FR400 Series
Instruction Set Manual

Version 2.2 Jan, 2004

FUJITSU

FUJITSU LIMITED

1) The contents of this document are subject to change without notice.
   Customers are advised to consult with FUJITSU sales representatives before ordering.

2) The information, such as descriptions of function and application circuit examples, in this document are presented solely for the purpose of reference to show examples of operations and uses of FUJITSU semiconductor device; FUJITSU does not warrant proper operation of the device with respect to use based on such information. When you develop equipment incorporating the device based on such information, you must assume any responsibility arising out of such use of the information. FUJITSU assumes no liability for any damages whatsoever arising out of the use of the information.

3) Any information in this document, including descriptions of function and schematic diagrams, shall not be construed as license of the use or exercise of any intellectual property right, such as patent right or copyright, or any other right of FUJITSU or any third party or does FUJITSU warrant non-infringement of any third-party's intellectual property right or other right by using such information. FUJITSU assumes no liability for any infringement of the intellectual property rights or other rights of third parties which would result from the use of information contained herein.

4) The products described in this document are designed, developed and manufactured as contemplated for general use, including without limitation, ordinary industrial use, general office use, personal use, and household use, but are not designed, developed and manufactured as contemplated (1) for use accompanying fatal risks or dangers that, unless extremely high safety is secured, could have a serious effect to the public, and could lead directly to death, personal injury, severe physical damage or other loss (i.e., nuclear reaction control in nuclear facility, aircraft flight control, air traffic control, mass transport control, medical life support system, missile launch control in weapon system), or (2) for use requiring extremely high reliability (i.e., submersible repeater and artificial satellite).
   Please note that FUJITSU will not be liable against you and/or any third party for any claims or damages arising in connection with above-mentioned uses of the products.

5) Any semiconductor devices have an inherent chance of failure. You must protect against injury, damage or loss from such failures by incorporating safety design measures into your facility and equipment such as redundancy, fire protection, and prevention of over-current levels and other abnormal operating conditions.

6) If any products described in this document represent goods or technologies subject to certain restrictions on export under the Foreign Exchange and Foreign Trade Law of Japan, the prior authorization by Japanese government will be required for export of those products from Japan.

Contents

Table Contents

1. Instruction Set Reference

This chapter describes instruction set architecture, which are categorized into related instructions.

1.1. Explanation of each term

Instruction table

This term is described about the ope-code of each instruction, op fields in instruction code, ope fields in instruction code and execution behavior.

Category

This term is described about instruction type of each instruction.

Instruction format

This term is described about instruction format of each instruction.

Assembler description

This term is described about suggested assembly language syntax.

Behavior description

This term is described about description of instruction execution.

Operation example

In the instruction which needs a complex operation, an example is shown in the table.

Altered register other than a destination register

This term is described an altered register which is not indicated by assembly description.
For ICC, it is described that the flag may be altered by the instruction. A symbol O means that the flag may be altered by the instruction. A symbol X means that the flag does not altered by the instruction.

Occurrence exception

This term is described exceptions that may be occurred by the instruction.

Detected exception

When the instruction detects exceptions, there are exceptions which do not initiate an exception processing and may store the information. This term is described about this kind of exception. When the instruction is non-excepting instruction, the detected exception is initiated by the COMMIT instruction. When the instruction is media instruction, the detected exception is initiated by executing MTRAP instruction.

1.2. Integer Instructions

1.2.1. Add / Subtract (ADDSS and SUBSS are available for MB93405/MB93451.)

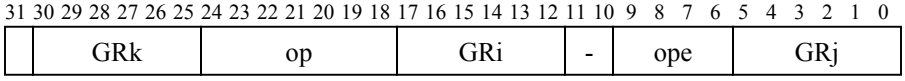
Ope-code	op	ope	Operation
ADD	0000000	0000	Add
ADDcc	0000000	0001	Add and ICC setting
ADDX	0000000	0010	Add with carry
ADDXcc	0000000	0011	Add with carry and ICC setting
SUB	0000000	0100	Subtract
SUBcc	0000000	0101	Subtract and ICC setting
SUBX	0000000	0110	Subtract with carry
SUBXcc	0000000	0111	Subtract with carry and ICC setting
ADDI	0010000	-	Add (Immediate)
ADDIcc	0010001	-	Add and ICC setting (Immediate)
ADDXI	0010010	-	Add with carry (Immediate)
ADDXIcc	0010011	-	Add with carry and ICC setting (Immediate)
SUBI	0010100	-	Subtract (Immediate)
SUBIcc	0010101	-	Subtract and ICC setting (Immediate)
SUBXI	0010110	-	Subtract with carry (Immediate)
SUBXIcc	0010111	-	Subtract with carry and ICC setting (Immediate)
ADDSS	1000110	000000	Add with Signed Saturation
SUBSS	1000110	000001	Substract with Signed Saturation

 # : ADDSS and SUBSS are available for MB93405/MB93451.

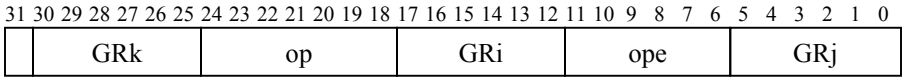
Category

Integer

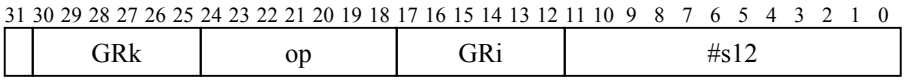
Instruction Format (INT, Logic, Shift Operation (R-R))



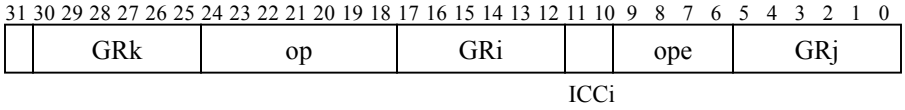
Instruction Format (ADDSS,SUBSS)



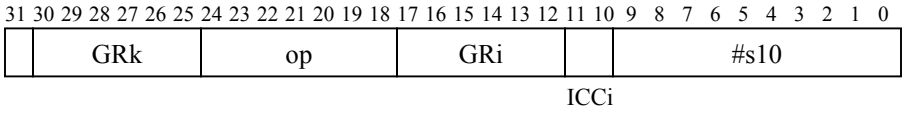
Instruction Format (INT, Logic, Shift Operation (R-simm))



Instruction Format (INT, Logic, Shift-cc Operation (R-R))



Instruction Format (INT, Logic, Shift-cc Operation (R-simm))



Assembler Syntax

ADD	GRi, GRj, GRk
ADDcc	GRi, GRj, GRk, ICCi
ADDX	GRi, GRj, GRk, ICCi
ADDXcc	GRi, GRj, GRk, ICCi
SUB	GRi, GRj, GRk
SUBcc	GRi, GRj, GRk, ICCi
SUBX	GRi, GRj, GRk, ICCi
SUBXcc	GRi, GRj, GRk, ICCi
ADDI	GRi, #s12, GRk
ADDIcc	GRi, #s10, GRk, ICCi
ADDXI	GRi, #s10, GRk, ICCi
ADDXIcc	GRi, #s10, GRk, ICCi
SUBI	GRi, #s12, GRk
SUBIcc	GRi, #s10, GRk, ICCi
SUBXI	GRi, #s10, GRk, ICCi
SUBXIcc	GRi, #s10, GRk, ICCi
ADDSS	GRi, GRj, GRk
SUBSS	GRi, GRj, GRk

Description

The integer addition and subtraction instructions add or subtract GRi and GRj (immediate instruction: sign_ext (#s12) or sign_ext (#s10)) and write the result in GRk.
The ADD instruction calculates “GRi+GRj”.
The ADDX instruction calculates “GRi+GRj+C”. C is the carry bit of ICCi.
The SUB instruction calculates “GRi - GRj”.
The SUBX instruction calculates “GRi - GRj – C”. C is the carry bit of ICCi.
The ADDcc, ADDXcc, SUBcc, and SUBXcc instructions and their immediate instructions change the integer condition code (ICC).
The ADDSS instruction adds a 32bit value of GRi to a 32bit value of GRj, and saturate,to pruduce a 32bit result. The result is placed into GRk. If the result is overflow as 32bit signed integer, maximum value of 32bit signed interger placed into GRk. (0x7FFFFFFF or 0x80000000)
The SUBSS instruction subrtacts a 32bit value of GRi to a 32bit value of GRj, and saturate,to pruduce a 32bit result. The result is placed into GRk. If the result is overflow as 32bit signed integer, maximum value of 32bit signed interger placed into GRk. (0x7FFFFFFF or 0x80000000)

Registers altered (except destination register)

ICCi … instructions with “cc” only

N	Z	V	C
O	O	O	O

Occurrence Exceptions

register_exception (unimplement_exception)

Detected Exceptions

none

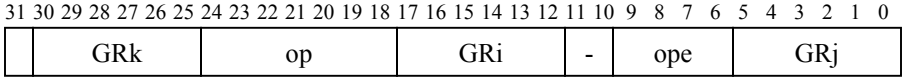
1.2.2. Multiply

Ope-code	op	ope	Operation
SMUL	0000000	1000	Signed Integer Multiply
SMULcc	0000000	1001	Signed Integer Multiply and ICC setting
UMUL	0000000	1010	Unsigned Integer Multiply
UMULcc	0000000	1011	Unsigned Integer Multiply and ICC setting
SMULI	0011000	-	Signed Integer Multiply (Immediate)
SMULIcc	0011001	-	Signed Integer Multiply and ICC setting (Immediate)
UMULI	0011010	-	Unsigned Integer Multiply (Immediate)
UMULIcc	0011011	-	Unsigned Integer Multiply and ICC setting (Immediate)

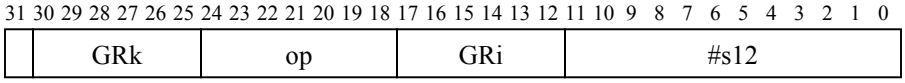
Category

Integer

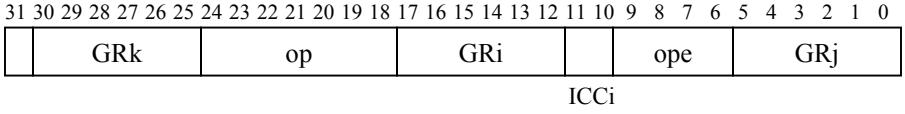
Instruction Format (INT, Logic, Shift Operation (R-R))



Instruction Format (INT, Logic, Shift Operation (R-simm))



Instruction Format (INT, Logic, Shift-cc Operation (R-R))



Instruction Format (INT, Logic, Shift-cc Operation (R-simm))

Assembler Syntax

SMUL	GRi, GRj, GRk
SMULcc	GRi, GRj, GRk, ICCi
UMUL	GRi, GRj, GRk
UMULcc	GRi, GRj, GRk, ICCi
SMULI	GRi, #s12, GRk
SMULIcc	GRi, #s10, GRk, ICCi
UMULI	GRi, #s12, GRk

UMULIcc GRi, #s10, GRk, ICCi

Description

The integer multiply instructions multiply GRi and GRj (immediate instruction: sign_ext (#s12) or sign_ext (#s10)) and write the high-order 32 bits of the result in GRk and the low-order 32 bits in GRk+1.
The SMUL and SMULcc instructions calculate the product of signed integer word operands and write a signed integer doubleword as the result.
The UMUL and UMULcc instructions calculate the product of unsigned integer word operands and write an unsigned integer doubleword as the result.
The SMULcc and UMULcc instructions and their immediate instructions change the integer condition code (ICC) specified by ICCi field.
register_exception (register_not_aligned) occur when the register number of GRk is an odd number.

