Return from Trap (continued)

Example:

```
RETT  16                                42 10
```

This example returns control from a trap service procedure to a procedure which invoked the trap deliberately after pushing 16 bytes of parameters onto its stack. This instruction removes the 16 bytes from that stack as it returns control.

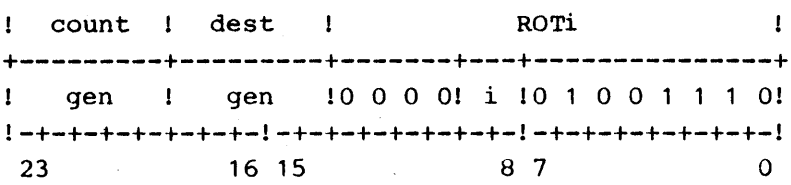
In the following illustration, it is assumed that the trap service routine is using the Interrupt Stack (with SP0 as its stack pointer) and is returning to a procedure which is using the User Stack (with SP1).

		Operand Values: Hex	
	Operands	Before	After
	16 (disp)	10	--
	PC	0000F033	00009005
	SB	0000F100	00009080
	MOD	0020	0010
	SP0	00001018	00001020
	SP1	0000FFE0	0000FFF0
	PSR	x000 (xxxxipsu/nzfxxltc)	x320 (xxxx0011/001xx000)
Interrupt Stack	00001018	00009005	xxxxxxxx *
	0000101C	03200010	xxxxxxxx *
	00001020	AAAAAAAA	AAAAAAAA
User Stack	0000FFE0	BBBBBBBB	xxxxxxxx *
	0000FFE4	BBBBBBBB	xxxxxxxx *
	0000FFE8	BBBBBBBB	xxxxxxxx *
	0000FFEC	BBBBBBBB	xxxxxxxx *
	0000FFF0	CCCCCCCC	CCCCCCCC
Module Table	00000010	00009080 (SB)	00009080 (SB)
	00000014	00002000 (LB)	00002000 (LB)
	00000018	00009000 (PB)	00009000 (PB)

* The RETT 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.

ROTi
Rotate

Syntax: **ROTi** **count,** **dest** ROTB
gen gen ROTW
read.B rmw.i ROTD



The ROTi instruction performs a rotation shift on the dest operand in the manner specified by the count operand. The sign of count determines the direction of the shift. The absolute value of count gives the number of bit positions to shift the dest operand.

The count operand value must be within the range -7 to +7 for the ROTB form, -15 to +15 for the ROTW form, and -31 to +31 for the ROTD form. A positive count specifies a left shift; a negative count specifies a right shift. In a rotation, each bit rotated off one end of dest is moved to the emptied bit position at the other end of dest.

The count operand is interpreted as a signed integer. The dest operand is interpreted as an unsigned integer.

Flags Affected: None.

Traps: None.

Rotate (continued)

Examples:

```
1.  ROTB   4, R5              4E 40 A1 04
2.  ROTB  -3, 16(SP)          4E 40 A6 FD 10
```

Example 1 rotates the least-significant byte of register R5 four bit positions to the left. The remaining bytes of R5 are unaffected.

Example 2 rotates the operand at address 16(SP) three bit positions to the right.

These instructions are illustrated below:

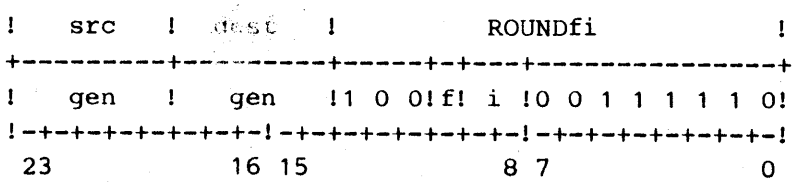
	Operands	Operand Values: Before	Binary (Dec) After
Ex. 1:	4 (immediate)	00000100 (+4)	--------
	R5 (low byte)	00001111	11110000
Ex. 2:	-3 (immediate)	11111101 (-3)	--------
	16(SP)	00001111	11100001

ROUNDfi

Round Floating to Integer

Syntax: **ROUNDfi** **src,** **dest**

gen	gen	ROUNDFB ROUNDLB
		ROUNDFW ROUNDLW
read.f	write.i	ROUNDFD ROUNDLD



The Round Floating to Integer instruction rounds the src operand to the nearest integer and places the result in the dest operand location as a signed integer. If src is exactly halfway between two integer values, it is rounded to the even value (i.e. the value that is exactly divisible by two).

Flags Affected: No PSR flags. One FSR flag may be affected:
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 to this instruction is the Overflow exception, which is caused by attempting to convert a floating-point number that is too great in absolute value to be held in a signed integer of the size specified for dest.

Round Floating to Integer (continued)

Examples:

```
1.  ROUNDFB  F0, R0          3E 24 00
2.  ROUNDLD  F2, 12(SB)      3E A3 16 0C
```

Example 1 rounds the single-precision number in register F0 to a one-byte integer and places the result in the low-order byte of register R0. The remaining bytes of R0 are unaffected.

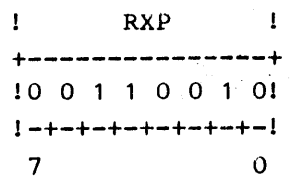
Example 2 rounds the double-precision number in register pair (F2,F3) to a double-word integer and places the result at address 12(SB).

