

## 3 INSTRUCTIONS IN ALPHABETICAL ORDER

In this section, all the instructions of the CRIS CPU are described in alphabetical order. Each description contains the following information:

**Assembler syntax:** Shows the assembler syntax for the instruction. Operands, addressing modes and size modifiers are described using the definitions shown in section 2.1 *Definitions*. Note that instructions, operands etc. may be written in upper or lower case.

**Size:** Lists the different data sizes for the instruction.

**Operation:** Describes the instruction in a form similar to the C programming language. Different data sizes are shown with the “type cast” method used in the C language. The behavior of the flags is usually not shown.

**Description:** A text description of the instruction.

**Flags affected:** Shows which flags that are affected by the instruction. The detailed behavior of the flags is shown in section 1.2. *Flags and Condition Codes*.

**Instruction format:** Shows the instruction format. The format of the Addressing mode prefix word for the complex addressing modes is not shown here. This can be found in section 1.5. *Addressing Modes*, and in section 2.4 *Addressing Mode Prefix Formats*.

# ABS

Absolute Value

# ABS

**Assembler syntax:** ABS Rs, Rd

**Size:** Dword

**Operation:**

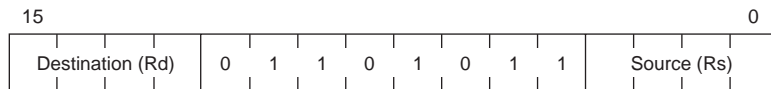
```
if (Rs < 0)
{
    Rd = -Rs;
}
else
{
    Rd = Rs;
}
```

**Description:** The absolute value of the contents of the source register is stored in the destination register. The size of the operation is dword.

**flags affected:**

F	P	U	M	B	I	X	N	Z	V	C
-	-	-	-	-	-	0	*	*	0	0

**Instruction format:**



**Note 1:** If the source operand is 0x80000000, the result of the operation will be 0x80000000

# ADD

2-operand

Add

# ADD

2-operand

**Assembler syntax:** ADD.m s,Rd

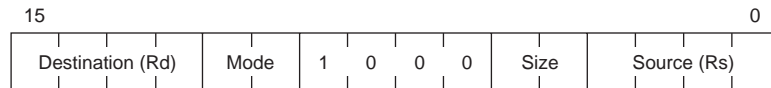
**Size:** Byte, word, or dword

**Operation:** (m)Rd += (m)s;

**Description:** The source data is added to the destination register. The size of the operation is m. The rest of the destination register is not affected.

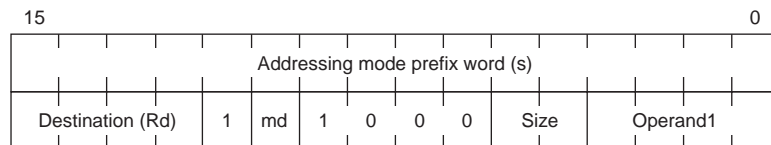
**flags affected:** F P U M B I X N Z V C  
 - - - - - 0 \* \* \* \*

**Instruction format:**  
 (register, indirect, or auto-increment addressing modes)



<b>Mode:</b>	01	Register addressing mode
	10	Indirect addressing mode
	11	Autoincrement addressing mode
<b>Size:</b>	00	Byte
	01	Word
	10	Dword

**Instruction format:**  
 (complex addressing modes)



<b>Mode (md):</b>	0	Indexed, offset, double indirect, absolute addressing modes. The Operand1 field must be the same as destination field (Rd).
	1	Indexed with assign, offset with assign addressing modes. The Operand1 field selects the register in which to store the address of the source operand.
<b>Size:</b>	00	Byte
	01	Word
	10	Dword

# ADD

3-operand

Add

# ADD

3-operand

**Assembler syntax:** `ADD.m se,Rn,Rd`

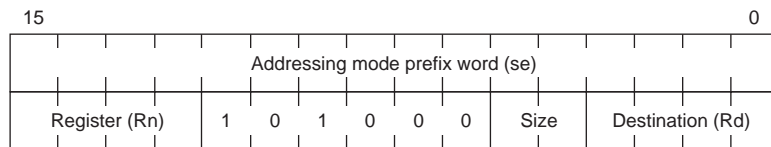
**Size:** Byte, word, or dword

**Operation:**  $(m)Rd = (m)se + (m)Rn;$

**Description:** The memory source data is added to the contents of a general register, and the result is stored in the destination register. The size of the operation is m. The rest of the destination register is not affected.

**flags affected:**  
 F P U M B I X N Z V C  
 - - - - - 0 \* \* \* \*

**Instruction format:**



<b>Size:</b>	00	Byte
	01	Word
	10	Dword

# ADDI

Add index

# ADDI

**Assembler syntax:** `ADDI Rs.m,Rd`**Size:** Rs is a pointer to byte, word or dword. The size of the operation is dword.**Operation:** `Rd += Rs * sizeof(m);`**Description:** Add a scaled index to a base. The contents of the source register is shifted left 0, 1 or 2 positions, depending on the size modifier m, and is then added to the destination register. The size of the operation is dword.**flags affected:**  
F P U M B I X N Z V C  
- - - - - 0 - - - -**Instruction format:**

<b>Size:</b>	00	Rs is pointer to Byte
	01	Rs is pointer to Word
	10	Rs is pointer to Dword

**Note 2:** PC is not allowed to be the base register.

# ADDQ

Add quick

# ADDQ

**Assembler syntax:** ADDQ j, Rd

**Size:** Source data is 6-bit. The size of the operation is dword

**Operation:** Rd += j;

**Description:** A 6-bit immediate value, zero extended to dword, is added to the destination register.

**flags affected:** F P U M B I X N Z V C  
- - - - - 0 \* \* \* \*

**Instruction format:**







# ADDU

Add with zero extend

# ADDU

2-operand

2-operand

**Assembler syntax:** `ADDU.z s,Rd`

**Size:** Source size is byte or word. Operation size is dword

**Operation:** `Rd += (unsigned z)s;`

**Description:** The source data is zero extended from z to dword, and then added to the destination register.

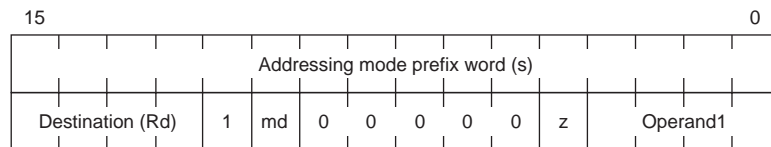
**flags affected:** F P U M B I X N Z V C  
 - - - - - 0 \* \* \* \*

**Instruction format:**  
 (register, indirect, or auto-increment addressing modes)



<b>Mode:</b>	01	Register addressing mode
	10	Indirect addressing mode
	11	Autoincrement addressing mode
<b>Size (z):</b>	0	Byte source operand
	1	Word source operand

**Instruction format:**  
 (complex addressing modes)



<b>Mode (md):</b>	0	Indexed, offset, double indirect, or absolute addressing modes. The Operand1 field must be the same as the Destination field (Rd).
	1	Indexed with assign, or offset with assign addressing modes. The Operand1 field selects the register in which to store the address of the source operand.
<b>Size (z):</b>	0	Byte source operand
	1	Word source operand







# ANDQ

Logical AND quick

# ANDQ

**Assembler syntax:** ANDQ *i*, *Rd*

**Size:** Source data is 6-bit. Operation size is dword.

**Operation:**  $Rd \ \&= \ i;$

**Description:** A logical AND is performed between a 6-bit immediate value, sign extended to dword, and the destination register.

**flags affected:**

F	P	U	M	B	I	X	N	Z	V	C
-	-	-	-	-	-	0	*	*	0	0

**Instruction format:**



# ASR

Arithmetic shift right

# ASR

**Assembler syntax:** ASR.m Rs,Rd

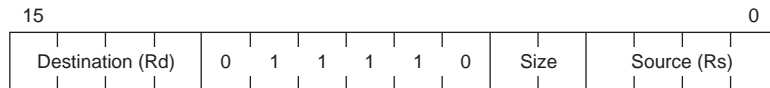
**Size:** Byte, word, or dword

**Operation:** (m)Rd >>= (Rs & 63);

**Description:** The destination register is right shifted the number of steps specified by the 6 least significant bits of the source register. The shift is performed with sign extend. The size of the operation is m. The rest of the destination register is not affected.

**flags affected:** F P U M B I X N Z V C  
 - - - - - 0 \* \* 0 0

**Instruction format:**



<b>Size:</b>	00	Byte
	01	Word
	10	Dword

**Note 3:** A shift of 32 bits or more will produce the same result as shifting the destination register 31 bits.

# ASRQ

Arithmetic shift right quick

# ASRQ

**Assembler syntax:** ASRQ c, Rd

**Size:** Dword

**Operation:** Rd >>= c;

**Description:** The destination register is right shifted the number of steps specified by the 5-bit immediate value. The shift is performed with sign extend. The size of the operation is dword.