Registers altered (except destination register)

ICCi … instructions with "cc" only

N	Z	V	C
O	O	X	X

Occurrence Exceptions

register_exception (unimplement_exception, register_not_aligned)

Detected Exceptions

none

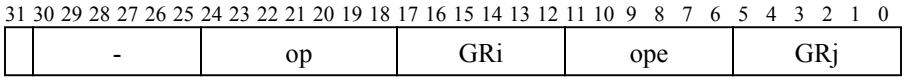
1.2.3. Multiply and Add / Subtract (These instructions are available for MB93405/MB93451.)

Ope-code	op	ope	Operation
SMASS	1000110	000110	Signed Multiply and Add with Signed Saturation (64bit + 32bit x 32bit -> 64 bit)
SMSSS	1000110	000111	Signed Multiply and Subtract with Signed Saturation (64bit – 32bit x32bit -> 64bit)

Category

Integer

Instruction Format (SMASS,SMSSS)



Assembler Syntax

SMASS GRi, GRj
SMSSS GRi, GRj

Description

SMASS multiplies the 32bit value in GRi and the 32bit value in GRj , then add the product with the 64bit value in IACC. All operands are treated as signed values, to produce a 64bit result. The result is placed into IACC. If the result is overflow as 64bit signed integer, maximum value of 64bit signed integer is placed into IACC. (0x7FFFFFFFFFFFFFFF or 0x8000000000000000)
SMSSS multiplies the 32bit value in GRi and the 32bit value in GRj , then subtract the product from the 64bit value in IACC. All operands are treated as signed values, to produce a 64bit result. The result is placed into IACC. If the result is overflow as 64bit signed integer, maximum value of 64bit signed integer is placed into IACC. (0x7FFFFFFFFFFFFFFF or 0x8000000000000000)

Registers altered (except destination register)

none

Occurrence Exceptions

register_exception (unimplement_exception)

Detected Exceptions

none

1.2.4. Multiply to IACC (This instruction is available for MB93405/MB93451.)

Ope-code	op	ope	Operation
SMU	1000110	000101	Signed Multiply (32bit x 32bit -> 64 bit)

Category

Integer

Instruction Format (SMASS,SMSSS)

31	30 29 28 27 26 25	24 23 22 21 20 19	18 17 16 15	14 13 12 11 10	9 8 7 6 5 4 3 2 1 0
	-	op	GRi	ope	GRj

Assembler Syntax

SMU GRi, GRj

Description

SMU multiplies the 32bit value in GRi and the 32bit value in GRj.All operands are treated as signed values, to produce a 64bit result. The result is placed into IACC.

Registers altered (except destination register)

none

Occurrence Exceptions

register_exception (unimplement_exception)

Detected Exceptions

none

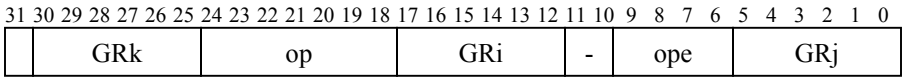
1.2.5. Divide

Ope-code	op	ope	Operation
SDIV	0000000	1110	Signed Integer Divide
UDIV	0000000	1111	Unsigned Integer Divide
SDIVI	0011110	-	Signed Integer Divide (Immediate)
UDIVI	0011111	-	Unsigned Integer Divide (Immediate)

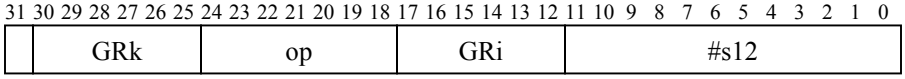
Category

Integer

Instruction Format (INT, Logic, Shift Operation (R-R))



Instruction Format (INT, Logic, Shift Operation (R-simm))



Assembler Syntax

SDIV GRi, GRj, GRk
UDIV GRi, GRj, GRk
SDIVI GRi, #s12, GRk
UDIVI GRi, #s12, GRk

Description

The integer division instructions divide GRi by GRj (immediate instruction: sign_ext (#s12)) and write the result in GRk. The surplus is not given as a result.
The SDIV instruction divides a signed integer word operand by a signed integer word operand and writes a signed integer word as the result.
The result of 0x80000000/ (-1) in a SDIV instruction is 0x7fffffff in division exception mask mode (ISR.EDEM=1), 0x80000000 in division exception detection mode.
Division_exception cannot be masked because Division exception mask mode is for only overflow. The UDIV instruction divides an unsigned integer word operand by an unsigned integer word operand and writes an unsigned integer word as the result.
The UDIVI instruction divides 32-bit signed integer which results from sign_ext(#s12) regarding as 32-bit unsigned integer.

Registers altered (except destination register)

none

Occurrence Exceptions

division_exception
register_exception (unimplement_exception)

Detected Exceptions

none

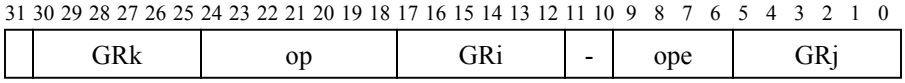
1.2.6. Logical Operations

Ope-code	op	ope	Operation
AND	0000001	0000	And
ANDcc	0000001	0001	And and ICC setting
OR	0000001	0010	Or
ORcc	0000001	0011	Or and ICC setting
XOR	0000001	0100	Xor
XORcc	0000001	0101	Xor and ICC setting
NOT	0000001	0110	Not
ANDI	0100000	-	And (Immediate)
ANDIcc	0100001	-	And and ICC setting (Immediate)
ORI	0100010	-	Or (Immediate)
ORIcc	0100011	-	Or and ICC setting (Immediate)
XORI	0100100	-	Xor (Immediate)
XORIcc	0100101	-	Xor and ICC setting (Immediate)

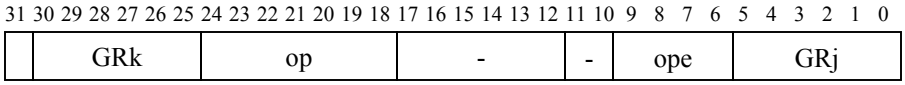
Category

Integer

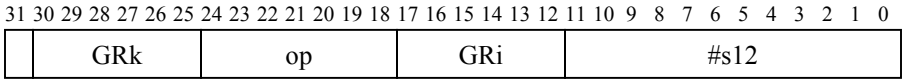
Instruction Format (INT, Logic, Shift Operation (R-R))



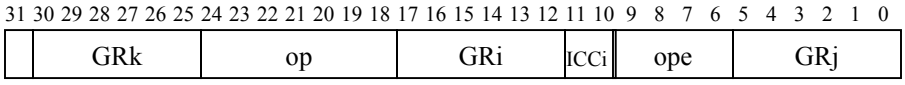
Instruction Format (NOT Operation (R-R))



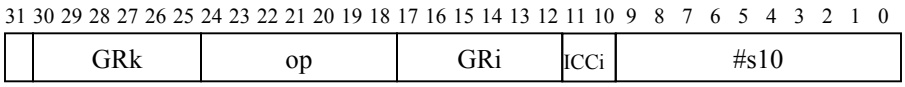
Instruction Format (INT, Logic, Shift Operation (R-simm))



Instruction Format (INT, Logic, Shift-cc Operation (R-R))



Instruction Format (INT, Logic, Shift-cc Operation (R-simm))



Assembler Syntax

AND	GRi, GRj, GRk
ANDcc	GRi, GRj, GRk, ICCi
OR	GRi, GRj, GRk
ORcc	GRi, GRj, GRk, ICCi
XOR	GRi, GRj, GRk
XORcc	GRi, GRj, GRk, ICCi
NOT	GRj, GRk
ANDI	GRi, #s12, GRk
ANDIcc	GRi, #s10, GRk, ICCi
ORI	GRi, #s12, GRk
ORIcc	GRi, #s10, GRk, ICCi
XORI	GRi, #s12, GRk
XORIcc	GRi, #s10, GRk, ICCi

Description

The integer logical instructions execute logical operations on GRi and GRj bit by bit (immediate instruction: GRi and sign_ext (#s12) or sign_ext (#s10)) and write the result in GRk. The ANDcc, ORcc, and XORcc instructions and their immediate instructions change the integer condition code (ICC) specified by ICCi field.

Registers altered (except destination register)

ICCi … instructions with “cc” only

N	Z	V	C
O	O	X	X

Occurrence Exceptions

register_exception (unimplement_exception)

Detected Exceptions

None

1.2.7. Shift (SLASS instruction is available for MB93405/MB93451.)

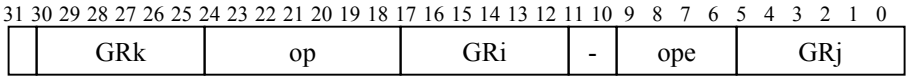
Ope-code	op	ope	Operation
SLL	0000001	1000	Shift Left Logical
SLLcc	0000001	1001	Shift Left Logical and ICC setting
SRL	0000001	1010	Shift Right Logical
SRLcc	0000001	1011	Shift Right Logical and ICC setting
SRA	0000001	1100	Shift Right Arithmetic
SRAcc	0000001	1101	Shift Right Arithmetic and ICC setting
SLLI	0101000	-	Shift Left Logical (Immediate)
SLLIcc	0101001	-	Shift Left Logical and ICC setting (Immediate)
SRLI	0101010	-	Shift Right Logical (Immediate)
SRLIcc	0101011	-	Shift Right Logical and ICC setting (Immediate)
SRAI	0101100	-	Shift Right Arithmetic (Immediate)
SRAIcc	0101101	-	Shift Right Arithmetic and ICC setting (Immediate)
SLASS	1000110	000010	Shift Left Arithmetic with Signed Saturation

 # : SLASS is available for MB93405/MB93451.

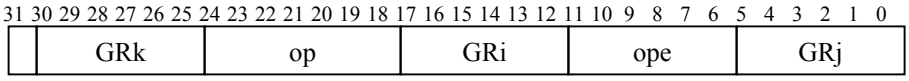
Category

Integer

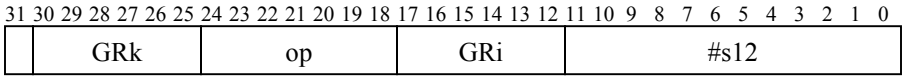
Instruction Format (INT, Logic, Shift Operation (R-R))



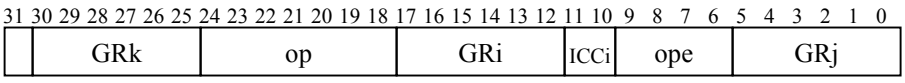
Instruction Format (SLASS)



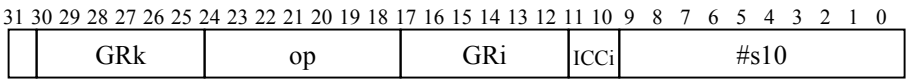
Instruction Format (INT, Logic, Shift Operation (R-simm))



Instruction Format (INT, Logic, Shift-cc Operation (R-R))



Instruction Format (INT, Logic, Shift-cc Operation (R-simm))