These instructions are illustrated below:

	Operands	Operand Values: Hex (Dec) Before	After
Ex. 1:	F0	40180000 (+2.375)	40180000 (+2.375)
	R0	AAAAAAAA	AAAAAA02 (+2)
Ex. 2:	(F2,F3)	41C0200888700000 (+541069584.875)	41C0200888700000 (+541069584.875)
	12(SB)	AAAAAAAA	20401111 (+541069585)

RXP
Return from External Procedure

Syntax: **RXP** **constant**
disp



The RXP instruction returns control from an externally-called procedure and removes any procedure parameters from the stack.

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 to the MOD register and increments the stack pointer by two. The stack pointer is modified by a total of four in this step.

3. Copies the double-word at address MOD+0 to the SB register.

4. Adds the constant operand to the current stack pointer.

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

Flags Affected: None.

Traps: None.

Return from External Procedure (continued)

Example:

```
RXP  16                          32 10
```

This example returns control from an externally-called procedure and removes 16 (H'10) bytes from the currently-selected stack.

The instruction is illustrated below:

Operands	Operand Values: Hex Before	After
16 (disp)	10 (+16)	--
PC	0000F033	00009005
SB	0000F100	00009080
MOD	0020	0010
SP	00001018	00001030 *
Stack:		
00001018	00009005	xxxxxxxx **
0000101C	xxxx0010	xxxxxxxx **
00001020	BBBBBBBB	xxxxxxxx **
00001024	BBBBBBBB	xxxxxxxx **
00001028	BBBBBBBB	xxxxxxxx **
0000102C	BBBBBBBB	xxxxxxxx **
00001030	AAAAAAAA	AAAAAAAA
Module Table:		
00000010	00009080 (SB)	00009080 (SB)
00000014	00002000 (LB)	00002000 (LB)
00000018	00009000 (PB)	00009000 (PB)

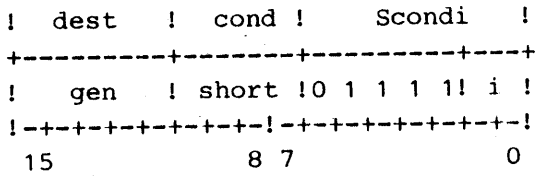
* The final Stack Pointer content is the initial address plus 4 (for double-word return address), 2 (for word MOD address), 2 (additional from step 3), and 16 as specified by the constant operand.

** The RXP 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.

Scondi
Save Condition as Boolean

Syntax: **Scondi** **dest**
gen
write.i

ScondB
ScondW
ScondD



The Scondi instruction sets the dest operand to the integer value "1" if the specified condition is true, and to "0" if false. These are the Boolean values "True" and "False", respectively.

Cond is a two-character condition name that specifies the state of a flag or flags in the PSR. If the flag(s) have the specified state, the condition is true; otherwise, the condition is false.

The Save Condition as Boolean instruction may specify the following conditions:

Condition	Condition Name	True State	Short Field
Equal	EQ	Z flag set	0000
Not Equal	NE	Z flag clear	0001
Carry Set	CS	C flag set	0010
Carry Clear	CC	C flag clear	0011
Higher	HI	L flag set	0100
Lower or Same	LS	L flag clear	0101
Greater Than	GT	N flag set	0110
Less Than or Equal	LE	N flag clear	0111
Flag Set	FS	F flag set	1000
Flag Clear	FC	F flag clear	1001
Lower	LO	Z and L flag clear	1010
Higher or Same	HS	Z or L flag set	1011
Less Than	LT	Z and N flag clear	1100
Greater than or Equal	GE	Z or N flag set	1101

The condition name must be embedded in the instruction mnemonic as illustrated in the examples below. The name is translated at assembly time to the corresponding Short Field of the basic instruction.

The interpretation of condition codes is such that the instruction sequence

```
CMPB    A,B
SGTB    RESULT
```

will store "1" in RESULT if operand A is greater than operand B in the CMPB instruction.

Save Condition as Boolean (continued)

Flags Affected: None.

Traps: None.

Examples:

```
1.  SEQB  R0          3C 00
2.  SLOW  10(SB)      3D D5 0A
3.  SHID  TOS         3F BA
```

Example 1 sets the low-order byte of register R0 to 1 if the Z flag is set, 0 if the Z flag is clear.

Example 2 sets the word at the operand address 10(SB) to 1 if the Z and L flags are clear, 0 otherwise.

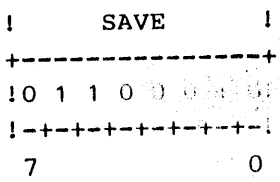
Example 3 pushes a double-word value onto the stack: 1 if the L flag is set, 0 otherwise.

In the following illustration, the Z and L flags are assumed to be set.