**flags affected:** F P U M B I X N Z V C  
 - - - - - 0 \* \* 0 0

**Instruction format:**



**AX**

Arithmetic extension

**AX**

**Assembler syntax:** AX

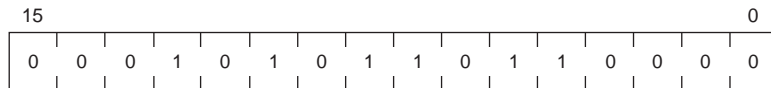
**Size:** -

**Operation:** X = 1;

**Description:** Arithmetic extension prefix. Set X flag. Disable interrupts until next instruction. This is a predefined assembler macro equivalent to SETF X.

**flags affected:** F P U M B I X N Z V C  
- - - - - 1 - - - -

**Instruction format:**



# Bcc

Branch conditionally

# Bcc

**Assembler syntax:** Bcc o  
Bcc xx

**Size:** Byte, Word

**Operation:** if (cc)  
{  
    PC += offset; offset = o or xx  
}

**Description:** If the condition cc is true, the offset is sign extended to dword and is added to PC. Interrupts are disabled until after the next instruction. The Bcc instruction is a delayed branch instruction, with one delay slot (see section 1.6.1 *Conditional Branch*). Valid instructions for the delay slot are all instructions except:

- Bcc
- BREAK/JBRC/JIR/JIRC/JMPU/JSR/JSRC/JUMP
- RET/RETB/RETI
- Instructions using addressing prefixes
- Immediate addressing other than Quick Immediate

The value of PC used for the address calculation is the address of the instruction *after* the branch instruction. Condition Codes:

(continued)

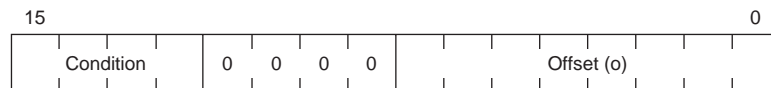
### 3 Instructions in Alphabetical Order

Code	Alt	Condition	Encoding	Boolean function
CC	HS	Carry Clear	0000	$\bar{C}$
CS	LO	Carry Set	0001	C
NE		Not Equal	0010	$\bar{Z}$
EQ		Equal	0011	Z
VC		Overflow Clear	0100	$\bar{V}$
VS		Overflow Set	0101	V
PL		Plus	0110	$\bar{N}$
MI		Minus	0111	N
LS		Low or Same	1000	C + Z
HI		High	1001	$\bar{C} * \bar{Z}$
GE		Greater or Equal	1010	$N * V + \bar{N} * \bar{V}$
LT		Less Than	1011	$N * \bar{V} + \bar{N} * V$
GT		Greater Than	1100	$N * V * \bar{Z} + \bar{N} * \bar{V} * Z$
LE		Less or Equal	1101	$Z + N * \bar{V} + \bar{N} * V$
A		Always True	1110	1
WF		Write Failed	1111	P

Table 3-1 Condition Codes

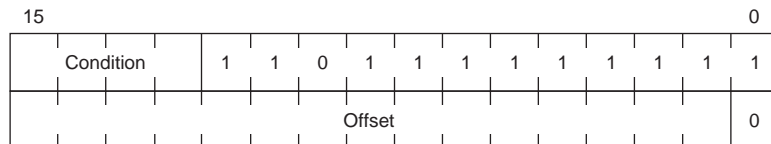
**flags affected:** F P U M B I X N Z V C  
 - - - - - 0 - - - -

**Instruction format:**  
 (8-bit offset)



**Offset:** Bits 7 - 1 of the offset represent bits 7 - 1 in the actual address increment/decrement. Bit 0 in the offset field is used as a sign bit in the computed offset. Bit 0 of the instruction field is bit 8 of the computed offset, and bit 0 of the computed offset is always 0.

**Instruction format:**  
 (16-bit offset)



**Offset:** Bits 15 - 1 make up the actual address increment/decrement. Bit 0 must always be 0 because of the word alignment of instructions.

# BOUND

2-operand

Adjust index to bound

# BOUND

2-operand

**Assembler syntax:** BOUND.m s,Rd

**Size:** Source is byte, word or dword. Operation is dword

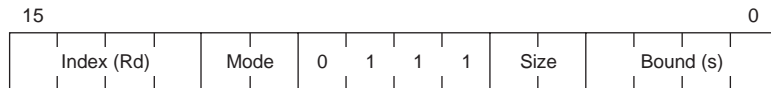
**Operation:**

```
if ((unsigned)Rd > (unsigned m)s)
{
    Rd = (unsigned m)s;
}
```

**Description:** This is a bounding instruction for adjusting branch indexes in switch statements. If the unsigned contents of the dword index (destination) register is greater than the unsigned bound (source) data, the bound data (zero extended to dword) is loaded to the index register. Otherwise, the index register is unaffected.

**flags affected:** F P U M B I X N Z V C  
 - - - - - 0 \* \* 0 0

**Instruction format:**  
 (register, indirect, or auto-increment addressing modes)



<b>Mode:</b>	01	Register addressing mode
	10	Indirect addressing mode
	11	Autoincrement addressing mode
<b>Size:</b>	00	Byte
	01	Word
	10	Dword

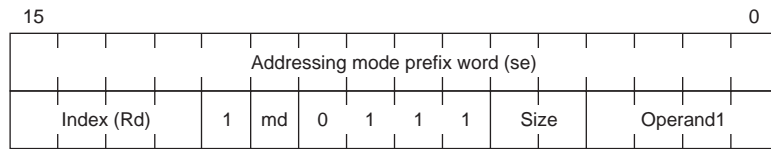
**Note 4:** PC is not allowed to be the Index (Rd) operand.

*(continued)*

### 3 Instructions in Alphabetical Order

---

**Instruction format:**  
(complex addressing modes)



<b>Mode (md):</b>	0	Indexed, offset, double indirect, or absolute addressing modes. The Operand1 field must be the same as index field (Rd).
	1	Indexed with assign, or offset with assign addressing modes. The Operand1 field selects the register in which to store the address of the source operand.
<b>Size:</b>	00	Byte
	01	Word
	10	Dword

**Note 5:** PC is not allowed to be the Index (Rd) operand.

# BOUND

3-operand

Adjust index to bound

# BOUND

3-operand

**Assembler syntax:** BOUND.m se,Rn,Rd**Size:** Source is byte, word or dword. Operation is dword

**Operation:**

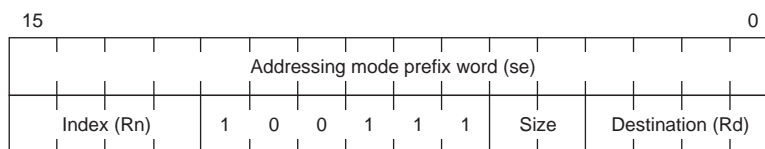
```

if ((unsigned) Rn > (unsigned m)se)
{
    Rd = (unsigned m)se;
}
else
{
    Rd = Rn;
}

```

**Description:** This is a bounding instruction for adjusting branch indexes in switch statements. If the unsigned contents of the dword index (Rn) register is greater than the unsigned bound (source) data, the bound data (zero extended to dword) is loaded to the destination register. Otherwise, the contents of the index register are loaded to the destination register.

**flags affected:** F P U M B I X N Z V C  
 - - - - - 0 \* \* 0 0

**Instruction format:**

<b>Size:</b>	00	Byte
	01	Word
	10	Dword

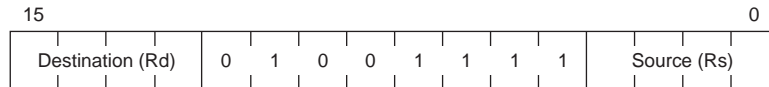
**Note 6:** PC is not allowed to be the Destination (Rd) operand.



# BTST

Bit test

# BTST

**Assembler syntax:** BTST Rs,Rd**Size:** Dword**Operation:** N = Bit number (Rs & 31) of Rd;  
Z = ((Bit numbers 0 to (Rs & 31) of Rd) == 0);**Description:** The N flag is set according to the selected bit in the destination register. The Z flag is set if the selected bit and all bits to the right of it are zero. The bit number is selected by the 5 least significant bits of the source register. The destination register is not affected.**flags affected:**  
F P U M B I X N Z V C  
- - - - - 0 \* \* 0 0**Instruction format:**

# BTSTQ

Bit test quick

# BTSTQ

**Assembler syntax:** BTSTQ c, Rd

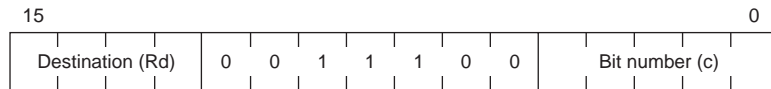
**Size:** Dword

**Operation:** N = Bit number c of Rd;  
 Z = ((Bit numbers 0 to c of Rd) == 0);

**Description:** The N flag is set according to the selected bit in the destination register. The Z flag is set if the selected bit and all bits to the right of it are zero. The bit number is selected by the 5-bit immediate value. The destination register is not affected.