Assembler Syntax

SLL	GRi, GRj, GRk
SLLcc	GRi, GRj, GRk, ICCi
SRL	GRi, GRj, GRk
SRLcc	GRi, GRj, GRk, ICCi
SRA	GRi, GRj, GRk
SRAcc	GRi, GRj, GRk, ICCi
SLLI	GRi, #s12, GRk
SLLIcc	GRi, #s10, GRk, ICCi
SRLI	GRi, #s12, GRk
SRLIcc	GRi, #s10, GRk, ICCi
SRAI	GRi, #s12, GRk
SRAIcc	GRi, #s10, GRk, ICCi
SLASS	GRi, GRj, GRk

Description

The integer shift instructions shift GRi by the number of bits implied by the shift-count and write the result in GRk. The shift-count is specified by the low-order 5 bits of GRj (immediate instruction: sign_ext (#s12) or sign_ext (#s10)).
The SLL instruction shifts GRi to the left, replacing the vacated positions with zero.
The SRL instruction shifts GRi to the right, replacing the vacated positions with zero.
The SRA instruction shifts GRi to the right, replacing the vacated positions with the highest bit of GRi.
The SLLcc SRLcc, and SRAcc instructions and their immediate instructions change the integer condition code (ICC) specified by ICCi field.
Each bit of shift out is calculated in logical OR, and result in the C flag of ICC in SRAcc instruction.
Each bit of shift out is calculate in logical OR, and result in the V flag of ICC in SLLcc instruction.
SLASS shifts a 32bit value of GRi toward left at a 32bit value of GRj, and saturate,to pruduce a 32bit result. The result is placed into GRk. If the result is overflow as 32bit signed integer, maximum value of 32bit signed interger placed into GRk. (0x7FFFFFFF or 0x80000000)

Registers altered (except destination register)

ICCi … instructions with “cc” only

SRLcc,SRLIcc

N	Z	V	C
O	O	X	X

SLLcc,SLLIcc

N	Z	V	C
O	O	O	X

SRAcc,SRAIcc

N	Z	V	C
O	O	X	O

Occurrence Exceptions

register_exception (unimplement_exception)

Detected Exceptions

none

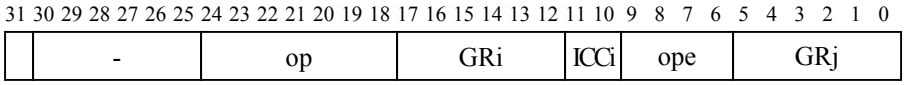
1.2.8. Byte Compare Instruction

Ope code	op	ope	Operation
CMPB	0000000	1100	Byte Comparison and ICC field
CMPBA	0000000	1101	OR of Byte Comparison and ICC field

Category

Integer

Instruction format (INT, Logic, Shift-cc operation (R – R))



Assembler description

CMPB GRi,GRj,ICCi
CMPBA GRi,GRj,ICCi

Description

The CMPB instruction compares GRi_0 with GRj_0; when they match, the instruction writes 1 to ICCi.n; when they do not match, the instruction writes 0 to ICCi.n. Simultaneously, the instruction compares GRi_1 with GRj_1, and when they match, the instruction writes 1 to ICCi.z; when they do not match, the instruction writes 0 to ICCi.z. Simultaneously, the instruction compares GRi_2 with GRj_2, and when they match, the instruction writes 1 to ICCi.v; when they do not match, the instruction writes 0 to ICCi.v. Simultaneously, the instruction compares GRi_3 with GRj_3, and when they match, the instruction writes 1 to ICCi.c; when they do not match, the instruction writes 0 to ICCi.c.
The CMPBA instruction compares GRi_0 with GRj_0; GRi_1 with GRj_1; GRi_2 with GRj_2; and GRi_3 with GRj_3, respectively; when there is at least one match, the instruction writes 1 to ICCi.c; when there is no match at all, the instruction writes 0 to ICCi.c. The instruction writes 0 to ICCi.n**,** ICCi.z**,** and ICCi.v.
Indexes used for GRi and GRj show the byte locations in the register. The relationship between byte location and bit position is shown below:

GRi0 = GRi (bit 31 to bit 24), GRi1 = GRi (bit 23 to bit 16)
GRi2 = GRi (bit 15 to bit 8), GRi3 = GRi (bit 7 to bit 0)
Note: The same relationship also applies to GRj.

Registers altered (except destination register)

None

Occurrence Exceptions

register_exception (unimplement_exception)

Detected Exceptions

None

1.2.9. Accumulator Cut Instruction (This instruction is available for MB93405/MB93451.)

Ope code	op	ope	Operation
SCUTSS	1000110	000100	Signed Cut with Signed Saturation (Immediate)

Category

Integer

Instruction format (SCTUTSS)

31	30 29 28 27	26 25 24 23 22 21	20 19 18 17 16 15 14 13 12 11 10	9 8 7 6	5 4 3 2 1 0
	GRk	op	-	ope	GRj

Assembler description

SCUTSS GRj,GRk

Description

SCUTSS round the 64bit value of Iacc with bit_position of 7bit signed value of GRj, and cut and saturate, to pruduce a 32bit result. The result is placed into GRk. At round phase, if (32-bit_posion)th bit of Iacc is '1', 2^(32-bit_position) is added to IACC. At cut phase, if bit_position is minus vlaue, the value of IACC is entended with sign bit. At saturation phase, if the result is overflow as 32bit signed integer, maximum value of 32bit signed interger placed into GRk. (0x7FFFFFFF or 0x80000000)

Ex.

IACC : FFFFFEDC BA987654 : without round_increment, saturation case
GRj : 16(00000010)
GRk : FEDCBA98

IACC : FFFFFEDC BA987654 : with round_increment case
GRj : 12(0000000c)
GRk : FFEDCBAA

IACC : FFFFFEDC BA987654 : with sign extention case
GRj : -4(FFFFFFFC)
GRk : FFFFFFEE

IACC : FFFFFEDC BA987654 : with saturation case
GRj : 24(00000018)
GRk : 80000000

Registers altered (except destination register)

None

Occurrence Exceptions

register_exception (unimplement_exception)

Detected Exceptions

None

1.3. Load/Store Instructions

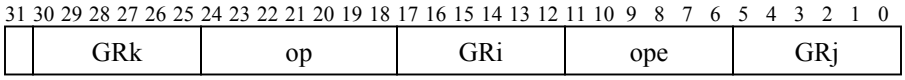
1.3.1. Load GR

Ope-code	op	ope	Operation
LDSB	0000010	000000	Load Signed Byte
LDUB	0000010	000001	Load Unsigned Byte
LDSH	0000010	000010	Load Signed Halfword
LDUH	0000010	000011	Load Unsigned Halfword
LD	0000010	000100	Load Word
LDD	0000010	000101	Load Double
LDSBU	0000010	010000	Load Signed Byte with Update Index
LDUBU	0000010	010001	Load Unsigned Byte with Update Index
LDSHU	0000010	010010	Load Signed Halfword with Update Index
LDUHU	0000010	010011	Load Unsigned Halfword with Update Index
LDU	0000010	010100	Load Word with Update Index
LDDU	0000010	010101	Load Double with Update Index
LDSBI	0110000	-	Load Signed Byte (Immediate)
LDSHI	0110001	-	Load Signed Halfword (Immediate)
LDI	0110010	-	Load Word (Immediate)
LDDI	0110011	-	Load Double (Immediate)
LDUBI	0110101	-	Load Unsigned Byte (Immediate)
LDUHI	0110110	-	Load Unsigned Halfword (Immediate)

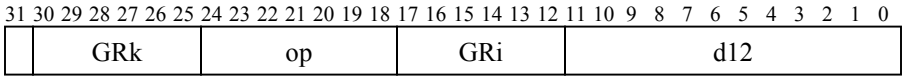
Category

Integer

Instruction Format (Load/Store (R-R))



Instruction Format (Load/Store (R-simm))



Assembler Syntax

LDSB @(GRi, GRj), GRk
LDUB @(GRi, GRj), GRk
LDSH @(GRi, GRj), GRk
LDUH @(GRi, GRj), GRk
LD @(GRi, GRj), GRk
LDD @(GRi, GRj), GRk
LDSBU @(GRi, GRj), GRk

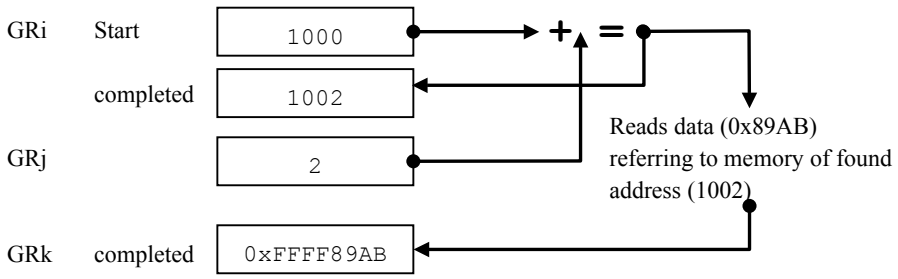
LDUBU @(GRi, GRj), GRk
LDSHU @(GRi, GRj), GRk
LDUHU @(GRi, GRj), GRk
LDU @(GRi, GRj), GRk
LDDU @(GRi, GRj), GRk
LDSBI @(GRi, d12), GRk
LDUBI @(GRi, d12), GRk
LDSHI @(GRi, d12), GRk
LDUHI @(GRi, d12), GRk
LDI @(GRi, d12), GRk
LDDI @(GRi, d12), GRk

Description

The integer Load instructions calculate "GRi + GRj" as an effective address (immediate instruction: "GRi + sign_ext (d12)") and copy data from memory to a general-purpose register (GR). Byte or halfword data is written in the GRk register which is right-justified and it is either sign-extended or zero-extended on the left, depending on whether or not the opecode specifies a signed or unsigned operation, respectively. In doubleword Load instruction, it is necessary to set the register number of GRk as an even number with software, and the high-order word is written in the even register and the low-order word in the odd. (sign_ext(X): X extended with a sign)
An instruction with "update" form calculates "GRi + GRj" as an effective address and writes the result in GRi.
The mem_address_not_aligned exception occurs when the effective address is not aligned on a halfword boundary in halfword Load instruction, when it is not aligned on a word boundary in word Load instruction or when it is not aligned on a doubleword boundary in doubleword Load instruction.
The register_exception (register_not_aligned) exception occurs when the register number of GRk is not an even number in doubleword Load instruction.
When an imprecise exception occurs in the Load with Update Index instruction, it completes to write the effective address in GRi and it does not complete to copy data from memory indicating by the effective address to GRk.