		Operand Values: Hex (Boolean)	
	Operands	Before	After
Ex. 1:	R0	AAAAAAAA	AAAAAA01 (True)
	UPSR	n1fxx1tc	n1fxx1tc
Ex. 2:	10(SB)	AAAA	0000 (False)
	UPSR	n1fxx1tc	n1fxx1tc
Ex. 3:	Stack:		
	00001000	xxxxxxxx	00000001 (True)
	00001004	AAAAAAAA	AAAAAAAA
	SP	00001004	00001000
	UPSR	n1fxx1tc	n1fxx1tc

SAVE
Save General Purpose Registers

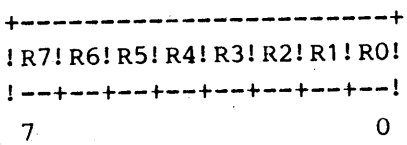
Syntax: **SAVE** **reglist**
imm



The SAVE instruction saves the general purpose registers specified by reglist, pushing them onto the currently-selected stack.

The reglist operand is a list of zero or more general purpose register names, enclosed in brackets "[]". The instruction pushes the contents of each register in the list as a double-word onto 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 8-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 pushes the corresponding register.



Flags Affected: None.

Traps: None.

Example:

```
SAVE  [R0, R2, R7]                    62 85
```

This instruction saves the contents of registers R0, R2, and R7 on the stack. The registers are stored in order beginning with register R0 and ending with R7.

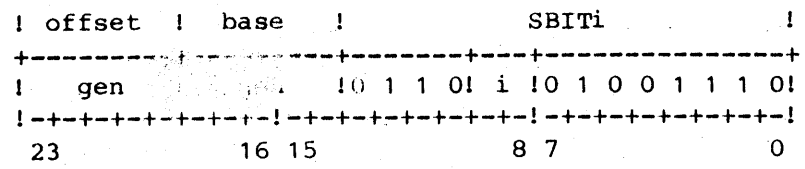
Save General Purpose Registers (continued)

The instruction is illustrated below:

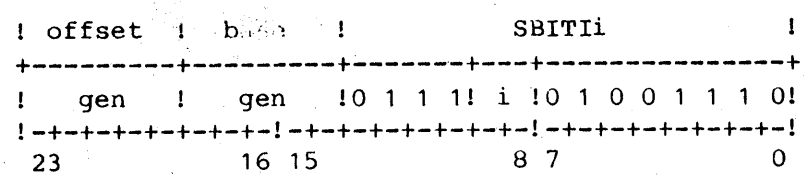
Operands	Operand Values: Hex Before	After
R0	00000010	00000010
R2	FFFFFFEF	FFFFFFEF
R7	FFFFF9AB	FFFFF9AB
SP	0000FFEC	0000FFE0
Stack:		
0000FFE0	xxxxxxxx	FFFFF9AB
0000FFE4	xxxxxxxx	FFFFFFEF
0000FFE8	xxxxxxxx	00000010
0000FFEC	AAAAAAAA	AAAAAAAA

SBITi
SBITIi
Set Bit, Set Bit Interlocked

Syntax: **SBITi** **offset, base** SBITB SBITIB
gen gen SBITW SBITIW
read.i regaddr SBITD SBITID



Syntax: **SBITIi** **offset, base**
gen gen
read.i regaddr



The SBITi and SBITIi instructions set to 1 the register or memory bit specified by base and offset after copying the bit value to the F flag in the PSR.

The SBITIB, SBITIW, and SBITID instructions, in addition, activate the Interlocked Operation output pin on the CPU, which may be used in multi-processor systems to interlock accesses to semaphore bits. See the applicable CPU data sheet for further details.

The location of the bit is determined from offset and base. Offset is a general operand, whose length is given by the operation length suffix. Base is an addressing expression giving a byte address from which offset specifies a bit position. See Section 3.5 for details of specifying bit positions.

If base is a register, then the bit is within that register, at the bit position given by the offset operand. If base is a memory location, then the bit is at bit position

offset MOD 8

within the memory byte whose address is

EA(base) + (offset DIV 8),

where EA(base) is the effective address of base. See Section 3.5 for definitions of the operators MOD and DIV above, and for further details of bit instructions.

Offset is interpreted as a signed integer.

Set Bit (continued)

Flags Affected: F is set to the original value of the specified bit.

Traps: None.

Example:

```
SBITW  R0, 1(R1)                    4E 59 02 01
```

This example sets a bit in memory to 1 after copying the bit value to the F flag. This performs the basic operation of a semaphore "Test and Set". In a multiprocessor system, the SBITIW instruction would have been used instead. For designating the location of the target bit, the low-order word of register R0 supplies the bit offset, and 1(R1) is specified as the base address.

In the following illustration, the target bit is assumed to be 0 prior to instruction execution.

Operands	Operand Values: Hex (Dec) [Binary] Before	After
R0 (offset)	AAAA004C (+76)	AAAA004C (+76)
R1	00001000 (+4096)	00001000 (+4096)
base address 1(R1)	00001001 (+4097)	--
0000100A * (+4106)	EF [11101111]	FF [11111111]
UPSR	nzfxxltc	nz0xxltc

* The address 100A (Hex) is the effective address of the byte containing the desired bit. This address is computed from the offset and the base address as follows:

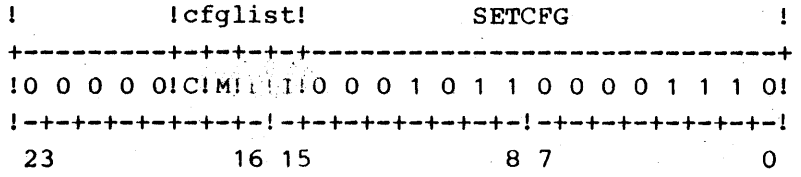
```
base address + (offset DIV 8)
   4097      +       9
   4106, or 100A (Hex)  .
```

The bit number within this byte is calculated as:

```
offset MOD 8
  76   MOD 8
     4  .
```

SETCFG
Set Configuration

Syntax: **SETCFG** **cfglist**
short



The SETCFG instruction loads the Configuration Register (CFG), enabling or disabling optional system features.

