as a list of zero or more general-purpose register names enclosed in brackets "[]". The instruction copies to each register in the list a double-word popped from the stack. Register names may appear in any order within reglist but must be separated by commas. Brackets are required even if no register names are given.

In the machine instruction, the reglist operand is encoded in an eight-bit field as shown below. Each bit in the field corresponds to one general-purpose register. When the instruction is executed, the instruction reads the bits in the field from right to left beginning with bit 0. If a bit is "0", the instruction ignores the corresponding register. If a bit is "1", it restores the corresponding register from the stack. Note that the binary format of the reglist operand is backward from the format of the reglist operand in the SAVE instruction; i.e., bit 0 corresponds to R7 instead of R0.



Flags Affected: None.

Traps: None

Example:

```
RESTORE  [R0, R2, R7]          72 A1
```

This instruction restores the contents of registers R0, R2, and R7 from the stack. The registers are restored in order beginning with register R7 and ending with R0.

Restore General Purpose Registers (continued)

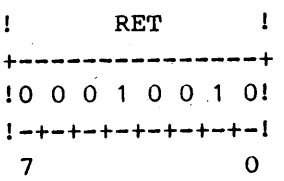
The action of the instruction is illustrated below.

Operands	Operand Values: Hex Before	After
R0	BBBBBBBB	00000010
R2	BBBBBBBB	FFFFFFEF
R7	BBBBBBBB	FFFFF9AB
SP	0000FFE0	0000FFEC
Stack:		
0000FFE0	FFFFF9AB	xxxxxxxx *
0000FFE4	FFFFFFEF	xxxxxxxx *
0000FFE8	00000010	xxxxxxxx *
0000FFEC	AAAAAAAA	AAAAAAAA

* The RESTORE instruction does not itself change the contents of these memory locations. However, information that is outside the stack should be considered unpredictable for other reasons. See Section 2.7.1.

RET
Return from Subroutine

Syntax: **RET** **constant**
disp



The RET instruction returns execution control from a local procedure and removes procedure parameters from the stack.

The instruction pops the return address as a 32-bit value from the currently-selected stack. It then removes the number of bytes specified by the constant operand from the stack by adding the constant operand to the current stack pointer register. Finally, it transfers control by loading the return address into the PC register.

Flags Affected: None.

Traps: None

Return from Subroutine (continued)

Example:

```
RET 16                                    12 10
```

This example pops a new address from the currently-selected stack into the PC and adds 16 (H'10) to the stack pointer.

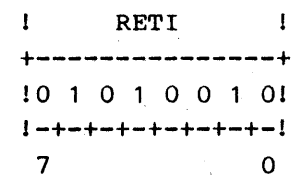
The instruction is illustrated below:

Operand	Operand Values: Hex Before	After
16 (disp)	10 (+16)	--
PC	00009000	00009010
SP	0000F000	0000F014
Stack:		
0000F000	00009010	xxxxxxxx *
0000F004	BBBBBBBB	xxxxxxxx *
0000F008	BBBBBBBB	xxxxxxxx *
0000F00C	BBBBBBBB	xxxxxxxx *
0000F010	BBBBBBBB	xxxxxxxx *
0000F014	AAAAAAAA	AAAAAAAA

* The RET instruction does not itself change the contents of these memory locations. However, information that is outside the stack should be considered unpredictable for other reasons. See Section 2.7.1.

RETI
Return from Interrupt

Syntax: **RETI**



The RETI instruction returns control from an interrupt service procedure to the program during which the interrupt was accepted, and informs any interrupt control circuitry present in the system that this is being done. See Chapter 6 for details of interrupt service.

The RETI instruction does the following:

1. Performs either one or two "End of Interrupt" bus cycles in order to inform the appropriate Interrupt Controller(s) that this interrupt service procedure is ending. For details of this aspect of the RETI instruction, see Chapter 6 and the data sheets for the NS16202 Interrupt Control Unit and the appropriate CPU.

2. Pops a 32-bit return address from the currently selected stack into the PC register.

3. Pops a 16-bit MOD address from the currently selected stack into the MOD register.

4. Pops a 16-bit PSR value from the currently selected stack into the PSR.

5. Copies the double-word from the memory address contained in the MOD register into the SB register.

Program execution continues at the new address placed in the PC register.

NOTE: The RETI instructon must not be used to return from the Non-Maskable or Non-Vectored interrupts or from any traps (including the Abort trap). Such use can cause anomalies in prioritization of interrupts by Interrupt Control circuits. For these use instead the Return from Trap instruction (RETT, q.v.).

Flags Affected: All flag states are restored from the stack.

Traps: Illegal Instruction Trap (ILL) is activated if this instruction is attempted while the U flag is set.

Return from Interrupt (continued)

Example:

```
RETI                    52
```

This example returns control from an interrupt service procedure.

The action of the above instruction is illustrated below. Note that the PSR S flag is assumed to be zero at the beginning of the instruction, thus selecting SP0 as the current Stack Pointer. However, note also that after the instruction is completed the CPU is in User mode, the currently-selected Stack Pointer has become SP1, and interrupts are re-enabled.

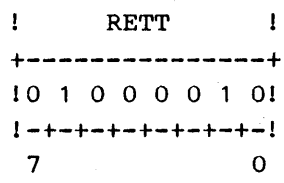
	Operand Values: Hex	
Operands	Before	After
PC	0000F033	00009005
SB	0000F100	00009080
MOD	0020	0010
SP0	00001000	00001008 *
PSR	x000 (xxxxipsu/nzfxxltc)	xB20 (xxxx1011/001xx000)
Stack:		
00001000	00009005	xxxxxxxx **
00001004	0010	xxxx **
00001006	0B20	xxxx **
00001008	AAAA	AAAA
Module Table:		
00000010	00009080 (SB)	00009080 (SB)

* The final Stack Pointer value is the initial address plus 8 as follows: 4 (for double-word return address), 2 (for MOD address), and 2 (for PSR contents).

** The RETI instruction does not itself change the contents of these memory locations. However, information that is outside the stack should be considered unpredictable for other reasons. See Section 2.7.1.

RETT
Return from Trap

Syntax: **RETT** **constant**
disp



The RETT instruction returns control from a trap service procedure. It restores the PC, MOD and PSR registers from the currently-selected stack, updates the SB register, and then removes any parameters passed by the procedure which caused the trap.

The instruction does the following:

1. Pops a 32-bit return address from the currently-selected stack into the PC register.

2. Pops a 16-bit MOD address from the currently-selected stack into the MOD register.

3. Pops a 16-bit PSR value from the currently-selected stack into the PSR. Note that this may switch stack pointers by changing the PSR S bit.

4. Copies the double-word from the address contained in the MOD register into the SB register.

5. Adds the constant operand to the stack pointer newly selected in step 3.

Program execution continues at the new address placed in the PC register.

For a full description of traps, see "Exceptions", Chapter 6.

NOTE: When using the NS16202 Interrupt Control Unit, the RETT instruction must not be used to return from a vectored interrupt, since this instruction does not inform the Interrupt Control Unit that it is returning from an interrupt. To return properly from a vectored interrupt, use the RETI instruction.

Flags Affected: All flag states are restored from the stack.

Traps: Illegal Instruction Trap (ILL) is activated if the U flag is set.