Operation example

LDSHU @(GRi,GRj),GRk



Registers altered (except destination register)

GRi … Instruction with "update" form only

Occurrence Exceptions

mem_address_not_aligned (except byte Load instruction)

data_access_exception
data_access_MMU_miss
data_access_error
register_exception (unimplement_exception, register_not_aligned)

Detected Exceptions

none

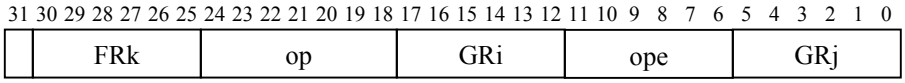
1.3.2. Load FR

Ope-code	op	ope	Operation
LDBF	0000010	001000	Load Byte FR register
LDHF	0000010	001001	Load Halfword FR register
LDF	0000010	001010	Load FR register
LDDF	0000010	001011	Load Double FR register
LDBFU	0000010	011000	Load Byte FR register with Update Index
LDHFU	0000010	011001	Load Halfword FR register with Update Index
LDFU	0000010	011010	Load FR register with Update Index
LDDFU	0000010	011011	Load Double FR register with Update Index
LDBFI	0111000	-	Load Byte FR register (Immediate)
LDHFI	0111001	-	Load Halfword FR register (Immediate)
LDFI	0111010	-	Load FR register (Immediate)
LDDFI	0111011	-	Load Double FR register (Immediate)

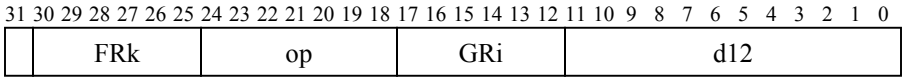
Category

Integer

Instruction Format (Load/Store (R-R))



Instruction Format (Load/Store (R-simm))



Assembler Syntax

LDBF	@(GRi, GRj), FRk
LDHF	@(GRi, GRj), FRk
LDF	@(GRi, GRj), FRk
LDDF	@(GRi, GRj), FRk
LDBFU	@(GRi, GRj), FRk
LDHFU	@(GRi, GRj), FRk
LDFU	@(GRi, GRj), FRk
LDDFU	@(GRi, GRj), FRk
LDBFI	@(GRi, d12), FRk
LDHFI	@(GRi, d12), FRk
LDFI	@(GRi, d12), FRk
LDDFI	@(GRi, d12), FRk

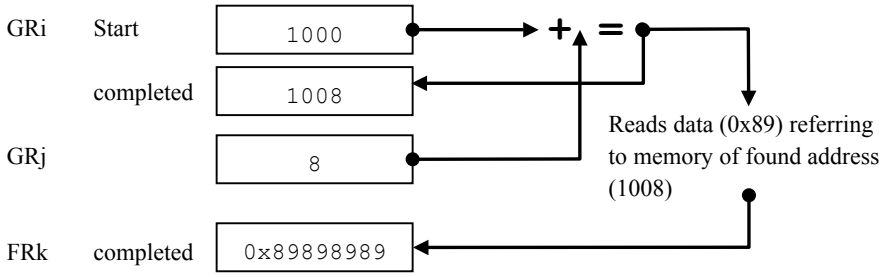
Description

The floating-point Load instructions calculate “GRi + GRj” as an effective address (immediate instruction: “GRi + sign_ext (d12)”) and copy data from memory to FR.

The LDBF instruction copies byte data from memory to all byte positions of FR without a signed extension.
The LDHF instruction copies halfword aligned data from memory to all halfword positions of FR without a signed extension.
The LDF instruction copies word aligned data from memory to FR.
The LDDF instruction copies doubleword aligned data from memory to FRk and FRk+1.
An instruction with "update" form calculates "GRi + GRj" as an effective address and writes the result in GRi.
The mem_address_not_aligned exception occurs when the effective address is not aligned on a halfword boundary in halfword Load instruction, when it is not aligned on a word boundary in word Load instruction, when it is not aligned on a doubleword boundary in doubleword Load instruction.
The register_exception (register_not_aligned) exception occurs when the register number of FRk is not an even number in doubleword Load instruction.
When an imprecise exception occurs in the Load with Update Index instruction, it completes to write the effective address in GRi and it does not complete to copy data from memory indicating by the effective address to FRk.

Operation example

LDBFU @(GRi,GRj),FRk



Registers altered (except destination register)

GRi … Instruction with update only

Occurrence Exceptions

fp_disabled
mem_address_not_aligned
data_access_exception
data_access_MMU_miss
data_access_error
register_exception (unimplement_exception, register_not_aligned)

Detected Exceptions

none

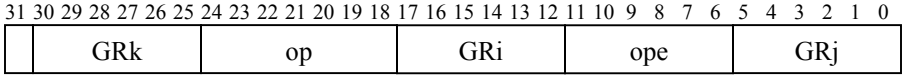
1.3.3. Store GR

Ope-code	op	ope	Operation
STB	0000011	000000	Store Byte
STH	0000011	000001	Store Halfword
ST	0000011	000010	Store Word
STD	0000011	000011	Store Double
STBU	0000011	010000	Store Byte with Update Index
STHU	0000011	010001	Store Halfword with Update Index
STU	0000011	010010	Store Word with Update Index
STDU	0000011	010011	Store Double with Update Index
STBI	1010000	-	Store Byte (Immediate)
STHI	1010001	-	Store Halfword (Immediate)
STI	1010010	-	Store Word (Immediate)
STDI	1010011	-	Store Double (Immediate)

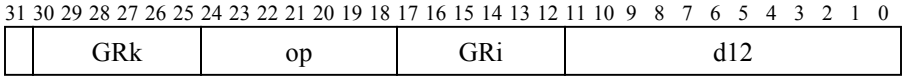
Category

Integer

Instruction Format (Load/Store (R-R))



Instruction Format (Load/Store (R-simm))



Assembler Syntax

STB	GRk, @(GRi, GRj)
STH	GRk, @(GRi, GRj)
ST	GRk, @(GRi, GRj)
STD	GRk, @(GRi, GRj)
STBU	GRk, @(GRi, GRj)
STHU	GRk, @(GRi, GRj)
STU	GRk, @(GRi, GRj)
STDU	GRk, @(GRi, GRj)
STBI	GRk, @(GRi, d12)
STHI	GRk, @(GRi, d12)
STI	GRk, @(GRi, d12)
STDI	GRk, @(GRi, d12)

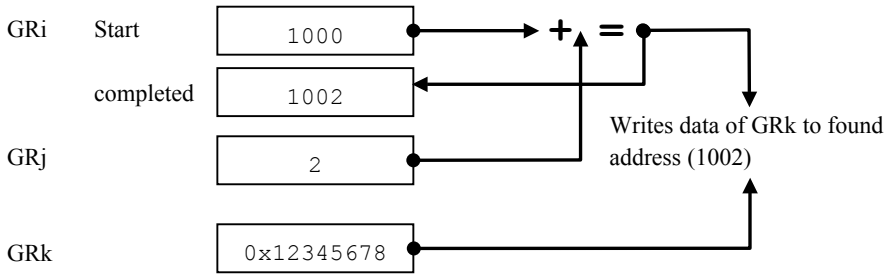
Description

The integer Store instructions calculate “GRi + GRj” as an effective address (immediate instruction: “GRi + sign_ext (d12)”) and copy data from a general-purpose register(GRk) into memory. In doubleword Store instruction, it is necessary to set the register number of GRk as

an even number with software, and the high-order word is stored in memory from the even register and the low-order word from the odd. (sign_ext (X): X extended with a sign)
An instruction with "update" form calculates "GRi + GRj" as an effective address and writes the result in GRi.
The mem_address_not_aligned exception occurs when the effective address is not aligned on a halfword boundary in halfword Store instruction, when it is not aligned on a word boundary in word Store instruction or when it is not aligned on a doubleword boundary in doubleword Store instruction.
The register_exception (register_not_aligned) occurs when the register number of GRk is an odd number in doubleword Store instruction.
When an imprecise exception occurs in the Store with Update Index instruction, it completes to write the effective address in GRi and it does not complete to copy data from GRk to memory indicating by the effective address.

Operation example

STU GRk,@(GRi,GRj)



Registers altered (except destination register)

GRi … Instruction with update only

Occurrence Exceptions

mem_address_not_aligned (except byte Store)
data_access_exception
data_access_MMU_miss
data_access_error
data_store_error
register_exception (unimplement_exception, register_not_aligned)

Detected Exceptions

none

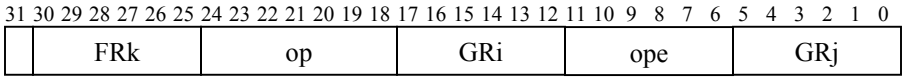
1.3.4. Store FR

Ope-code	op	ope	Operation
STBF	0000011	001000	Store Byte FR register
STHF	0000011	001001	Store Halfword FR register
STF	0000011	001010	Store FR register
STDF	0000011	001011	Store Double FR register
STBFU	0000011	011000	Store Byte FR register with Update Index
STHFU	0000011	011001	Store Halfword FR register with Update Index
STFU	0000011	011010	Store FR register with Update Index
STDFU	0000011	011011	Store Double FR register with Update Index
STBFI	1001110	-	Store Byte FR register (Immediate)
STHFI	1001111	-	Store Halfword FR register (Immediate)
STFI	1010101	-	Store FR register (Immediate)
STDFI	1010110	-	Store Double FR register (Immediate)

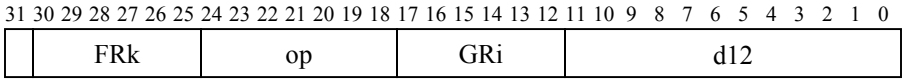
Category

Integer

Instruction Format (Load/Store (R-R))



Instruction Format (Load/Store (R-simm))



Assembler Syntax

STBF	FRk, @(GRi, GRj)
STHF	FRk, @(GRi, GRj)
STF	FRk, @(GRi, GRj)
STDF	FRk, @(GRi, GRj)
STBFU	FRk, @(GRi, GRj)
STHFU	FRk, @(GRi, GRj)
STFU	FRk, @(GRi, GRj)
STDFU	FRk, @(GRi, GRj)
STBFI	FRk, @(GRi, d12)
STHFI	FRk, @(GRi, d12)
STFI	FRk, @(GRi, d12)
STDFI	FRk, @(GRi, d12)

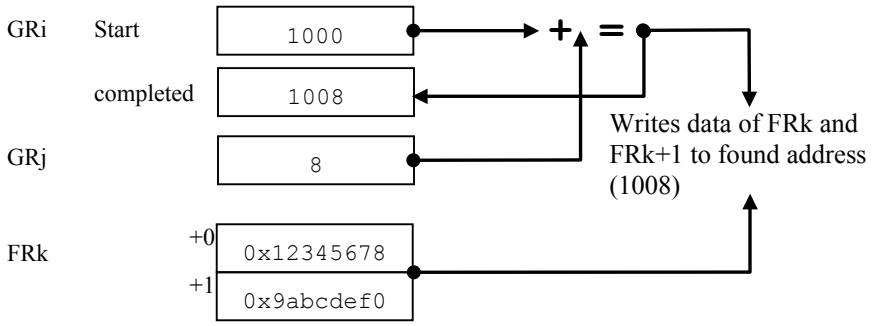
Description

The floating-point Store instructions calculate “GRi + GRj” as an effective address (immediate instruction: “GRi + sign_ext (d12)”) and copy data from FRk into memory.

The STBF instruction copies the lowest-order byte of FR to memory.
The STHF instruction copies low-order halfword data of FR to memory.
The STF instruction copies word data from FR to memory.
The STDF instruction copies doubleword data from FRk and FRk+1 to memory.
An instruction with "update" form calculates "GRi + GRj" as an effective address and writes the result in GRi.
The mem_address_not_aligned exception occurs when the effective address is not aligned on a halfword boundary in halfword Store instruction, when it is not aligned on a word boundary in word Store instruction or when it is not aligned on a doubleword boundary in doubleword Store instruction.
When an imprecise exception occurs in executing the Store with Update Index instruction, it completes to write the effective address in GRi and it does not complete to copy data from FRk to memory indicated by the effective address.