In assembly language, cfglist is a list of zero or more configuration bit names. The names are I, F, M and C. The names may appear in any order in the list, but must be separated by commas. The list itself must be enclosed in brackets. Brackets are required even if no bit name is given.

The cfglist operand is held in a 4-bit field in the basic instruction, as shown above. Each bit corresponds to one bit in the CFG register.

If I is specified, the I bit in the CFG register is set and vectored interrupt processing is enabled. (An NS16202 Interrupt Control Unit or equivalent must be present in the system.) Otherwise, the I bit is cleared and all maskable interrupts are serviced as non-vectored interrupts. See Chapter 6 for interrupt handling modes.

If F is specified, the F bit in the CFG register is set and Floating Point instructions are available. (An NS16081 Floating-Point Unit must be present in the system.) Otherwise, the F bit is cleared and Floating Point instructions activate the Undefined Instruction Trap (UND).

If M is specified, the M bit in the CFG register is set and Memory Management instructions are available. (An NS16082 Memory Management Unit must be present in the system.) Otherwise, the M bit is cleared and Memory Management instructions activate the Undefined Instruction Trap (UND).

If C is specified, the C bit in the CFG register is set and Custom Slave instructions are available. (System-dependent Custom Slave hardware must be present to execute the instructions.) Otherwise, the C bit is cleared and Custom Slave instructions activate the Undefined Instruction Trap (UND).

Set Configuration (continued)

NOTES: 1. A CFG bit name may only be specified if the corresponding device is present in the system.

2. The state of the M bit in the CFG register does not directly affect address translation hardware or bus timing. It only enables or disables the Memory Management instruction set.

3. When a Floating Point, Memory Management, or Custom Slave instruction activates an Undefined Instruction Trap (UND) (i.e, when the corresponding CFG register bit is 0), it is possible to intercept the trap and simulate the instruction in software.

Flags Affected: None.

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

Example:

```
SETCFG  [I,M,F]                    0E 8B 03
```

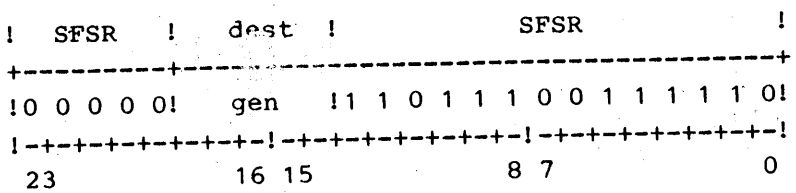
This instruction sets the I, M, and F bits in the CFG register, enabling vectored interrupt processing and the Memory Management and Floating Point instruction sets. The C bit is cleared, disabling the Custom Slave instruction set.

The instruction is illustrated below:

	Bit Values	
Operands	Before	After
CFG	cmfi	0111

SFSR
Store Floating-Point Status Register (FSR)

Syntax: **SFSR** **dest**
gen
write.D



The SFSR instruction copies the contents of the Floating Point Status Register (FSR) to the dest operand location. The FSR is treated as a 32-bit value. See Section 2.4.2 for the format of the FSR.

Flags Affected: None.

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

Example:

```
SFSR  TOS                              3E F7 05
```

This example pushes the contents of the FSR onto the top of the currently selected stack as a double word.

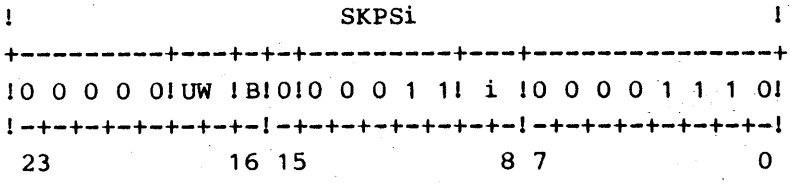
The instruction is illustrated below:

Operands	Operand Values: Hex Before	After
FSR	xxxx0028	xxxx0028
SP	0000FFDE	0000FFDA
Stack:		
0000FFDA	xxxxxxxx	xxxx0028
0000FFDE	AAAAAAAA	AAAAAAAA

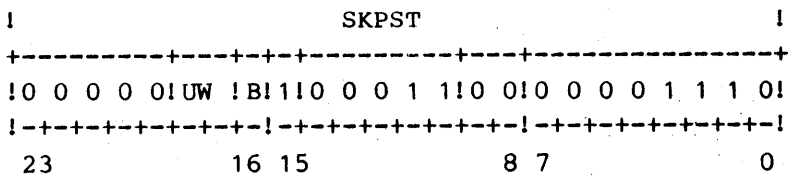
Skip String

Syntax: **SKPSi** **options**

SKPSB
SKPSW
SKPSD
SKPST



Syntax: **SKPST** **options**



Operands of the SKPSi and SKPST instructions are specified in General-Purpose Registers:

R0 - Number of string elements to be processed.
R1 - Address of current String 1 element.
R2 - (not used)
R3 - Address of translation table (SKPST form only).
R4 - Match value (with Until Match or While Match option only).