**flags affected:** F P U M B I X N Z V C  
 - - - - - 0 \* \* 0 0

**Instruction format:**



# CLEAR

Clear

# CLEAR

**Assembler syntax:** CLEAR.m d**Size:** Byte, word, or dword**Operation:** (m)d = 0;**Description:** The destination is cleared to all zeroes. The size of the operation is m. Interrupts are disabled until the next instruction has been executed.**flags affected:**  
F P U M B I X N Z V C  
- - - - - 0 - - - -**Instruction format:**  
(register, indirect, or auto-increment addressing modes)

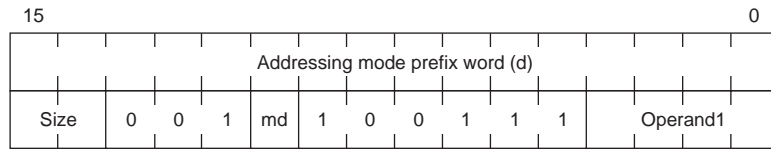
<b>Mode:</b>	01	Register addressing mode
	10	Indirect addressing mode
	11	Autoincrement addressing mode
<b>Size:</b>	00	Byte
	01	Word
	10	Dword

**Note 7:** If PC is used as the destination operand, the resulting jump will have a delayed effect with one delay slot.*(continued)*

### 3 Instructions in Alphabetical Order

---

**Instruction format:**  
(complex addressing modes)



<b>Mode (md):</b>	0	Indexed, offset, double indirect, or absolute addressing modes. The contents of the Operand1 field are ignored.
	1	Indexed with assign, or offset with assign addressing modes. The Operand1 field selects the register in which to store the address of the source operand.
<b>Size:</b>	00	Byte
	01	Word
	10	Dword

# CLEARF

Clear flags

# CLEARF

**Assembler syntax:** CLEARF <list of flags>**Size:** -**Operation:** Selected flags = 0;  
X = 0;  
F = 0;  
P = 0;**Description:** The specified flags are cleared to 0. The F, P, and X flags are always cleared even if they are not in the list supplied with CLEARF. The M and U flags are not affected. Interrupts are disabled until the next instruction has been executed.

When the list of flags contains more than one flag, the flags may be written in any order. The CLEARF instruction accepts an empty list of flags.

Examples:

```

CLEARF CVX      ; Clear F, P, C, V and X flags.
CLEARF          ; Clear F, P, and X flags.
CLEARF BI       ; Clear F, P, B, I and X flags.
CLEARF FP       ; Clear F, P and X flags.

```

**flags affected:** F P U M B I X N Z V C  
0 0 - - \* \* 0 \* \* \* \***Instruction format:**

15											0			
M	B	I	X	0	1	0	1	1	1	1	N	Z	V	C

# CMP

Compare

# CMP

**Assembler syntax:** `CMP.m s, Rd`

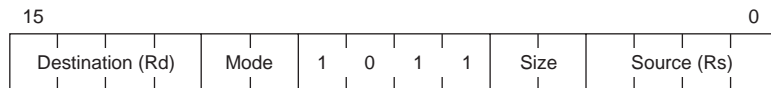
**Size:** Byte, word, or dword

**Operation:**  $(m)Rd - (m)s;$

**Description:** The source data is subtracted from the destination register, and the flags are set accordingly. The size of the operation is m. The destination register is not updated.

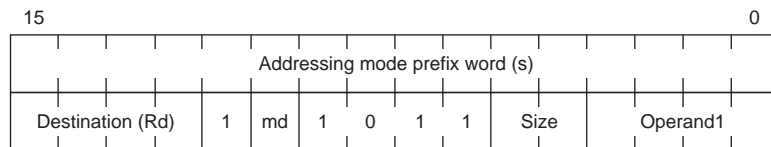
**flags affected:** F P U M B I X N Z V C  
- - - - - 0 \* \* \* \*

**Instruction format:**  
(register, indirect, or auto-increment addressing modes)



<b>Mode:</b>	01	Register addressing mode
	10	Indirect addressing mode
	11	Autoincrement addressing mode
<b>Size:</b>	00	Byte
	01	Word
	10	Dword

**Instruction format:**  
(complex addressing modes)



<b>Mode (md):</b>	0	Indexed, offset, double indirect, or absolute addressing modes. The contents of the Operand1 field are ignored.
	1	Indexed with assign, or offset with assign addressing modes. The Operand1 field selects the register in which to store the address of the source operand in.
<b>Size:</b>	00	Byte
	01	Word
	10	Dword

# CMPQ

Compare quick

# CMPQ

**Assembler syntax:** `CMPQ i, Rd`

**Size:** Dword

**Operation:**  $Rd - i;$

**Description:** A 6-bit immediate value, sign extended to dword, is subtracted from the destination register, and the flags are set accordingly. The destination register is not updated.

**flags affected:**

F	P	U	M	B	I	X	N	Z	V	C
-	-	-	-	-	-	0	*	*	*	*

**Instruction format:**

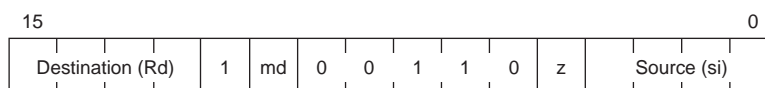




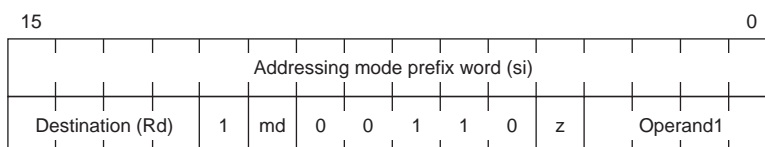
# CMPU

Compare with zero extend

# CMPU

**Assembler syntax:** `CMPU.z si,Rd`**Size:** Source size is byte or word. Operation size is dword**Operation:** `Rd - (unsigned z)si;`**Description:** The source data, zero extended to dword, is subtracted from the destination register, and the flags are set accordingly. The destination register is not updated.**flags affected:**  
F P U M B I X N Z V C  
- - - - - 0 \* \* \* \***Instruction format:**  
(indirect or autoincrement addressing modes)

<b>Mode (md):</b>	0	Indirect addressing mode
	1	Autoincrement addressing mode
<b>Size (z):</b>	0	Byte source operand
	1	Word source operand

**Instruction format:**  
(complex addressing modes)

<b>Mode (md):</b>	0	Indexed, offset, double indirect, or absolute addressing modes. The contents of the Operand1 field are ignored.
	1	Indexed with assign, or offset with assign addressing modes. The Operand1 field selects the register in which to store the address of the source operand.
<b>Size (z):</b>	0	Byte source operand
	1	Word source operand

**DI**

Disable interrupts

**DI**

**Assembler syntax:** DI

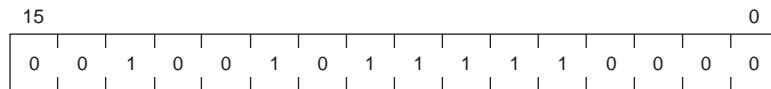
**Size:** -

**Operation:** I = 0;  
X = 0;  
F = 0;  
P = 0;

**Description:** Disable interrupts. This is a predefined assembler macro equivalent to CLEARF I.

**flags affected:** F P U M B I X N Z V C  
0 0 - - - 0 0 - - - -

**Instruction format:**



# DSTEP

Divide step

# DSTEP

**Assembler syntax:** DSTEP Rs,Rd

**Size:** Dword

**Operation:**

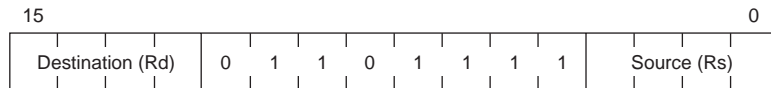
```
Rd <<= 1;
if ((unsigned)Rd >= (unsigned)Rs)
{
    Rd -= Rs;
}
```

**Description:** This is a divide-step operation, which performs one iteration of an iterative divide operation. The destination operand is shifted one step to the left. If the shifted destination operand is unsigned-greater-than or equal to the source operand, the source operand is subtracted from the shifted destination operand. The size of the operation is dword.

**flags affected:**

F	P	U	M	B	I	X	N	Z	V	C
-	-	-	-	-	-	0	*	*	0	0

**Instruction format:**



**Note 8:** PC is not allowed to be the destination operand (Rd).

**EI**

Enable interrupts

**EI**

**Assembler syntax:** EI

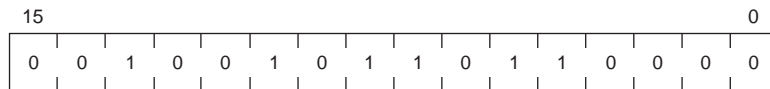
**Size:** -

**Operation:** I = 1;  
X = 0;

**Description:** Enable interrupts after the next instruction. This is a predefined assembler macro equivalent to SETF I.