Operation example

STDFU FRk,@(GRi,GRj)



Registers altered (except destination register)

GRi … Instruction with update only

Occurrence Exceptions

fp_disabled
mem_address_not_aligned
data_access_exception
data_access_MMU_miss
data_access_error
data_store_error
register_exception (unimplement_exception, register_not_aligned)

Detected Exceptions

none

1.4. Data transfer Instructions

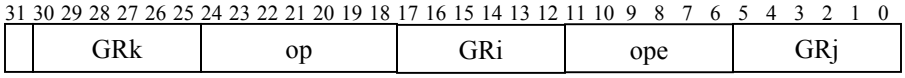
1.4.1. Swap

Ope-code	op	ope	Operation
SWAP	0000011	000101	SWAP register with memory
SWAPI	1001101	-	SWAP register with memory (Immediate)

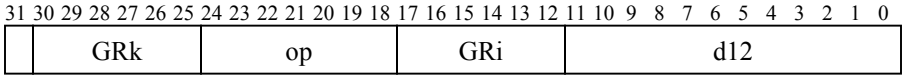
Category

Control

Instruction Format (Load/Store (R-R))



Instruction Format (Load/Store (R-simm))



Assembler Syntax

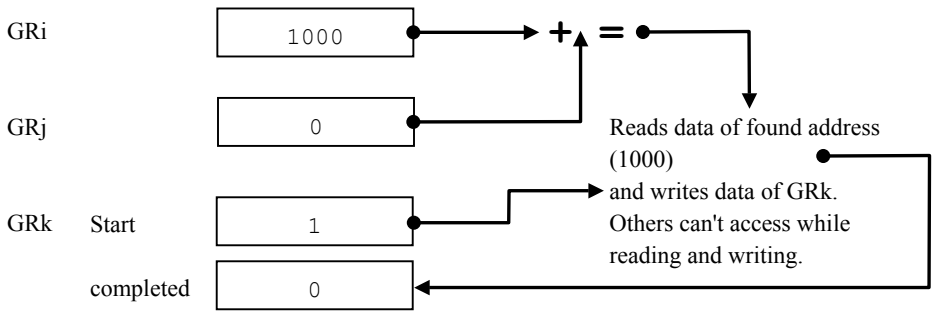
SWAP @(GRi, GRj), GRk
SWAPI @(GRi, d12), GRk

Description

The swap instructions calculates “GRi+GRj“ as an effective address (Immediate instruction: “GRi+sign_ext (d12)”) and exchange the contents of GRk with the contents of the word addressed memory location.The operation is performed atomically, that is, without allowing intervening interrupts. In a multiprocessor system, two or more processors executing SWAP or atomic Load-Store instruction addressing the same word or byte simultaneously are guaranteed to be executed without allowing intervening.
The mem_address_not_aligned exception occurs when the effective address is not aligned on a word boundary.

Operation example

SWAP @(GRi,GRj),GRk



Registers altered (except destination register)

none

Occurrence Exceptions

mem_address_not_aligned
data_access_exception
data_access_MMU_miss
data_access_error
data_store_error
register_exception (unimplement_exception)

Detected Exceptions

none

1.4.2. Move

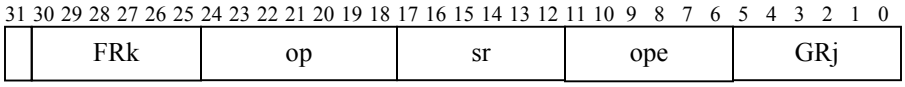
Ope-code	op	ope	Operation
MOVGF	0000011	010101	Move GR to FR
MOVGFD	0000011	010110	Move GR to FR Double
MOVFG	0000011	001101	Move FR to GR
MOVFGD	0000011	001110	Move FR to GR Double
MOVGS(*1)	0000011	000110	Move GR to SPR
MOVSG(*1)	0000011	000111	Move SPR to GR

(*1) privileged instruction if source register or destination register is privileged.

Category

Integer, Control (MOVGS, MOVSG)

Instruction Format (Register transfer instruction)



Assembler Syntax

MOVGF GRj, FRk
MOVGFD GRj, FRk
MOVFG FRk, GRj
MOVFGD FRk, GRj
MOVGS GRj, SPR
MOVSG SPR, GRj
("SPR" in assembler syntax is 12 bits field concatenating the FRk field and the sr field, following FRk with sr, in MOVGS instruction and MOVSG instruction.)

Description

The register transfer instructions copy the contents between GR, FR, and SPR.
The instruction MOVGF copies the contents of GRj to FRk.
The instruction MOVGFD copies the contents of GRj to FRk and from GRj+1 to FRk+1
The instruction MOVFG copies the contents of FRk to GRj.
The instruction MOVFGD copies the contents of FRk to GRj and from FRk+1 to GRj+1
The instruction MOVGS copies the contents of GRj to SPR.
The instruction MOVSG copies the contents of SPR to GRj.
The register_exception (register_not_aligned) occurs when the register number of FRk or GRj is an odd number in doubleword transfer.

Registers altered (except destination register)

none

Occurrence Exceptions

fp_disabled
register_exception (unimplement_exception, register_not_aligned)
privileged_instruction

Detected Exceptions

none

1.5. Control transfer Instructions

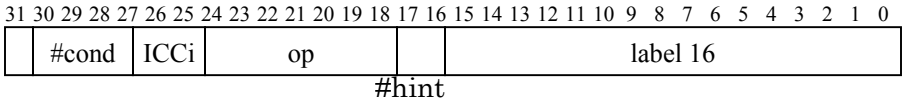
1.5.1. Integer Conditional Branch

Ope-code	op	Operation
Bicc	0000110	Integer Conditional Branch

Category

Branch

Instruction Format (Branch instruction)



Assembler Syntax

Bicc ICCi, #hint, label16

BEQ : Z==1
BNE : Z==0
BLE : (Z or(N xor V))==1
BGT : (Z or (N xor V))==0
BLT : (N xor V)==1
BGE : (N xor V)==0
BLS : (C or Z)==1
BHI : (C or Z)==0
BC : C==1
BNC : C==0
BN : N==1
BP : N==0
BV : V==1
BNV : V==0

BNO
BRA label16

Description

The Integer Conditional Branch instruction evaluates the #cond field of ICCi, according to the #cond field of the instruction.
When the evaluation is true, the instruction cause control transfer to the address "PC + (4 x sign_ext (label 16))". When the evaluation is false, the branch is not taken.
The #hint bit is 2bit field which is set by software, and the hint bit is used for a branch prediction. When there is a high probability of branching, you should specify 2 to #hint. When there is not much probability of branching, you should specify 0 to #hint.
The following table is shown the evaluation condition of ICCi indicating by #cond field.

Table 1 #cond field and pseudo opecode

Pseudo opecode	cond	mean	ICC test
BEQ	0100	Branch Equal	Z
BNE	1100	Branch Not Equal	not Z
BLE	0111	Branch Less or Equal	Z or (N xor V)
BGT	1111	Branch Greater	Not (Z or (N xor V))
BLT	0011	Branch Less	N xor V
BGE	1011	Branch Greater or Equal	Not (N xor V)
BLS	0101	Branch Less or Equal Unsigned	C or Z
BHI	1101	Branch Greater Unsigned	Not (C or Z)
BC	0001	Branch Carry Set	C
BNC	1001	Branch Carry Clear	Not C
BN	0110	Branch Negative	N
BP	1110	Branch Positive	Not N
BV	0010	Branch Overflow Set	V
BNV	1010	Branch Overflow Clear	Not V
BNO	0000	Branch Never	0
BRA	1000	Branch Always	1

Registers altered (except destination register)

none

Occurrence Exceptions

none

Detected Exceptions

none

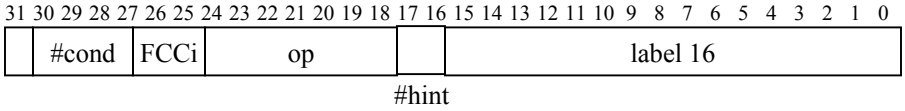
1.5.2. Floating-point / media Conditional Branch

Ope-code	op	Operation
FBfcc	0000111	Floating-point/Media Conditional Branch

Category

Branch

Instruction Format (Branch instruction)



Assembler Syntax

```
FBfcc       FCCi, #hint, label16
      FBNE      : (L or G or U)==1
      FBEQ      : E==1
      FBLG      : (L or G)==1
      FBUE      : (E or U)==1
      FBUL      : (L or U)==1
      FBGE      : (E or G)==1
      FBLT      : L == 1
      FBUGE     : (E or G or U)==1
      FBUG      : (G or U)==1
      FBLE      : L==1
      FBGT      : G==1
      FBULE     : (E or L or G)==1
      FBU       : U==1
      FBO       : (E or L or G)==1
FBNO
FBRA   label16
```

Description

The floating-point / media Conditional Branch instruction evaluates the #cond field of FCCi, according to the #cond field of the instruction.

When the evaluation is true, the instruction cause control transfer to the address "PC + (4 x sign_ext (label 16))". When the evaluation is false, the branch is not taken.

The #hint bit is 2-bit field which is set by software, and the hint bit is used for a branch prediction. When there is a high probability of branching, you should specify 2 to #hint. When there is not much probability of branching, you should specify 0 to #hint.

The following table is shown the evaluation condition of FCCi indicating by #cond field.

Table 2 #cond field and pseudo opecode

Pseudo opecode	#cond	mean	FCC test
FBNE	0111	Branch Not Equal	L or G or U
FBEQ	1000	Branch Equal	E
FBLG	0110	Branch Less or Greater	L or G
FBUE	1001	Branch Unordered or Equal	E or U
FBUL	0101	Branch Unordered or Less	L or U
FBGE	1010	Branch Greater or Equal	E or G
FBLT	0100	Branch Less	L
FBUGE	1011	Branch Unordered or Greater or Equal	E or G or U
FBUG	0011	Branch Unordered or Greater	G or U
FBLE	1100	Branch Less or Equal	E or L
FBGT	0010	Branch Greater	G
FBULE	1101	Branch Unordered or Less or Equal	E or L or U
FBU	0001	Branch Unordered	U
FBO	1110	Branch Ordered	E or L or G
FBNO	0000	Branch Never	0
FBRA	1111	Branch Always	1

Registers altered (except destination register)

none

Occurrence Exceptions

fp_disabled

Detected Exceptions

none

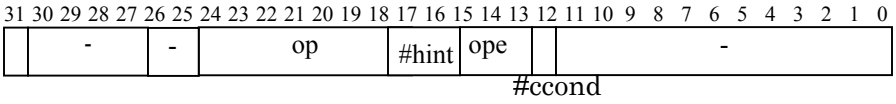
1.5.3. LCR Conditional Branch to LR

Ope-code	Op	ope	Operation
BctrLR	0001110	001	LCR Conditional Branch to LR

Category

Branch

Instruction Format (Branch instruction)



Assembler Syntax

BctrLR #ccond, #hint

Description

BctrLR instruction decrements the LCR and evaluates the condition shown by the following table. When the evaluation is true, the instruction cause control transfer to the address stored in the LR. When the evaluation is false, the branch is not taken.