The SKPSi instruction examines and skips over consecutive elements in String 1 until either an Until/While condition is met or register R0 is decremented to 0 (i.e., the string is exhausted). After each element is examined, the CPU sets register R1 to the address of the next element to be examined and register R0 to the number of integers remaining to be examined. Register R2 is not used or affected.

The SKPST instruction causes the CPU to internally replace the current String 1 element value with a corresponding translated value before performing its examination. The translated value to be examined 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 examines elements and sets registers as described above. The SKPST instruction operates on byte-long elements only.

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.

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:

```
SKPSB   U                                   0E 0C 06
```

This example examines and skips over byte-long elements in String 1 until the current integer and the contents of the low-order byte of register R4 are equal or until register R0 contains zero.

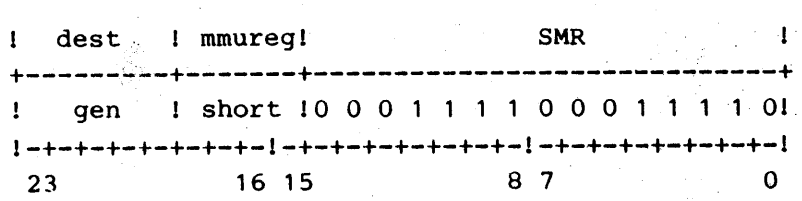
In the following illustration, the underlined string element shows the point at which the instruction terminates.

Operands	Operand Values: Hex (Dec) Before	After
R0	00000020 (+32)	00000016 (+22)
R1	00002000	0000200A
R4	AAAAAA1F (+31)	AAAAAA1F (+31)
UPSR	nzfxxltc	nz1xxltc

Starting Address	String Contents
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

Store Memory Management Register

Syntax: **SMR** **mmureg, dest**
short gen
write.D



The SMR instruction copies the contents of the Memory Management register specified by mmureg to the dest operand location.

The Store MMU Register instruction may store the following registers. The short field of the basic instruction holds a four-bit value which relates to the corresponding mmureg specifications as follows:

Register	mmureg	short field
Breakpoint Register 0	BPR0	0000
Breakpoint Register 1	BPR1	0001
Program Flow Register 0	PF0	0100
Program Flow Register 1	PF1	0101
Sequential Count Registers	SC	1000
Memory Management Status Register	MSR	1010
Breakpoint Count Register	BCNT	1011
Page Table Base Register 0	PTB0	1100
Page Table Base Register 1	PTB1	1101
Error/Invalidate Address Register	EIA	1111

Flags Affected: None.

Traps: Undefined Instruction Trap (UND) is activated if the M bit in the CFG register is clear. The instruction is not executed.

Illegal Instruction Trap (ILL) is activated if the U flag is set. The instruction is not executed.

Example:

```
SMR  BCNT, R0                    1E 8F 05
```

This example copies the contents of the Breakpoint Count Register in the MMU to register R0.

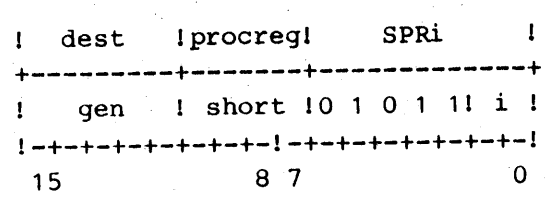
The instruction is illustrated below:

Operands	Operand Values: Hex Before	After
BCNT	xx009000	xx009000
R0	AAAAAAAA	xx009000

Store Processor Register

Syntax: **SPRi** procreg, **dest**
short gen
write.i

SPRB
SPRW
SPRD



The SPRi instruction stores the contents of the dedicated processor register specified by procreg in the dest operand location.

The register contents are right-justified in dest. High-order dest bits are zero-filled if the register is shorter than dest. High-order register bits are discarded if the register is longer than dest. See Section 2.2 for the formats of the dedicated registers.

The Store Processor Register instruction may store the following registers. The specified procreg corresponds to the 4-bit short field in the basic instruction as shown below.

Register	procreg	short field	
User PSR	UPSR	0000	(Note 2)
Frame Pointer	FP	1000	
Stack Pointer	SP	1001	(Note 1)
Static Base Register	SB	1010	
Program Status Register	PSR	1101	(Note 2)
Interrupt Base Register	INTBASE	1110	(Note 2)
Module Register	MOD	1111	

NOTES:
1. If SP is specified in the instruction and the S flag in the PSR is set, the instruction copies the SP1 register to the dest operand location. If the S flag is clear, the instruction copies the SP0 register to the dest operand location.

2. Access to the PSR or INTBASE register is privileged. If the U flag in the PSR is "1" when this is attempted, the Illegal Operation trap (ILL) occurs instead. If UPSR is specified, however, only the low-order eight bits of the PSR are affected. This is not a privileged operation, and no trap occurs.

Flags Affected: None.

Traps: Illegal Instruction Trap (ILL) is activated if the U flag is set and PSR or INTBASE is specified.

Examples:

```
1.  SPRD  FP, R0          2F 04
2.  SPRW  MOD, 4(SB)      AD D7 04
```

Example 1 copies the entire FP register to register R0.