**flags affected:** F P U M B I X N Z V C  
- - - - - 1 0 - - - -

**Instruction format:**



# JBRC

Jump to breakpoint routine,  
with context information

# JBRC

**Assembler syntax:** JBRC s

**Size:** Dword

**Operation:** BRP = PC + 4;  
PC = s;

**Description:** Jump to interrupt routine. The Breakpoint Return Pointer (BRP) is loaded with the contents of the program counter (PC). PC is then loaded with the contents of the source operand. Interrupts are disabled until the next instruction has been executed. The size of the operation is dword.

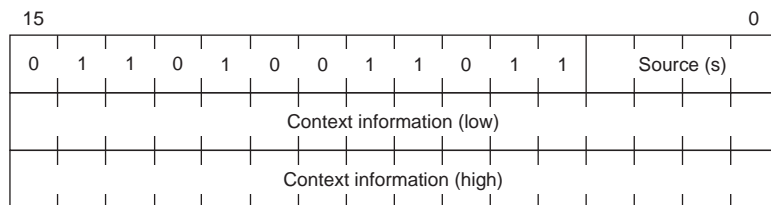
The JBRC instruction skips one dword at the PC and thus reserves one dword for context information, see section 1.6.6 *The JBRC, JIRC and JSRC Subroutine Instructions*. The context information is not used by the instruction.

The jump takes place immediately after the JBRC instruction.

The value of PC loaded to BRP is the address of the instruction after the JBRC instruction.

**flags affected:** F P U M B I X N Z V C  
- - - - - 0 - - - -

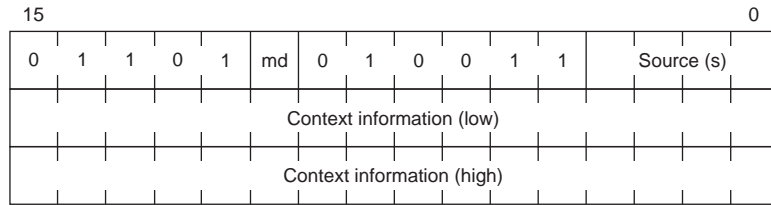
**Instruction format:**  
(register addressing mode)



(continued)

### 3 Instructions in Alphabetical Order

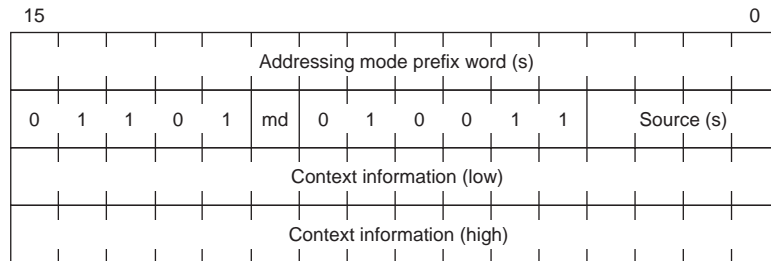
**Instruction format:**  
(indirect or autoincrement addressing modes)



<b>Mode (md):</b>	0	Indirect addressing mode
	1	Autoincrement addressing mode

**Note 9:** In immediate addressing mode, the immediate address is placed before the context information.

**Instruction format:**  
(complex addressing modes)



<b>Mode (md):</b>	0	Indexed, offset, double indirect, or absolute addressing modes. The contents of the Operand1 field are ignored.
	1	Indexed with assign, offset with assign addressing modes. The Operand1 field selects the register in which to store the address of the source operand.

# JIR

Jump to interrupt routine

# JIR

**Assembler syntax:** JIR s

**Size:** Dword

**Operation:** IRP = PC;  
PC = s;

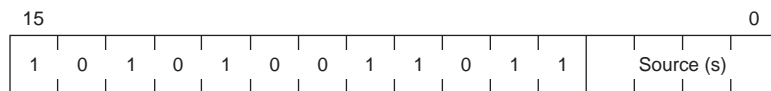
**Description:** Jump to interrupt routine. The interrupt return pointer (IRP) is loaded with the contents of the program counter (PC). PC is then loaded with the contents of the source operand. Interrupts are disabled until the next instruction has been executed. The size of the operation is dword.

The jump takes place immediately after the JIR instruction.

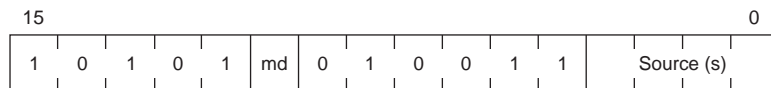
The value of PC loaded to IRP is the address of the instruction after the JIR instruction.

**flags affected:** F P U M B I X N Z V C  
- - - - - 0 - - - -

**Instruction format:**  
(register addressing mode)



**Instruction format:**  
(indirect or autoincrement addressing modes)



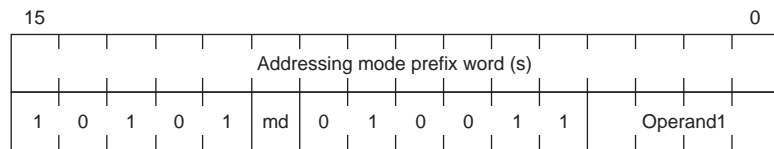
<b>Mode (md):</b>	0	Indirect addressing mode
	1	Autoincrement addressing mode

(continued)

### 3 Instructions in Alphabetical Order

---

**Instruction format:**  
(complex addressing modes)



<b>Mode (md):</b>	0	Indexed, offset, double indirect, or absolute addressing modes. The contents of the Operand1 field are ignored.
	1	Indexed with assign, offset with assign addressing modes. The Operand1 field selects the register in which to store the address of the source operand.

# JIRC

Jump to interrupt routine, with context information

# JIRC

**Assembler syntax:** JIRC s

**Size:** Dword

**Operation:** IRP = PC;  
PC = s;

**Description:** Jump to interrupt routine. The interrupt return pointer (IRP) is loaded with the contents of the program counter (PC). PC is then loaded with the contents of the source operand. Interrupts are disabled until the next instruction has been executed. The size of the operation is dword.

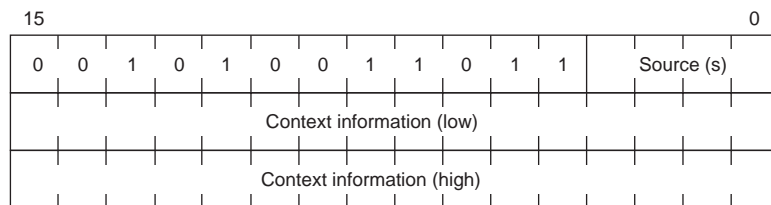
The JIRC instruction skips one dword at the PC and thus reserves one dword for context information, see section 1.6.6 *The JBRC, JIRC and JSRC Subroutine Instructions*. The context information is not used by the instruction.

The jump takes place immediately after the JIRC instruction.

The value of PC loaded to IRP is the address of the instruction after the JIRC instruction.

**flags affected:** F P U M B I X N Z V C  
- - - - - 0 - - - -

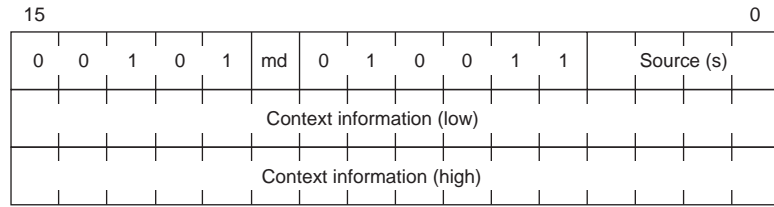
**Instruction format:**  
(register addressing mode)



(continued)

### 3 Instructions in Alphabetical Order

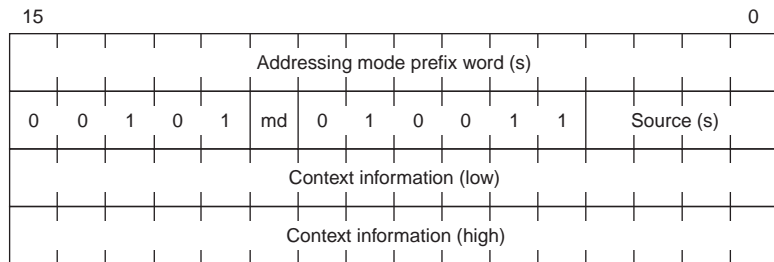
**Instruction format:**  
(indirect or autoincrement addressing modes)



<b>Mode (md):</b>	0	Indirect addressing mode
	1	Autoincrement addressing mode

**Note 10:** In immediate addressing mode, the immediate address is placed before the context information.

**Instruction format:**  
(complex addressing modes)



<b>Mode (md):</b>	0	Indexed, offset, double indirect, or absolute addressing modes. The contents of the Operand1 field are ignored.
	1	Indexed with assign, or offset with assign addressing modes. The Operand1 field selects the register in which to store the address of the source operand.