Table 3 #ccond field and evaluation condition of LCR

#ccond	Operation	LCR test
0	Branch if LCR != 0	LCR != 0
1	Branch if LCR = 0	LCR = 0

Registers altered (except destination register)

none

Occurrence Exceptions

none

Detected Exceptions

none

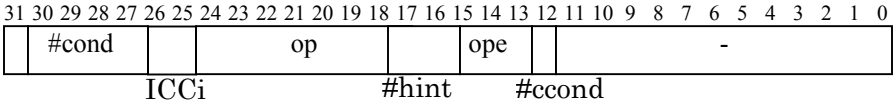
1.5.4. Integer conditional Branches to LR

Ope-code	op	ope	Operation
BiccLR	0001110	010	Integer Conditional Branch to LR
BCiccLR	0001110	011	Integer and LCR Conditional Branch to LR

Category

Branch

Instruction Format (Branch instruction)



Assembler Syntax

```
BiccLR   ICCi,  #hint
      BEQLR     : Z==1
      BNELR     : Z==0
      BLELR     : (Z or (N xor V))==1
      BGTLR     : (Z or (N xor V))==0
      BLTLR     : (N xor V)==1
      BGELR     : (N xor V)==0
      BLSLR     : (C or Z)==1
      BHILR     : (C or Z)==0
      BCLR      : C==1
      BNCLR     : C==0
      BNLR      : N==1
      BPLR      : N==0
      BVLR      : V==1
      BNVLR     : V==0
BNOLR
BRALR
```

BCiccLR ICCi, #ccond, #hint
BCEQLR : Z==1
BCNELR : Z==0
BCLELR : (Z or (N xor V))==1
BCGTLR : (Z or (N xor V))==0
BCLTLR : (N xor V)==1
BCGELR : (N xor V)==0
BCLSLR : (C or Z)==1
BCHILR : (C or Z)==0
BCCLR : C==1
BCNCLR : C==0
BCNLR : N==1
BCPLR : N==0
BCVLR : V==1
BCNVLR : V==0
BCNOLR
BCRALR #ccond

Description

BiccLR instruction evaluates the #cond field of ICCi, according to the #cond field of the instruction. When the evaluation is true, the instruction cause control transfer to the address stored in the LR. When the evaluation is false, the branch is not taken.
BCiccLR instruction decrements the LCR and evaluates the condition shown Table 4 and also evaluates the #cond field of ICCi, according to the #cond field of the instruction. When the both evaluations are true, the instruction cause control transfer to the address stored in the LR. When the either condition is false, the branch is not taken.
Even if the Integer conditional Branch to LR Instructions and another branch instructions are in a same VLIW instruction and the condition of another branch instructions are true, the decrement of LCR in Integer condition Branch to LR instruction is executed.
The #hint bit is 2-bit field which is set by software, and the hint bit is used for a branch prediction. When there is a high probability of branching, you should specify 2 to #hint. When there is not much probability of branching, you should specify 0 to #hint.

Registers altered (except destination register)

none

Occurrence Exceptions

none

Detected Exceptions

none

Table 4 #ccond field and evaluation condition of LCR

#ccond	Operation	LCR test
0	Branch if LCR != 0	LCR != 0
1	Branch if LCR = 0	LCR = 0

Table 5 #cond field and pseudo opecode

Pseudo opecode	ope	#cond	Operation	ICC test
BEQLR	010	0100	Branch Equal to LR	Z
BNELR	010	1100	Branch Not Equal to LR	Not Z
BLELR	010	0111	Branch Less or Equal to LR	Z or (N xor V)
BGTLR	010	1111	Branch Greater to LR	Not (Z or (N xor V))
BLTLR	010	0011	Branch Less to LR	N xor V
BGELR	010	1011	Branch Greater or Equal to LR	Not (N xor V)
BLSLR	010	0101	Branch Less or Equal Unsigned to LR	C or Z
BHILR	010	1101	Branch Greater Unsigned to LR	Not (C or Z)
BCLR	010	0001	Branch Carry Set to LR	C
BNCLR	010	1001	Branch Carry Clear to LR	Not C
BNLR	010	0110	Branch Negative to LR	N
BPLR	010	1110	Branch Positive to LR	Not N
BVLR	010	0010	Branch Overflow Set to LR	V
BNVLR	010	1010	Branch Overflow Clear to LR	Not V
BNOLR	010	0000	Branch Never to LR	0
BRALR	010	1000	Branch Always to LR	1
BCEQLR	011	0100	Branch LCR and Equal to LR	Z
BCNELR	011	1100	Branch LCR and Not Equal to LR	Not Z
BCLELR	011	0111	Branch LCR and Less or Equal to LR	Z or (N xor V)
BCGTLR	011	1111	Branch LCR and Greater to LR	Not (Z or (N xor V))
BCLTLR	011	0011	Branch LCR and Less to LR	N xor V
BCGELR	011	1011	Branch LCR and Greater or Equal to LR	Not (N xor V)
BCLSLR	011	0101	Branch LCR and Less or Equal Unsigned to LR	C or Z
BCHILR	011	1101	Branch LCR and Greater Unsigned to LR	Not (C or Z)
BCCLR	011	0001	Branch LCR and Carry Set to LR	C
BCNCLR	011	1001	Branch LCR and Carry Clear to LR	Not C
BCNLR	011	0110	Branch LCR and Negative to LR	N
BCPLR	011	1110	Branch LCR and Positive to LR	Not N
BCVLR	011	0010	Branch LCR and Overflow Set to LR	V
BCNVLR	011	1010	Branch LCR and Overflow Clear to LR	Not V
BCNOLR	011	0000	Branch LCR and Never to LR	0
BCRALR	011	1000	Branch LCR and Always to LR	1

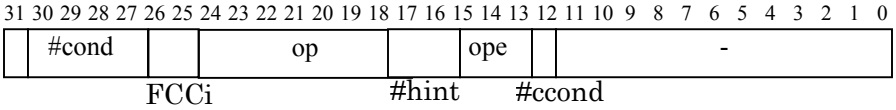
1.5.5. Floating-point/Media Branches to LR

Ope-code	Op	ope	Operation
FBfccLR	0001110	110	Floating-point/Media Conditional Branch to LR
FCBfccLR	0001110	111	Floating-point/Media and LCR Conditional Branch to LR

Category

Branch

Instruction Format (Branch instruction)



Assembler Syntax

```
FBfccLR    FCCi, #hint
      FBNELR      : (L or G or U)==1
      FBEQLR      : E==1
      FBLGLR      : (L or G)==1
      FBUELR      : (E or U)==1
      FBULLR      : (L or U)==1
      FBGELR      : (E or G)==1
      FBLTLR      : L==1
      FBUGELR     : (E or G or U)==1
      FBUGLR      : (G or U)==1
      FBLELR      : (E or L)==1
      FBGTLR      : G==1
      FBULELR     : (E or L or U)==1
      FBULR       : U==1
      FBOLR       : (E or L or G)==1
   FBNOLR
   FBRALR
```

```
FCBfccLR    FCCi, #ccond, #hint
      FCBNELR       : (L or G or U)==1
      FCBEQLR       : E==1
      FCBLGLR       : (L or G)==1
      FCBUELR       : (E or U)==1
      FCBULLR       : (L or U)==1
      FCBGELR       : (E or G)==1
      FCBLTLR       : L==1
      FCBUGELR      : (E or G or U)==1
      FCBUGLR       : (G or U)==1
      FCBLELR       : (E or L)==1
      FCBGTLR       : G==1
      FCBULELR      : (E or L or U)==1
      FCBULR        : U==1
      FCBOLR        : (E or L or G)==1
FCBNOLR
FCBRALR   #ccond
```

Description

FBfccLR instruction evaluates the #cond field of FCCi, according to the pseudo opecode shown by Table 7. When the evaluation is true, the instruction cause control transfer to the address stored in the LR. When the evaluation is false, the branch is not taken.

FCBfccLR instruction decrements the LCR and evaluates the condition shown by Table 6 and also evaluates the #cond field of FCCi, according to the pseudo opecode shown by Table 7. When the both evaluations are true, the instruction cause control transfer to the address stored in the LR. When the either condition is false, the branch is not taken.

Even if the Floating-point/Media Branch instruction to LR Instructions and other branch instructions are in the same VLIW instruction and the condition of the other branch instructions are true, the decrement of LCR in the FCBfccLR instruction is executed.

The #hint is 2-bit field which is set by software, and the hint bit is used for a branch prediction. When there is a high probability of branching, you should specify 2 to #hint. When there is not much probability of branching, you should specify 0 to #hint.

Registers altered (except destination register)

none

Occurrence Exceptions

fp_disabled

Detected Exceptions

none

Table 6 #ccond field and evaluation condition of LCR

#ccond	Operation	LCR test
0	Branch if LCR!= 0	LCR!= 0
1	Branch if LCR = 0	LCR = 0

Table 7 #cond field and pseudo opecode

Pseudo opecode	ope	#cond	Operation	FCC test
FBNELR	110	0111	Branch Not Equal to LR	L or G or U
FBEQLR	110	1000	Branch Equal to LR	E
FBLGLR	110	0110	Branch Less or Greater to LR	L or G
FBUELR	110	1001	Branch Unordered or Equal to LR	E or U
FBULLR	110	0101	Branch Unordered or Less to LR	L or U
FBGELR	110	1010	Branch Greater or Equal to LR	E or G
FBLTLR	110	0100	Branch Less to LR	L
FBUGELR	110	1011	Branch Unordered or Greater or Equal to LR	E or G or U
FBUGLR	110	0011	Branch Unordered or Greater to LR	G or U
FBLELR	110	1100	Branch Less or Equal to LR	E or L
FBGTLR	110	0010	Branch Greater to LR	G
FBULELR	110	1101	Branch Unordered or Less or Equal to LR	E or L or U
FBULR	110	0001	Branch Unordered to LR	U
FBOLR	110	1110	Branch Ordered to LR	E or L or G
FBNOLR	110	0000	Branch Never to LR	0
FBRALR	110	1111	Branch Always to LR	1
FCBNELR	111	0111	Branch LCR and Not Equal to LR	L or G or U
FCBEQLR	111	1000	Branch LCR and Equal to LR	E
FCBLGLR	111	0110	Branch LCR and Less or Greater to LR	L or G
FCBUELR	111	1001	Branch LCR and Unordered or Equal to LR	E or U
FCBULLR	111	0101	Branch LCR and Unordered or Less to LR	L or U
FCBGELR	111	1010	Branch LCR and Greater or Equal to LR	E or G
FCBLTLR	111	0100	Branch LCR and Less	L
FCBUGELR	111	1011	Branch LCR and Unordered or Greater or Equal to LR	E or G or U
FCBUGLR	111	0011	Branch LCR and Unordered or Greater to LR	G or U
FCBLELR	111	1100	Branch LCR and Less or Equal to LR	E or L
FCBGTLR	111	0010	Branch LCR and Greater to LR	G
FCBULELR	111	1101	Branch LCR and Unordered or Less or Equal	E or L or U
FCBULR	111	0001	Branch LCR and Unordered	U
FCBOLR	111	1110	Branch LCR and Ordered	E or L or G
FCBNOLR	111	0000	Branch LCR and Never to LR	0
FCBRALR	111	1111	Branch LCR and Always to LR	1

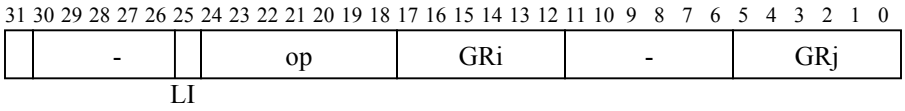
1.5.6. Jump and Link

Ope-code	Op	Operation
JMPL	0001100	Jump and Link
JMPIL	0001101	Jump (Immediate) and Link

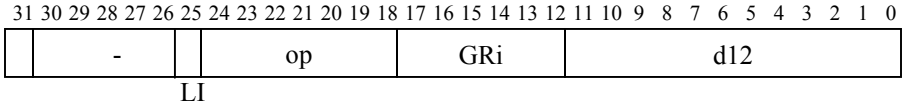
Category

Integer

Instruction Format (Jmp instruction (R-R))



Instruction Format (Jmp instruction (R-simm))



Assembler Syntax

JMPL	@(GRi,GRj)	(In case LI=0)
CALLL	@(GRi,GRj)	(In case LI=1)
JMPIL	@(GRi,d12)	(In case LI=0)
CALLIL	@(GRi,d12)	(In case LI=1)

Description

The JMPL instructions execute the unconditional branch to the branch target address given by “GRi+GRj” (Immediate: “GRi+sign_ext (d12)”)
When the LI field is ‘1’, the JMPL (CALLL or CALLIL) instruction writes the PC value of the first instruction of next VLIW instruction to LR. When the LI field is ‘0’, it does not write the value to the LR.
Even if the branch target address is not aligned on word boundary, the mem_address_not_aligned exception doesn't occur. In this case, the low-order bits below the word boundary in the branch target address are "0".