Example 2 copies the contents of the MOD register to a word at the address specified by 4(SB).

The instructions are illustrated below:

	Operands	Operand Values: Hex Before	After
Ex. 1:	FP	00000021	00000021
	R0	AAAAAAAA	00000021
Ex. 2:	MOD	0030	0030
	4(SB)	AAAA	0030

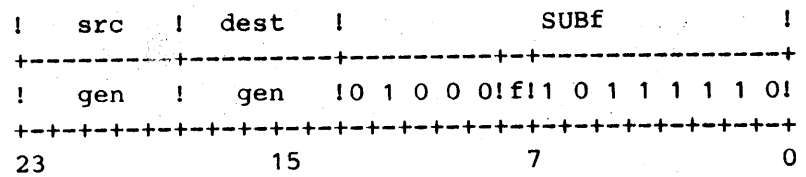
Subtract Floating

Syntax: **SUBf** **src,** **dest**

	gen
	read.f

	gen
	rmw.f

SUBF
SUBL



The SUBf instruction subtracts the src operand from the dest operand and places the result in the dest operand location. Subtraction can be modelled as negating the src operand and adding the result to the dest operand. For details of the addition step see the ADDf instruction.

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.	SUBF F0, F7	BE D1 01
2.	SUBL F2, 16(SB)	BE 90 16 10

Example 1 subtracts the single-precision number in register F0 from the single-precision number in register F7 and places the result in register F7.

Example 2 subtracts the double-precision number in register pair (F2,F3) from the double-precision number at the address 16(SB) and places the double-precision result at the address 16(SB).

These instructions are illustrated below:

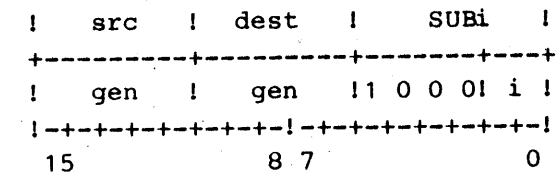
	Operands	Operand Values: Hex (Dec) Before	After
Ex. 1:	F0	40840000 (+4.125)	40840000 (+4.125)
	F7	41F50000 (+30.625)	41D40000 (+26.500)
Ex. 2:	(F2,F3)	4114C86300000000 (+340504.750)	4114C86300000000 (+340504.750)
	16(SB)	41C022A194900000 (+541410089.125)	41C0200888300000 (+541069584.375)

Subtract

Syntax: **SUBi** **src,** **dest**

gen	gen
read.i	rmw.i

SUBB
SUBW
SUBD



The SUBi instruction subtracts the src operand from the dest operand and places the result in the dest operand location.

Flags Affected: C is set on a borrow from subtraction, cleared if no borrow. F is set on an overflow from subtraction, cleared if no overflow.

Integer carry and overflow conditions are defined in Section 3.1.

Traps: None.

Examples:

1.	SUBB R0, R1	60 00
2.	SUBD 4(SB), 20(SB)	A3 D6 04 14

Example 1 subtracts the low-order byte of register R0 from the low-order byte of register R1 and places the result in the low-order byte of register R1. The remaining bytes of R1 are not affected.

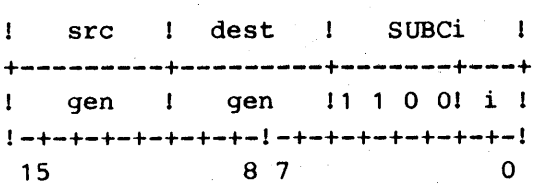
Example 2 subtracts the double-word at the memory address specified by 4(SB) from the double-word at the memory address specified by 20(SB). The instruction places the result at memory address 20(SB).

These instructions are illustrated below:

	Operands	Operand Values: Hex (Dec) Before	After
Ex. 1:	R0	AAAAAA01 (+1)	AAAAAA01 (+1)
	R1	BBBBBB7F (+127)	BBBBBB7E (+126)
	UPSR	nzfxxltc	nz0xxlt1
Ex. 2:	4(SB)	FFFFFFFE (-2)	FFFFFFFE (-2)
	20(SB)	00010000 (+65536)	00010002 (+65538)
	UPSR	nzfxxltc	nz0xxlt1

Subtract with Carry [Borrow]

Syntax: **SUBCi** **src,** **dest** SUBCB
gen gen SUBCW
read.i rmw.i SUBCD



The SUBCi instruction subtracts the sum of the src operand and the C flag from the dest operand and places the result in the dest operand location.

Flags Affected: C is set on a borrow from subtraction, cleared if no borrow. F is set on an overflow from subtraction, cleared if no overflow.

Integer carry and overflow conditions are defined in Section 3.1.

Traps: None.

Examples:

```
1.  SUBCB  32, R1                70 A0 20
2.  SUBCW  TOS, -8(FP)           31 BE 78
```

Example 1 subtracts the sum of 32 and the C flag value from the low-order byte of register R1 and places the result in the low-order byte of register R1. The remaining bytes of R1 are not affected.

Example 2 subtracts the sum of the word at the top of the stack and the C flag value from the word at the memory address specified by -8(FP). The instruction then places the two-byte result at the memory address specified as -8(FP).