# JSR

Jump to subroutine

# JSR

**Assembler syntax:** JSR s

**Size:** Dword

**Operation:** SRP = PC;  
PC = s;

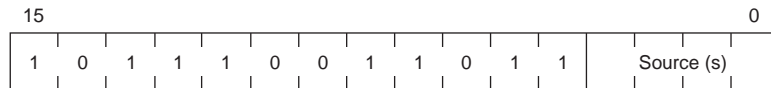
**Description:** Jump to subroutine. The subroutine return pointer (SRP) is loaded with the contents of the program counter (PC). PC is then loaded with the contents of the source operand. Interrupts are disabled until the next instruction has been executed. The size of the operation is dword.

The jump takes place immediately after the JSR instruction.

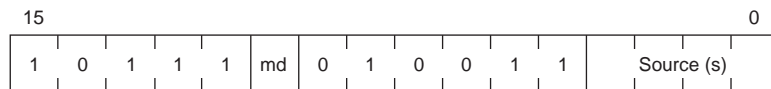
The value of PC loaded to SRP is the address of the instruction after the JSR instruction.

**flags affected:** F P U M B I X N Z V C  
- - - - - 0 - - - -

**Instruction format:**  
(register addressing mode)



**Instruction format:**  
(indirect or autoincrement addressing modes)



<b>Mode (md):</b>	0	Indirect addressing mode
	1	Autoincrement addressing mode

*(continued)*



# JSRC

Jump to subroutine, with context information

# JSRC

**Assembler syntax:** JSRC *s*

**Size:** Dword

**Operation:** SRP = PC;  
PC = *s*;

**Description:** Jump to subroutine. The subroutine return pointer (SRP) is loaded with the contents of the program counter (PC). PC is then loaded with the contents of the source operand. Interrupts are disabled until the next instruction has been executed. The size of the operation is dword.

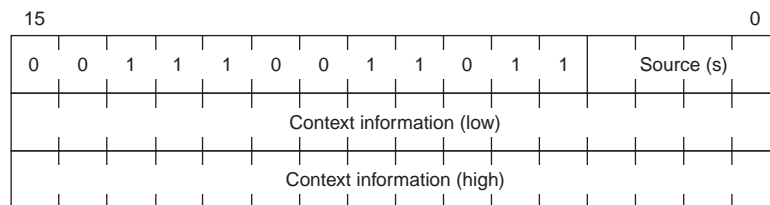
The JSRC instruction skips one dword at the PC and thus reserves one dword for context information, see section 1.6.6 *The JBRC, JIRC and JSRC Subroutine Instructions*. The context information is not used by the instruction.

The jump takes place immediately after the JSRC instruction.

The value of PC loaded to SRP is the address of the instruction after the JSRC instruction.

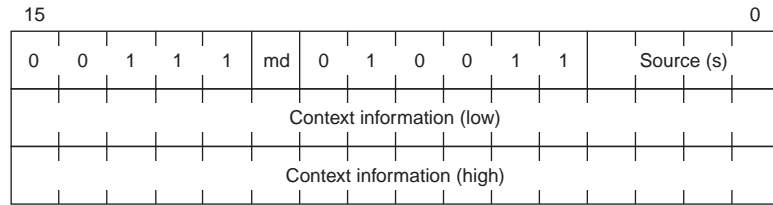
**flags affected:** F P U M B I X N Z V C  
- - - - - 0 - - - -

**Instruction format:**  
(register addressing mode)



(continued)

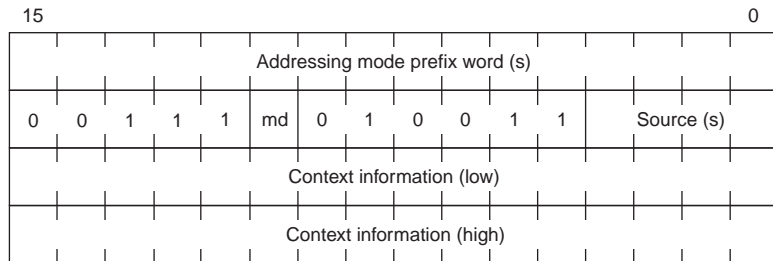
**Instruction format:**  
(indirect or autoincrement addressing modes)



<b>Mode (md):</b>	0	Indirect addressing mode
	1	Autoincrement addressing mode

**Note 11:** In immediate addressing mode, the immediate address is placed before the context information.

**Instruction format:**  
(complex addressing modes)



<b>Mode (md):</b>	0	Indexed, offset, double indirect, or absolute addressing modes. The contents of the Operand1 field are ignored.
	1	Indexed with assign, or offset with assign addressing modes. The Operand1 field selects the register in which to store the address of the source operand.

# JUMP

Jump

# JUMP

**Assembler syntax:** `JUMP s`

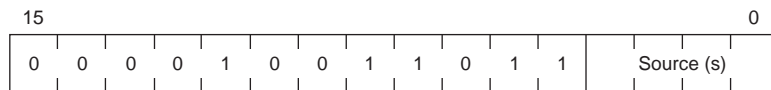
**Size:** Dword

**Operation:** `PC = s;`

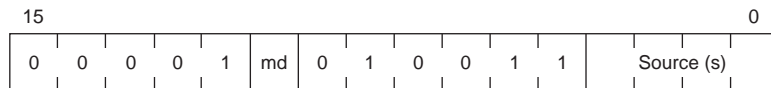
**Description:** PC is loaded with the contents of the source operand. The size of the operation is dword. The jump takes place immediately after the JUMP instruction. Interrupts are disabled until the next instruction has been executed.

**flags affected:** F P U M B I X N Z V C  
 - - - - - 0 - - - -

**Instruction format:**  
 (register addressing mode)

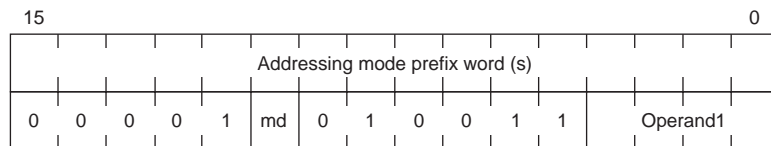


**Instruction format:**  
 (indirect or autoincrement addressing modes)



<b>Mode (md):</b>	0	Indirect addressing mode
	1	Autoincrement addressing mode

**Instruction format:**  
 (complex addressing modes)



<b>Mode (md):</b>	0	Indexed, offset, double indirect, or absolute addressing modes. The contents of the Operand1 field are ignored.
	1	Indexed with assign, or offset with assign addressing modes. The Operand1 field selects the register in which to store the address of the source operand.

# LSL

Logical shift left

# LSL

**Assembler syntax:** LSL.m Rs,Rd

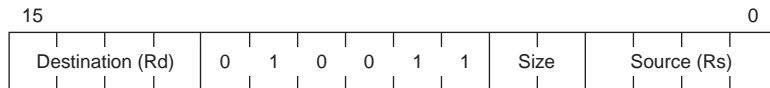
**Size:** Byte, word, or dword

**Operation:** (m)Rd <<= (Rs & 63);

**Description:** The destination register is left shifted the number of steps specified by the 6 least significant bits of the source register. The size of the operation is m. The rest of the destination register is not affected.

**flags affected:**  
 F P U M B I X N Z V C  
 - - - - - 0 \* \* 0 0

**Instruction format:**



	00	Byte
<b>Size:</b>	01	Word
	10	Dword

**Note 12:** PC is not allowed to be the destination operand (Rd).

**Note 13:** A shift of 32 bits or more will give a zero result.

# LSLQ

Logical shift left quick

# LSLQ

**Assembler syntax:** LSLQ c, Rd

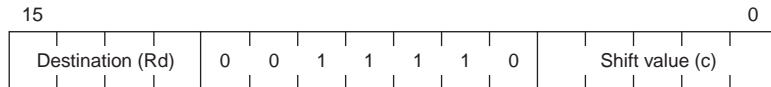
**Size:** Dword

**Operation:** Rd <<= c;

**Description:** The destination register is left shifted the number of steps specified by the 5-bit immediate value. The size of the operation is dword.

**flags affected:** F P U M B I X N Z V C  
- - - - - 0 \* \* 0 0

**Instruction format:**



**Note 14:** PC is not allowed to be the destination operand (Rd).

# LSR

Logical shift right

# LSR

**Assembler syntax:** LSR.m Rs,Rd

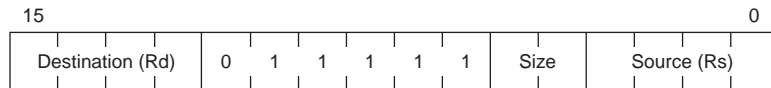
**Size:** Byte, word, or dword

**Operation:** (unsigned m)Rd >>= (Rs & 63);

**Description:** The destination register is right shifted the number of steps specified by the 6 least significant bits of the source register. The shift is performed with zero extend. The size of the operation is m. The rest of the destination register is not affected.