Registers altered (except destination register)

LR

Occurrence Exceptions

register_exception (unimplement_exception)

Detected Exceptions

none

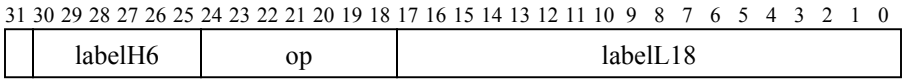
1.5.7. Call

Ope-code	op	Operation
CALL	0001111	Call and Link

Category

Branch

Instruction Format (Call instruction)



Assembler Syntax

CALL label24
(label 24 is 24 bits fields concatenating the labelH6 field and the labelL18 in the CALL instruction.)

Description

The CALL instruction executes the unconditional PC relative control transfer to the address "PC + (4 x sign_ext (label 24))", and writes the PC value of the first instruction in the next VLIW instruction in LR.
The PC relative displacement is 26-bit width which is following disp 24 field with 0 as 2-bit value.

Registers altered (except destination register)

LR

Occurrence Exceptions

none

Detected Exceptions

none

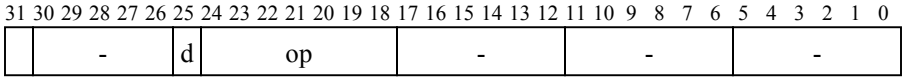
1.5.8. Return from Trap

Ope-code	op	Operation
RETT	0000101	Return from Trap

Category

Privileged Control

Instruction Format (Return from Trap instruction (R-R))



Assembler Syntax

RETT #d

Description

The RETT instruction is used for returning from a trap handler.
When an exception does not occur and the d field is 0 in the normal running mode, the RETT instruction

1) executes unconditional control transfer to the target address in PCSR register.
2) restores the S field of PSR from the PS field of PSR.
3) sets the ET field of PSR to '1'

When the d field is 1, the RETT instruction

1) executes unconditional control transfer to the target address, which is calculated by lower 2 bits of BPCSR register defined to 0.
2) restores the S field of PSR from the BS field of BPSR.
3) restores the ET field of PSR from the BET field of BPSR.
4) Transfers the processor mode to normal running mode from debug running mode.

The following exception may occur when a RETT instruction executed
The privileged_instruction exception occurs when the trap is permitted (PSR.ET=1) and the processor executes in the user-mode (PSR.S=0).
It transits in the HALT mode when the trap is permitted (PSR.ET=1) and the processor executes in super-visor mode (PSR.S=1).
It transits in the HALT mode when the trap is inhibit (PSR.ET=0) and the processor executes in the usr-mode (PSR.S=0).

Registers altered (except destination register)

PSR

Occurrence Exceptions

privileged_instruction

Detected Exceptions

none

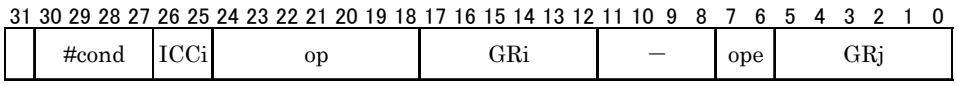
1.5.9. Integer Conditional Trap

Ope-code	op	Ope	Operation
Ticc	0000100	00	Integer Conditional Trap
TIicc	0011100	-	Integer Conditional Trap (Immediate)

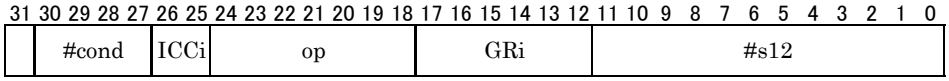
Category

Control

Instruction Format (Trap instruction (R-R))



Instruction Format (Trap instruction (R-simm))



Assembler Syntax

```
Ticc    ICCi, GRi, GRj
        TEQ     : Z==1
        TNE     : Z==0
        TLE     : (Z or (N xor V))==1
        TGT     : (Z or (N xor V))==0
        TLT     : (N xor V)==1
        TGE     : (N xor V)==0
        TLS     : (C or Z)==1
        THI     : (C or Z)==0
        TC      : C==1
        TNC     : C==0
        TN      : N==1
        TP      : N==0
        TV      : V==1
        TNV     : V==0
TNO
TRA     GRi, GRj
```

Tlicc ICCi, GRi, #s12
TIEQ : Z==1
TINE : Z==0
TILE : (Z or (N xor V))==1
TIGT : (Z or (N xor V))==0
TILT : (N xor V)==1
TIGE : (N xor V)==0
TILS : (C or Z)==1
TIHI : (C or Z)==0
TIC : C==1
TINC : C==0
TIN : N==1
TIP : N==0
TIV : V==1
TINV : V==0
TINO
TIRA GRi, #s12

Description

The integer conditional trap instruction evaluates the #cond field of ICCi. When the evaluation is true and the higher priority trap or interrupt requests are not deferred, the trap_instruction is generated. When the evaluation is false, it behaves like a NOP operation.
When the trap_instruction is generated, the tt field of the trap base register (TBR) is written with 128 plus the least significant seven bits of "GRi+GRj" (immediate instruction: "GRi + sign_ext (#s12)") (sign_ext (X): X extended with a sign).
If the break interrupt or the program interrupt occurs in the previous instruction in a same VLIW instruction, this instruction is not executed and software interrupt is not initiated. The multiple trap instructions are not put in a same VLIW instruction.
The test condition of ICC shown by the content of the #cond field and ICCi is as follows.

Registers altered (except destination register)

none

Occurrence Exceptions

trap_instruction

Detected Exceptions

none

Table 8 Pseudo opecode

Pseudo opecode	#cond	Operation	ICC test
TEQ	0100	Trap Equal	Z
TNE	1100	Trap Not Equal	Not Z
TLE	0111	Trap Less or Equal	Z or (N xor V)
TGT	1111	Trap Greater	Not (Z or (N xor V))
TLT	0011	Trap Less	N xor V
TGE	1011	Trap Greater or Equal	Not (N xor V)
TLS	0101	Trap Less or Equal Unsigned	C or Z
THI	1101	Trap Greater Unsigned	Not (C or Z)
TC	0001	Trap Carry Set	C
TNC	1001	Trap Carry Clear	Not C
TN	0110	Trap Negative	N
TP	1110	Trap Positive	Not N
TV	0010	Trap Overflow Set	V
TNV	1010	Trap Overflow Clear	Not V
TNO	0000	Trap Never	0
TRA	1000	Trap Always	1
TIEQ	0100	Trap Immediate Equal	Z
TINE	1100	Trap Immediate Not Equal	Not Z
TILE	0111	Trap Immediate Less or Equal	Z or (N xor V)
TIGT	1111	Trap Immediate Greater	Not (Z or (N xor V))
TILT	0011	Trap Immediate Less	N xor V
TIGE	1011	Trap Immediate Greater or Equal	Not (N xor V)
TILS	0101	Trap Immediate Less or Equal Unsigned	C or Z
TIHI	1101	Trap Immediate Greater Unsigned	Not (C or Z)
TIC	0001	Trap Immediate Carry Set	C
TINC	1001	Trap Immediate Carry Clear	Not C
TIN	0110	Trap Immediate Negative	N
TIP	1110	Trap Immediate Positive	Not N
TIV	0010	Trap Immediate Overflow Set	V
TINV	1010	Trap Immediate Overflow Clear	Not V
TINO	0000	Trap Immediate Never	0
TIRA	1000	Trap Immediate Always	1

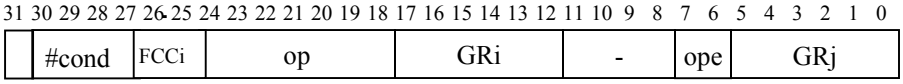
1.5.10. Floating-point / media Conditional Trap

Ope-code	op	ope	Operation
FTfcc	0000100	01	Floating-point/Media Conditional Trap
FTIfcc	0011101	-	Floating-point/Media Conditional Trap (Immediate)

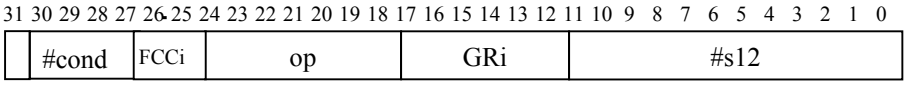
Category

Control

Instruction Format (Conditional instruction (R-R))



Instruction Format (Conditional instruction (R-simm))



Assembler Syntax

```
FTfcc    FCCi, GRi, GRj
     FTNE      : (L or G or U)==1
     FTEQ      : E ==1
     FTLG      : (L or G)==1
     FTUE      : (E or U) ==1
     FTUL      : (L or U) ==1
     FTGE      : (E or G) ==1
     FTLT      : L==1
     FTUGE     : (E or G or U) ==1
     FTUG      : (G or U)==1
     FTLE      : (E or L) ==1
     FTGT      : G==1
     FTULE     : (E or L or U) ==1
     FTU       : U==1
     FTO       : (E or L or G) ==1
FTNO
FTRA     GRi, GRj
```

```
FTIfcc   FCCi, GRi, #s12
      FTINE      : (L or G or U)==1
      FTIEQ      : E==1
      FTILG      : (L or G)==1
      FTIUE      : (E or U) ==1
      FTIUL      : (L or U) ==1
      FTIGE      : (E or G) ==1
      FTILT      : L==1
      FTIUGE     : (E or G or U) ==1
      FTIUG      : (G or U)==1
      FTILE      : (E or L) ==1
      FTIGT      : G==1
      FTIULE     : (E or L or U) ==1
      FTIU       : U==1
      FTIO       : (E or L or G) ==1
   FTINO
   FTIRA   GRi, #s12
```

Description

The floating-point / media conditional trap instruction evaluates the #cond field of FCCi. When the evaluation is true and the higher priority trap or interrupt requests are not deferred, the trap_instruction is generated. When the evaluation is false, it behaves like a NOP operation. When the trap_instruction is generated, the tt field of the trap base register (TBR) is written with 128 plus the least significant seven bits of "GRi+GRj" (immediate instruction: "GRi + sign_ext (#s12)") (sign_ext (X): X extended with a sign).

If the break interrupt or the program interrupt occurs in the previous instruction in a same VLIW instruction, this instruction is not executed and software interrupt is not initiated. The multiple trap instructions are not put in a same VLIW instruction.

The test condition of FCC shown by the content of the #cond field and the FCC is as follows.