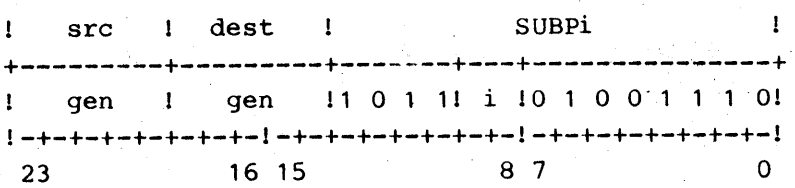
In the following illustration, the C flag value is assumed to be 1.

	Operands	Operand Values: Hex (Dec) Before	After
Ex. 1:	32 (immediate)	20 (+32)	--
	R1	00000050 (+80)	0000002F (+47)
	UPSR	nzfxxlt1	nz0xxlt0
Ex. 2:	-8(FP)	CB99 (-13415)	9286 (-28026)
	UPSR	nzfxxlt1	nz0xxlt0
	Stack:		
	0000FFEE	3912 (+14610)	xxxx *
	0000FFF0	AAAA	AAAA
	SP	0000FFEE	0000FFF0

* The instruction has not itself changed 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.

Subtract Packed Decimal

Syntax: **SUBPi** **src,** **dest** SUBPB
gen gen SUBPW
read.i rmw.i SUBPD



The SUBPi instruction subtracts the src operand from the dest operand and then subtracts the C flag. The instruction places the result in the dest operand location as a packed decimal (BCD) integer.

The src and dest operands are interpreted as unsigned packed decimal (BCD) integers. If either operand contains invalid digits, the result is undefined. See Section 3.2 for details of packed decimal arithmetic.

Flags Affected: C is set on a borrow from subtraction, cleared if no borrow.
F is cleared.

Traps: None.

Examples:

```
1.  SUBPD  H'99, R1              4E 6F A0 00 00 00 99
2.  SUBPB  -8(FP), 16(FP)        4E 2C C6 78 10
```

Example 1 subtracts the packed decimal integer 99 from the contents of register R1 and then subtracts the C flag. The result is placed in resister R1.

Example 2 subtracts the packed decimal integer at memory address -8(FP) from the packed decimal integer at memory address 16(FP) and then subtracts the C Flag. The instruction places the result at memory address 16(FP).

In the following illustration, the C flag value is assumed to be 0.

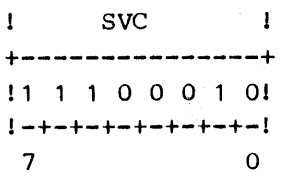
	Operands	Operand Values: Hex * Before	After
Ex. 1:	H'99 (immediate)	00 00 00 99	--
	R1	00000187	00000088
	UPSR	nzfxxlt0	nz0xxlt0
Ex. 2:	-8(FP)	10	10
	16(FP)	01	91 **
	UPSR	nzfxxlt0	nz0xxlt1

* The hexadecimal representation also expresses the decimal interpretation of the value.

** In example 2, subtraction results in a borrow.

Supervisor Call

Syntax: **SVC**



The SVC instruction activates the Supervisor Call Trap (SVC). The Supervisor Call Trap passes program execution control to the SVC service procedure (see "Exceptions", Chapter 6.) The return address pushed onto the Interrupt Stack is the address of the SVC instruction itself.

Flags Affected: None.

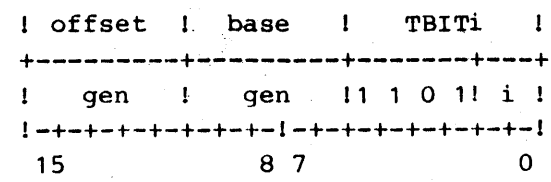
Traps: Supervisor Call Trap (SVC) is activated.

Example:

```
SVC                                E2
```

TBITi
Test Bit

Syntax: **TBITi** **offset,** **base** TBITB
gen gen TBITW
read.i regaddr TBITD



The TBITi instruction copies the register or memory bit specified by base and offset to the F flag.

The location of the bit is determined from offset and base. Offset is a general operand, whose length is given by the operation length suffix. Base is an addressing expression giving a byte address from which offset specifies a bit position. See Section 3.5 for details of specifying bit positions.

If base is a register, then the bit is within that register, at the bit position given by the offset operand. If base is a memory location, then the bit is at bit position

offset MOD 8

within the memory byte whose address is

EA(base) + (offset DIV 8),

where EA(base) is the effective address of base. See Section 3.5 for definitions of the operators MOD and DIV above, and for further details of bit instructions.

Offset is interpreted as a signed integer.

Flags Affected: F is set to the value of the specified bit.

Traps: None.

Test Bit (continued)

Example:

```
TBITW  R0, 0(R1)                    75 02 00
```

This example copies a bit from memory to the F flag. The low-order word of register R0 supplies the bit offset, and 0(R1) is specified as the base address.

In the following illustration, the target bit is assumed to be 1.