**flags affected:** F P U M B I X N Z V C  
 - - - - - 0 \* \* 0 0

**Instruction format:**



<b>Size:</b>	00	Byte
	01	Word
	10	Dword

**Note 15:** PC is not allowed to be the destination operand (Rd).

**Note 16:** A shift with 32 bits or more will give a zero result.

# LSRQ

Logical shift right quick

# LSRQ

**Assembler syntax:** LSRQ c, Rd

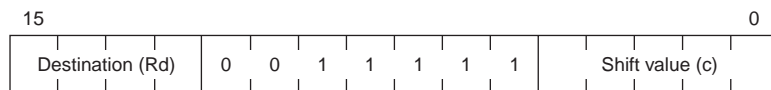
**Size:** Dword

**Operation:** (unsigned)Rd >>= c;

**Description:** The destination register is right shifted the number of steps specified by the 5-bit immediate value. The shift is performed with zero extend. The size of the operation is dword.

**flags affected:** F P U M B I X N Z V C  
- - - - - 0 \* \* 0 0

**Instruction format:**



**Note 17:** PC is not allowed as the destination operand (Rd).







# MOVE

to Pd

Move to special register

# MOVE

to Pd

**Assembler syntax:** MOVE s, Pd

**Size:** Byte, word or dword depending on the size of register Pd.

**Operation:** Pd = s;

**Description:** Move data from source to the destination special register. The size of the operation is the same as the size of the special register involved. Interrupts are disabled until the next instruction has been executed.

**flags affected:** F P U M B I X N Z V C  
(Pd != CCR, DCCR) - - - - - 0 - - - -

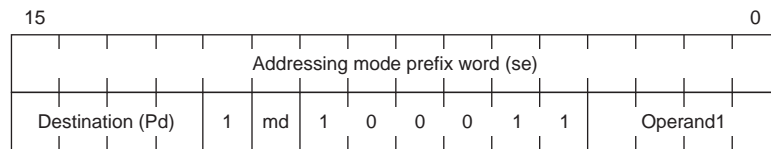
**flags affected:** F P U M B I X N Z V C  
(Pd == CCR, DCCR) \* \* \* - \* \* 0 \* \* \* \*

**Instruction format:**  
(register, indirect, or auto-increment addressing modes)



<b>Mode:</b>	01	Register addressing mode
	10	Indirect addressing mode
	11	Autoincrement addressing mode

**Instruction format:**  
(complex addressing modes)



<b>Mode (md):</b>	0	Indexed, offset, double indirect, or absolute addressing modes. The contents of the Operand1 field are ignored.
	1	Indexed with assign, or offset with assign addressing modes. The Operand1 field selects the register in which to store the address of the source operand.

# MOVE

from Ps

Move from special register

# MOVE

from Ps

**Assembler syntax:** MOVE Ps,d

**Size:** Byte, word or dword depending on the size of register Ps.

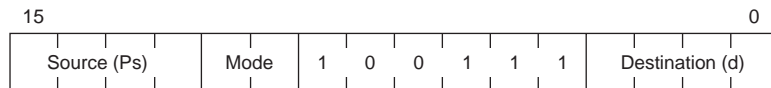
**Operation:** (size)d = Ps;

**Description:** Move data from the source special register to the destination. The size of the operation is the same as the size of the special register involved. The rest of the destination register is not affected. Interrupts are disabled until the next instruction has been executed.

**flags affected:** F P U M B I X N Z V C  
- - - - - 0 - - - -

The X flag is cleared *after* the instruction. If the X flag was set before a MOVE CCR,d instruction, the destination will have the bit corresponding to the X flag set.

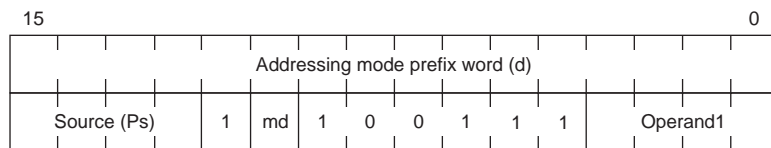
**Instruction format:**  
(register, indirect, or auto-increment addressing modes)



<b>Mode:</b>	01	Register addressing mode
	10	Indirect addressing mode
	11	Autoincrement addressing mode

**Note 18:** If PC is used as the destination operand, the resulting jump will have delayed effect, with one delay slot.

**Instruction format:**  
(complex addressing modes)



<b>Mode (md):</b>	0	Indexed, offset, double indirect, or absolute addressing modes. The contents of the Operand1 field are ignored.
	1	Indexed with assign, or offset with assign addressing modes. The Operand1 field selects the register in which to store the address of the source operand.

# MOVEM

from memory

Move to multiple registers  
from memory

# MOVEM

from memory

**Assembler syntax:** MOVEM si,Rd

**Size:** Dword

**Operation:**

```
n = rnumber;
while (n >= 0)
{
    Rn = si[rnumber - n];
    n--;
}
```

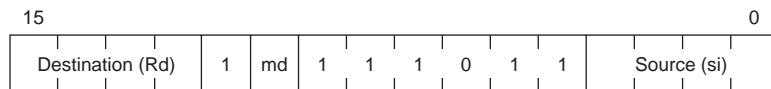
where rnumber is the register number of Rd, n is an integer and Rn the general register with register number n.

**Description:** The registers R0 to Rd are loaded from memory, starting at the memory location given by si. The size of each register transfer is dword. Rd is loaded from the lowest address (si), and R0 is loaded from the highest address: (si + 4 \* (<number of stored registers> - 1)).

**Note 19:** MOVEM from memory to register with autoincrement or assign is only valid if the source register number is greater than the destination register number.

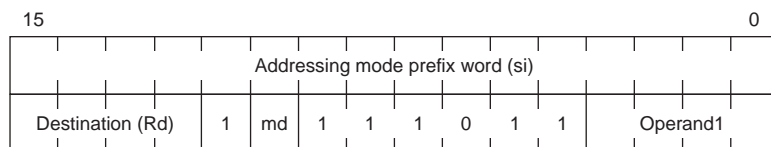
**flags affected:** F P U M B I X N Z V C  
- - - - - 0 - - - -

**Instruction format:**  
(indirect or autoincrement addressing modes)



<b>Mode (md):</b>	0	Indirect addressing mode
	1	Autoincrement addressing mode

**Instruction format:**  
(complex addressing modes)



<b>Mode (md):</b>	0	Indexed, offset, double indirect, or absolute addressing modes. The contents of the Operand1 field are ignored.
	1	Indexed with assign, or offset with assign addressing modes. The Operand1 field selects the register in which to store the address of the source operand.

# MOVEM

to memory

Move from multiple registers  
to memory

# MOVEM

to memory

**Assembler syntax:** MOVEM Rs,di**Size:** Dword

**Operation:**

```
n = rnumber;
while (n >= 0)
{
    di[rnumber - n] = Rn;
    n--;
}
```

where rnumber is the register number of Rd, n is an integer and Rn the general register with register number n.

**Description:** The contents of registers R0 to Rs are stored to memory, starting at the memory location given by di. The size of each register transfer is dword. Rs is stored at the lowest address: (di), and R0 is stored at the highest address: (di + 4 \* (<number of stored registers> - 1)).

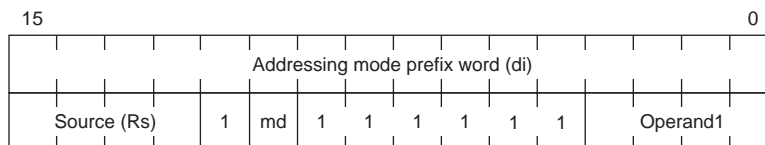
**flags affected:** F P U M B I X N Z V C  
- - - - - 0 - - - -

**Instruction format:**  
(indirect or autoincrement  
addressing modes)



<b>Mode (md):</b>	0	Indirect addressing mode
	1	Autoincrement addressing mode

**Instruction format:**  
(complex addressing modes)



<b>Mode (md):</b>	0	Indexed, offset, double indirect, or absolute addressing modes. The contents of the Operand1 field are ignored.
	1	Indexed with assign, or offset with assign addressing modes. The Operand1 field selects the register in which to store the address of the source operand.

# MOVEQ

Move quick

# MOVEQ

**Assembler syntax:** MOVEQ i, Rd

**Size:** Source data is 6-bit. Operation size is dword.

**Operation:** Rd = i;

**Description:** The destination register is loaded with a 6-bit immediate value, sign extended to dword.

**flags affected:** F P U M B I X N Z V C  
- - - - - 0 \* \* 0 0

**Instruction format:**







# MSTEP

Multiply step

# MSTEP

**Assembler syntax:** MSTEP Rs,Rd**Size:** Dword

**Operation:**

```
Rd <<= 1;
if (N)
{
  Rd += Rs;
}
```

**Description:** This is a multiply-step instruction, which performs one iteration of an iterative multiply operation. The destination operand is shifted one step to the left, and if the N flag is set before the instruction, the source operand is added to the shifted destination. The size of the operation is dword.

**flags affected:**

F	P	U	M	B	I	X	N	Z	V	C
-	-	-	-	-	-	0	*	*	0	0

**Instruction format:**

**Note 20:** PC is not allowed to be the destination operand (Rd).

# MULS

Signed multiply

# MULS

**Assembler syntax:** MULS.m Rs,Rd

**Size:** The operands are byte, word, or dword. The result is 64 bits.

**Operation:**  
 $MOF = ((m)Rs * (m)Rd) \gg 32;$   
 $Rd = (dword)((m)Rs * (m)Rd);$

**Description:** Both operands are sign extended from the size (m) to dword, and the extended operands are multiplied, generating a 64-bit result.

The lower 32 bits of the result are written to Rd, and the upper 32 bits are written to the multiply overflow register (MOF).

N and Z flags are set depending on the 64-bit result.

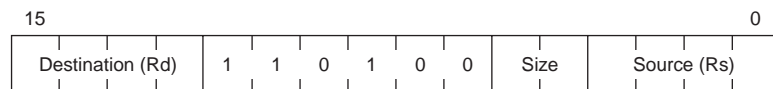
The V flag is set if the result is more than 32 bits:

$$V\text{-flag} = ((Rd \geq 0) \&\& (MOF \neq 0)) \mid \mid ((Rd < 0) \&\& (MOF \neq -1))$$

**flags affected:**

F	P	U	M	B	I	X	N	Z	V	C
-	-	-	-	-	-	0	*	*	*	0

**Instruction format:**



<b>Size:</b>	00	Byte
	01	Word
	10	Dword

**Note 21:** PC is not allowed to be the destination operand (Rd).



# NEG

Negate

# NEG

**Assembler syntax:** NEG.m Rs,Rd

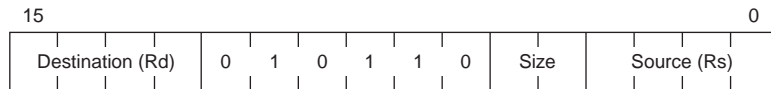
**Size:** Byte, word, or dword

**Operation:** (m)Rd = -(m)Rs;

**Description:** The contents of the source register is negated (2's complement), and stored in the destination register. The size of the operation is m.

**flags affected:** F P U M B I X N Z V C  
 - - - - - 0 \* \* \* \*

**Instruction format:**



<b>Size:</b>	00	Byte
	01	Word
	10	Dword

**Note 23:** PC is not allowed to be the destination operand (Rd).

# NOP

No operation

# NOP

**Assembler syntax:** NOP

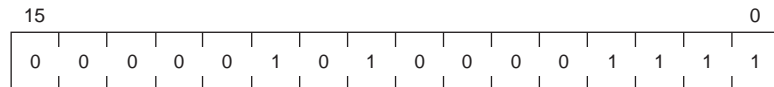
**Size:** -

**Operation:** ;

**Description:** No operation.

**flags affected:** F P U M B I X N Z V C  
- - - - - 0 - - - -

**Instruction format:**



# NOT

Logical complement

# NOT

**Assembler syntax:** NOT Rd

**Size:** Dword

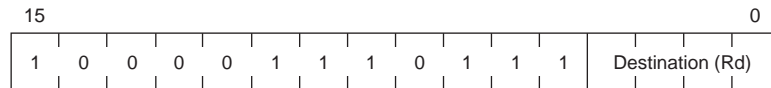
**Operation:** Rd = ~Rd;

**Description:** The contents of the source register is bitwise inverted (1's complement). The size of the operation is dword.

**flags affected:**

F	P	U	M	B	I	X	N	Z	V	C
-	-	-	-	-	-	0	*	*	0	0

**Instruction format:**



**Note 24:** PC is not allowed to be the destination operand (Rd).





# ORQ

Logical OR quick

# ORQ

**Assembler syntax:** ORQ *i*,*Rd*

**Size:** Source data is 6-bit. Operation size is dword.

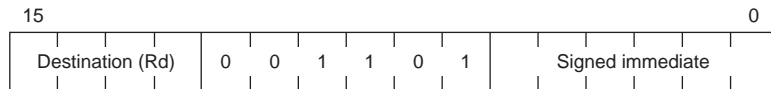
**Operation:**  $Rd \mid= i;$

**Description:** A logical OR is performed between a 6-bit immediate value, sign extended to dword, and the destination register.

**flags affected:**

F	P	U	M	B	I	X	N	Z	V	C
-	-	-	-	-	-	0	*	*	0	0

**Instruction format:**









# PUSH

from Ps

Push special register onto stack

# PUSH

from Ps

**Assembler syntax:** PUSH Ps

**Size:** Byte, word, or dword depending on the size of register Ps

**Operation:**  $*(--(size * )SP) = Ps;$

**Description:** The entire source special register is pushed on the stack, assuming SP as stack pointer. Interrupts are disabled until the next instruction has been executed. This is a predefined assembler macro equivalent to `MOVE Ps,[SP=SP-sizeof(Ps)]`, where `sizeof(Ps)` is the size of the source special register in Bytes.

**flags affected:** F P U M B I X N Z V C  
- - - - - 0 - - - -

**Instruction format:**



Size is set according to the size of the pushed register.

<b>Size:</b>	11	Byte (Ps = VR)
	10	Word (Ps = CCR)
	00	Dword (Ps = BAR, BRP, DCCR, IBR, IRP, MOF, SRP, or USP)

# RBF

Return from Bus Fault

# RBF

**Assembler syntax:** RBF si

**Size:** -

**Operation:** -

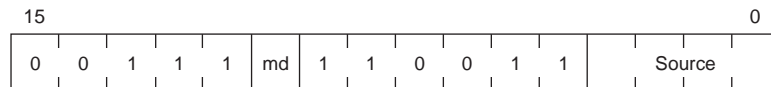
**Description:** The RBF instruction uses a 16 byte CPU status record to restore the internal CPU state, and to resume the execution that was interrupted by a previous bus fault. If the U flag is set before the instruction, the CPU will go to user mode, otherwise it will stay in its current mode.

RBF restarts execution from the latest instruction boundary before the interrupted instruction. (In this case, addressing prefixes are considered as separate instructions.) The cycles between the latest instruction boundary and the point where the instruction was interrupted will be run internally in the CPU, without causing bus request. Any data that the CPU reads in these cycles is taken from the restored CPU status record. MOVEM instructions are handled specially. They will be restarted with the register number that was in transfer when the bus fault occurred.

The X and U flags will be set or cleared depending on bits in the CPU status record.

**flags affected:** F P U M B I X N Z V C  
 - - \* - - - \* - - - -

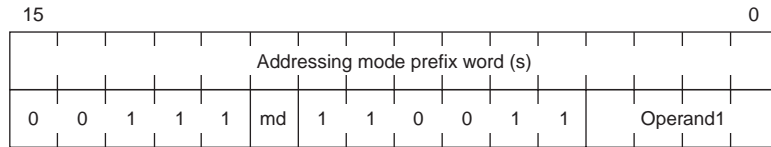
**Instruction format:**  
 (indirect, or auto-increment addressing modes)



<b>Mode (md):</b>	0	Indirect addressing mode
	1	Autoincrement addressing mode

*(continued)*

**Instruction format:**  
(complex addressing modes)



<b>Mode (md):</b>	0	Indexed, offset, double indirect, and absolute addressing modes. Operand1 field should be 0000 (binary).
	1	Indexed with assign, and offset with assign addressing modes. Operand1 field selects the register in which to store the source address.

# RET

Return from subroutine

# RET

**Assembler syntax:** RET

**Size:** Dword

**Operation:** PC = SRP;

**Description:** Return from subroutine (see note). The contents of the subroutine return pointer (SRP) is loaded to PC. The size of the operation is dword. Interrupts are disabled until the next instruction has been executed.

The RET instruction is a delayed jump instruction, with one delay slot. Valid instructions for the delay slot are all instructions except:

- Bcc
- BREAK/JBRC/JIR/JIRC//JSR/JSRC/JUMP
- RET/RETB/RETI
- Instructions using addressing prefixes
- Immediate addressing other than Quick Immediate

The RET instruction is a predefined assembler macro equivalent to MOVE SRP,PC.

**Note 25:** The RET instruction is only used for returns from terminal subroutines (subroutines that do not call other subroutines). For non-terminal subroutines, where the return address is saved on the stack, it is more efficient to use the JUMP [SP+] instruction.

**flags affected:** F P U M B I X N Z V C  
- - - - - 0 - - - -

**Instruction format:**



# RETB

Return from breakpoint

# RETB

**Assembler syntax:** RETB**Size:** Dword**Operation:** PC = BRP;**Description:** Return from breakpoint routine (see note). The contents of the breakpoint return pointer (BRP) is loaded to PC. The size of the operation is dword. Interrupts are disabled until the next instruction has been executed.