Operands	Operand Values: Hex (Dec) [Binary] Before	After
R0 (offset)	AAAA004C (+76)	AAAA004C (+76)
R1	00001000 (+4096)	00001000 (+4096)
base address 0(R1)	00001000 (+4096)	--
00001009 * (+4105)	10 [00010000]	10 [00010000]
UPSR	nzfxxltc	nz1xxltc

* The address 1009 (Hex) is the effective address of the byte containing the desired bit. This address is computed from the offset and the base address as follows:

```
base address + (offset DIV 8)
   4096      +       9
   4105, or 1009 (Hex)  .
```

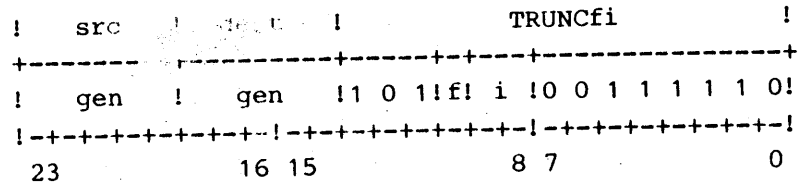
The bit number within this byte is calculated as:

```
offset MOD 8
  76   MOD 8
     4  .
```

TRUNCfi
Truncate Floating to Integer

Syntax: **TRUNCfi** **src,** **dest**
gen gen
read.f write.i

TRUNCFB TRUNCLB
TRUNCFW TRUNCLW
TRUNCFD TRUNCLD



The TRUNCfi instruction truncates the src operand to the nearest integer which is less than or equal to it in absolute value and places the result in the dest operand location as a signed integer.

Flags Affected: No PSR flags. One FSR flag may be affected:
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 to this instruction is the Overflow exception, which is caused by attempting to convert a floating-point number which is too great in absolute value to be held in a signed integer of the size specified for dest.

Truncate Floating to Integer (continued)

Examples:

```
1.  TRUNCFB  F0, R0          3E 2C 00
2.  TRUNCLD  F2, 8(SB)       3E AB 16 08
```

Example 1 truncates the single-precision number in register F0 to a one-byte integer and copies the integer to the low-order byte of register R0.

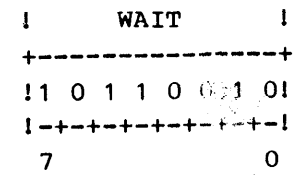
Example 2 truncates the double-precision number in register pair (F2,F3) to a double-word integer and copies the integer to address 8(SB).

These instructions are illustrated below:

		Operand Values: Hex (Dec)	
	Operands	Before	After
Ex. 1:	F0	C0280000 (-2.625)	C0280000 (-2.625)
	R0	AAAAAAAA	AAAAAAFE (-2)
Ex. 2:	(F2,F3)	41C0200888700000 (+541069584.875)	41C0200888700000 (+541069584.875)
	8(SB)	AAAAAAAA	20401110 (+541069584)

WAIT
Wait

Syntax: **WAIT**



The WAIT instruction suspends program execution until an interrupt occurs. An interrupt restores program execution by passing it to an interrupt service procedure. Interrupts are described in "Exceptions", Chapter 6. When the WAIT instruction is interrupted, the return address saved is the address of the instruction following the WAIT instruction.

Flag Affected: None.

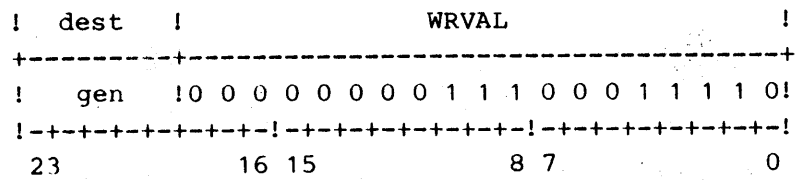
Traps: None.

Example:

```
WAIT                    B2
```

Validate Address for Writing

Syntax: **WRVAL** loc
gen
addr



The WRVAL instruction checks the protection level assigned to the user-mode virtual memory address specified as loc. If the address can be written to while in user mode, the F flag in the PSR is cleared. If the address cannot be written to (i.e., if loc is write-protected), the F flag in the PSR is set. See Section 3.12 for details of Memory Management instructions.

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 write-protected, 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 flag 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.

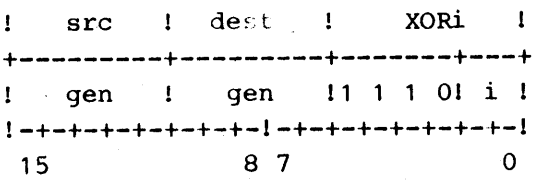
Example:

```
WRVAL 512(R0)                    1E 07 40 82 00
```

This example checks the protection level assigned to the user-mode virtual address 512(R0) and sets or clears the F flag to indicate the result.

XORi
Exclusive Or

Syntax: **XORi** **src,** **dest** XORB
gen gen XORW
read.i rmw.i XORD



The XORi instruction performs a bit-wise Exclusive-OR operation on the src and dest operands and places the result in the dest operand location.

The instruction XORs each bit of src with the corresponding dest bit. If two corresponding bits are equal, the dest bit is set to "0"; otherwise, the dest bit is set to "1".

Flags Affected: None.

Traps: None.

Example:

```
XORB  -8(FP), -4(FP)                    38 C6 78 7C
```

This example XORs the bytes at the addresses specified by -8(FP) and -4(FP) and places the result in the byte at address -4(FP).

The instruction is illustrated below:

Operands	Operand Values: Binary Before	After
-8(FP)	11110000	11110000
-4(FP)	10010101	01100101