The RETB instruction is a delayed jump instruction, with one delay slot. Normally the delay slot after RETB should be used to pop the flags, and the jump is performed after the instruction that follows RETB.

RETB performs a transition to user mode if the U flag is set. If the U flag is not set, the CPU stays in its current mode. The transition to user mode is delayed until after the delay slot so that the delay slot is run in the current mode. The transition to user mode will depend on the value of the U flag after the delay slot instruction.

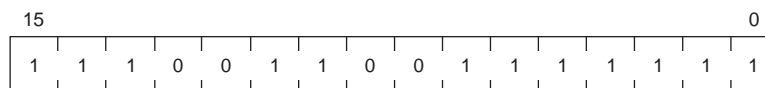
A special case occurs if you get a bus fault in the delay slot of the RETB instruction. The bus fault sequence will, in this case, set the U flag corresponding to the operating mode that was valid in the delay slot so that the interrupted instruction can be restarted in the correct mode. A separate bit in the CPU status record will be set to tell the RBF instruction to set operating mode according to the U flag once more after the restarted instruction.

If RETB is placed in a delay slot of a branch, RET, RETI or RETB that is taken, the RETB in the delay slot will not be performed. Consequently, the operating mode of the CPU will not be altered in that case.

The RETB instruction is a predefined assembler macro equivalent to MOVE BRP,PC.

**Note 26:** The RETB instruction is only used for returns from interrupt routines that are not nested. For nested interrupt routines, where the return address is saved on the stack, it is more efficient to use the JMPU [SP+] instruction.

**flags affected:** F P U M B I X N Z V C  
- - - - - 0 - - - -

**Instruction format:**





# Scc

Set according to condition

# Scc

**Assembler syntax:** Scc Rd

**Size:** Dword

**Operation:**

```

if (cc)
{
    Rd = 1;
}
else
{
    Rd = 0;
}
    
```

**Description:** The destination register is loaded with 1 if the condition cc is true, and with 0 otherwise. The size of the operation is dword. Interrupts are disabled until the next instruction has been executed.

**Condition Codes:**

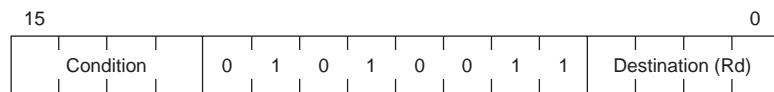
Code	Alt	Condition	Encoding	Boolean function
CC	HS	Carry Clear	0000	$\bar{C}$
CS	LO	Carry Set	0001	C
NE		Not Equal	0010	$\bar{Z}$
EQ		Equal	0011	Z
VC		Overflow Clear	0100	$\bar{V}$
VS		Overflow Set	0101	V
PL		Plus	0110	$\bar{N}$
MI		Minus	0111	N
LS		Low or Same	1000	C + Z
HI		High	1001	$\bar{C} * \bar{Z}$
GE		Greater or Equal	1010	$N * V + \bar{N} * \bar{V}$
LT		Less Than	1011	$N * \bar{V} + \bar{N} * V$
GT		Greater Than	1100	$N * V * \bar{Z} + \bar{N} * \bar{V} * \bar{Z}$
LE		Less or Equal	1101	$Z + N * \bar{V} + \bar{N} * V$
A		Always True	1110	1
WF		Write Failed	1111	P

Table 3-2

**flags affected:** F P U M B I X N Z V C  
 - - - - - 0 - - - -

(continued)

**Instruction format:**



**Note 28:** PC is not allowed to be the destination operand (Rd).

# SETF

Set flags

# SETF

**Assembler syntax:** SETF <list of flags>

**Size:** -

**Operation:** X = 0;  
Selected flags = 1;

**Description:** The specified flags are set to 1. If the X flag is not in the list, it will be cleared. Interrupts are disabled until the next instruction has been executed.

When the list of flags contains more than one flag, the flags may be written in any order. The SETF instruction accepts an empty list of flags.

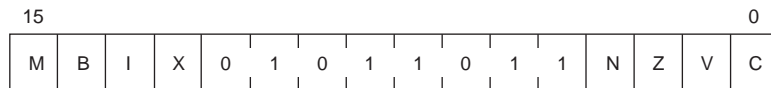
Examples:

```

SETF    CVX    ; Set C, V and X flags.
SETF                    ; Clear X flag.
SETF    MBI    ; ;Set M, B and I flags, and clear X flag.
    
```

**flags affected:** F P U M B I X N Z V C  
- - - \* \* \* \* \* \* \* \*

**Instruction format:**







# SUBQ

Subtract quick

# SUBQ

**Assembler syntax:** SUBQ j, Rd

**Size:** Source data is 6-bit. Operation size is dword

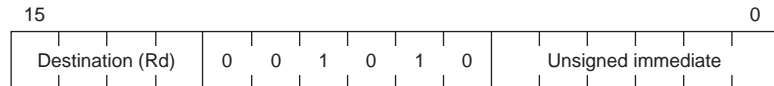
**Operation:** Rd -= j;

**Description:** A 6-bit immediate value, zero extended to dword, is subtracted from the destination register.

**flags affected:**

F	P	U	M	B	I	X	N	Z	V	C
-	-	-	-	-	-	0	*	*	*	*

**Instruction format:**











# SWAP

Swap bits

# SWAP

**Assembler syntax:** SWAP<option list> Rd

**Size:** Dword

**Operation:**

```
if (option N)
{
    Rd = ~Rd;
}
if (option W)
{
    Rd = (Rd << 16) | ((Rd >> 16) & 0xffff);
}
if (option B)
{
    Rd = ((Rd << 8) & 0xff00ff00) |
        ((Rd >> 8) & 0x00ff00ff);
}
if (option R)
{
    Rd = ((Rd << 7) & 0x80808080) |
        ((Rd << 5) & 0x40404040) |
        ((Rd << 3) & 0x20202020) |
        ((Rd << 1) & 0x10101010) |
        ((Rd >> 1) & 0x08080808) |
        ((Rd >> 3) & 0x04040404) |
        ((Rd >> 5) & 0x02020202) |
        ((Rd >> 7) & 0x01010101);
}
```

**Description:** The bits in the destination register are reorganized according to the specified option(s). The following options apply:

N Invert all bits in the operand.

W Swap the words of the operand.

B Swap the two bytes within each word of the operand.

R Reverse the bit order within each byte of the operand.

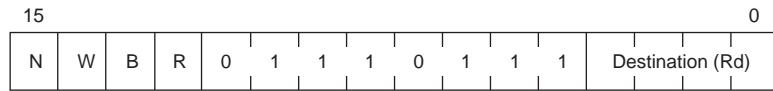
Any combination of the four options is allowed. If more than one option is specified, they must be given in the order NWBR. The size of the operation is dword.

The SWAPN instruction is a synonym for the NOT instruction.

*(continued)*

**flags affected:** F P U M B I X N Z V C  
- - - - - 0 \* \* 0 0

**Instruction format:**



**Note 29:** PC is not allowed to be the destination operand (Rd).

# TEST

Compare with zero

# TEST

**Assembler syntax:** TEST.m s

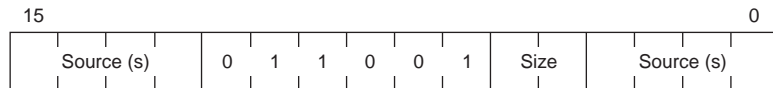
**Size:** Byte, word, or dword

**Operation:** (m)s - 0;

**Description:** Zero is subtracted from the source data, and the flags are set accordingly. For a register operand, this is a predefined assembler macro equivalent to MOVE.m Rs,Rs.

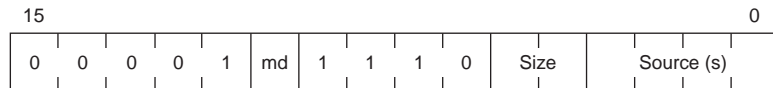
**flags affected:** F P U M B I X N Z V C  
 - - - - - 0 \* \* 0 0

**Instruction format:**  
 (register addressing mode)



<b>Size:</b>	00	Byte
	01	Word
	10	Dword

**Instruction format:**  
 (indirect or autoincrement addressing modes)



<b>Mode (md):</b>	0	Indirect addressing mode
	1	Autoincrement addressing mode

<b>Size:</b>	00	Byte
	01	Word
	10	Dword

*(Continued)*



# XOR

Exclusive logical OR

# XOR

**Assembler syntax:** XOR Rs, Rd

**Size:** Dword

**Operation:** Rd ^= RS;

**Description:** A logical exclusive OR is performed between the contents of the source register and the destination register. The size of the operation is dword.

**flags affected:** F P U M B I X N Z V C  
- - - - - 0 \* \* 0 0

**Instruction format:**