Registers altered (except destination register)

none

Occurrence Exceptions

fp_disabled
trap_instruction

Detected Exceptions

none

Table 9 Pseudo opecode

Pseudo opecode	#cond	Operation	FCC test
FTNE	0111	Trap Not Equal	L or G or U
FTEQ	1000	Trap Equal	E
FTLG	0110	Trap Less or Greater	L or G
FTUE	1001	Trap Unordered or Equal	E or U
FTUL	0101	Trap Unordered or Less	L or U
FTGE	1010	Trap Greater or Equal	E or G
FTLT	0100	Trap Less	L
FTUGE	1011	Trap Unordered or Greater or Equal	E or G or U
FTUG	0011	Trap Unordered or Greater	G or U
FTLE	1100	Trap Less or Equal	E or L
FTGT	0010	Trap Greater	G
FTULE	1101	Trap Unordered or Less or Equal	E or L or U
FTU	0001	Trap Unordered	U
FTO	1110	Trap Ordered	E or L or G
FTNO	0000	Trap Never	0
FTRA	1111	Trap Always	1
FTINE	0111	Trap Immediate Not Equal	L or G or U
FTIEQ	1000	Trap Immediate Equal	E
FTILG	0110	Trap Immediate Less or Greater	L or G
FTIUE	1001	Trap Immediate Unordered or Equal	E or U
FTIUL	0101	Trap Immediate Unordered or Less	L or U
FTIGE	1010	Trap Immediate Greater or Equal	E or G
FTILT	0100	Trap Immediate Less	L
FTIUGE	1011	Trap Immediate Unordered or Greater or Equal	E or G or U
FTIUG	0011	Trap Immediate Unordered or Greater	G or U
FTILE	1100	Trap Immediate Less or Equal	E or L
FTIGT	0010	Trap Immediate Greater	G
FTIULE	1101	Trap Immediate Unordered or Less or Equal	E or L or U
FTIU	0001	Trap Immediate Unordered	U
FTIO	1110	Trap Immediate Ordered	E or L or G
FTINO	0000	Trap Immediate Never	0
FTIRA	1111	Trap Immediate Always	1

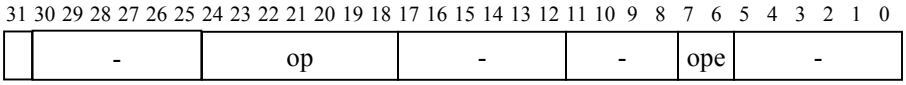
1.5.11. Break

Ope-code	op	ope	Operation
Break	0000100	11	Break Trap

Category

Control

Instruction Format (Trap instruction (R-R))



Assembler Syntax

BREAK

Description

The break interrupt occurs in the break instruction.
When the break interrupt is detected, the tt field of the trap base register (TBR) is set to 255.
.

Registers altered (except destination register)

none

Occurrence Exceptions

none

Detected Exceptions

none

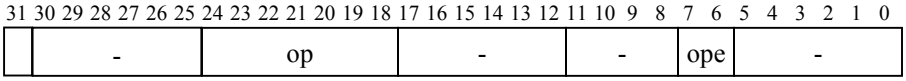
1.5.12. Media Trap

Ope-code	op	ope	Operation
Mtrap	0000100	10	Media Trap

Category

Control

Instruction Format (Trap instruction (R-R))



Assembler Syntax

MTRAP

Description

The media trap instruction tests the msr.mtt and the msr.ov, and finds out whether the msr.mtt and the msr.ov are true or false.
When the result is true, i.e. the exception factors are held by the msr, the media_exception is generated. When the result is false, the media_exception is not generated and it behaves like a nop operation.

Registers altered (except destination register)

none

Occurrence Exceptions

mp_disabled
mp_exception

Detected Exceptions

none

1.6. Constant Setting Instructions

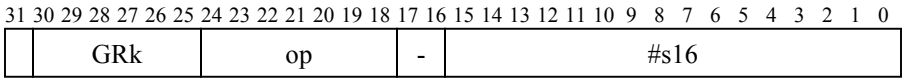
1.6.1. Set

Ope-code	op	Operation
SETLO	0111101	Set Low-Order unsigned 16 bits
SETHI	0111110	Set High-Order 16 bits
SETLOS	0111111	Set Low-Order Signed 16 bits

Category

Integer

Instruction Format (INT,Logic,Shift Operation (R-R))



Assembler Syntax

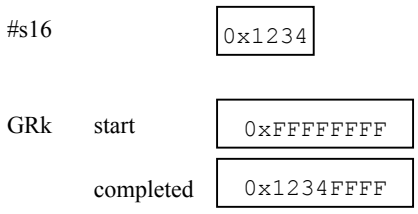
SETHI #i16, GRk
SETLO #u16, GRk
SETLOS #s16, GRk

Description

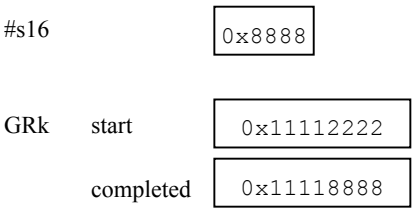
The SETHI instruction replaces the high-order 16 bits of the GRk with #s16.
The SETLO instruction replaces the low-order 16 bits of the GRk with #s16.
The SETLOS instruction sets sign extended #s16 data into the GRk.

Operation example

SETHI #0x1234,GRk



SETLO #0x8888,GRk



SETLOS #0x89AB,GRk

#s16		0x89AB
GRk	start	0x11112222
	completed	0xFFFF89AB

Registers altered (except destination register)

none

Occurrence Exceptions

register_exception (unimplement_exception)

Detected Exceptions

none

1.7. Scan instruction

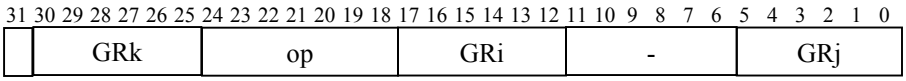
1.7.1. Scan

Ope-code	Op	Operation
SCAN	0001011	SCAN
SCANI	1000111	SCAN (Immediate)

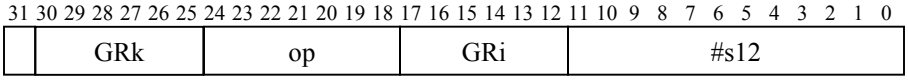
Category

Integer

Instruction Format (INT, Logic, Shift Operation (R-R))



Instruction Format (INT, Logic, Shift Operation (R-simm))



Assembler Syntax

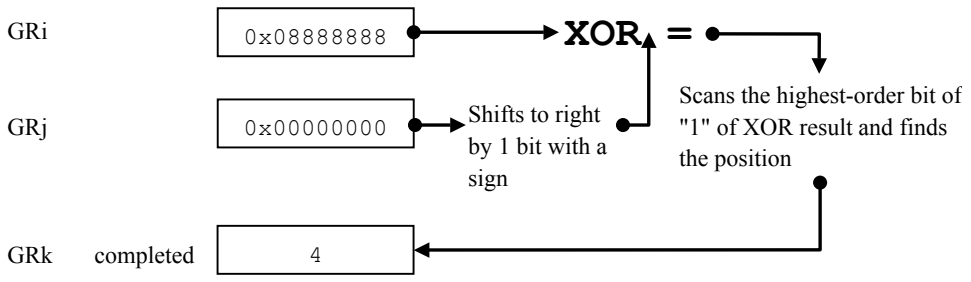
SCAN GRi, GRj, GRk
SCANI GRi, #s12, GRk

Description

The SCAN instruction shifts the value of GRj (immediate instruction : sign_ext (#s12)) to the right by 1 bit and extends with a sign, and executes XOR calculating on the value of GRi and the shifted-extended value of GRj, and writes the position of the highest-order bit of "1" in GRk as a result. So when the MSB of the result of XOR calculating is "1", the result is 0. When the LSB is "1" and the others bits are "0", the result is 31. When all of the bits as the result of XOR calculating are "0", the result of SCAN instruction is 0.

Operation example

SCAN GRi,GRj,GRk



Registers altered (except destination register)

none

Occurrence Exceptions

register_exception (unimplement_exception)

Detected Exceptions

none

1.8. Condition Code Operating Instructions

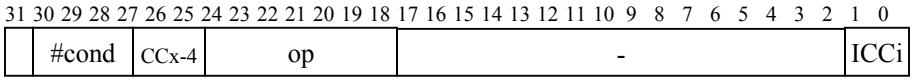
1.8.1. Check for Integer Condition Code

Ope-code	op	Operation
CKicc	0001000	Check for Integer Condition code

Category

Branch

Instruction Format (Checking Instruction)



Assembler Syntax

CKicc ICCi, CCx (x = 4-7)
- CKEQ : Z==1
- CKNE : Z==0
- CKLE : (Z or (N xor V))==1
- CKGT : (Z or (N xor V))==0
- CKLT : (N xor V)==1
- CKGE : (N xor V)==0
- CKLS : (C or Z)==1
- CKHI : (C or Z)==0
- CKC : C==1
- CKNC : C==0
- CKN : N==1
- CKP : N==0
- CKV : V==1
- CKNV : V==0

CKNO CCx (x = 4-7)
CKRA CCx (x = 4-7)

Description

The CKicc instruction tests ICCi according to the contents of #cond field, shown by Table 10 The result of test is written into the condition code register for conditional instruction CCx, shown by Table 11.

Table 10 #cond field and Pseudo opecode

Pseudo opecode	#cond	Operation	ICC test
CKEQ	0100	Check Equal	Z
CKNE	1100	Check Not Equal	Not Z
CKLE	0111	Check Less or Equal	Z or (N xor V)
CKGT	1111	Check Greater	Not (Z or (N xor V))
CKLT	0011	Check Less	N xor V
CKGE	1011	Check Greater or Equal	Not (N xor V)
CKLS	0101	Check Less or Equal Unsigned	C or Z
CKHI	1101	Check Greater Unsigned	Not (C or Z)
CKC	0001	Check Carry Set	C
CKNC	1001	Check Carry Clear	Not C
CKN	0110	Check Negative	N
CKP	1110	Check Positive	Not N
CKV	0010	Check Overflow Set	V
CKNV	1010	Check Overflow Clear	Not V
CKNO	0000	Check Never	0
CKRA	1000	Check Always	1

Table 11 Result of ICC test and Value of CCCR(CCx)

Result of test	value of cccr
false	10
true	11

Registers altered (except destination register)

none

Occurrence Exceptions

none

Detected Exceptions

none

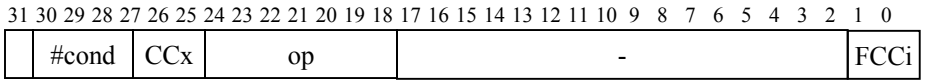
1.8.2. Check for Floating-point/Media Condition Code

Ope-code	op	Operation
FCKfcc	0001001	Check for Floating-point/Media Conditional code

Category

Branch

Instruction Format (Checking Instruction)



Assembler Syntax

FCKfcc FCCi, CCx (x = 0-3)
- FCKNE : (L or G or U) ==1
- FCKEQ : E==1
- FCKLG : (L or G)==1
- FCKUE : (E or U)==1
- FCKUL : (L or U)==1
- FCKGE : (E or G)==1
- FCKLT : L==1
- FCKUGE : (E or G or U) ==1
- FCKUG : (G or U)==1
- FCKLE : (E or L)==1
- FCKGT : G==1
- FCKULE : (E or L or U) ==1
- FCKU : U==1
- FCKO : (E or L or G) ==1

FCKNO CCx (x = 0-3)
FCKRA CCx (x = 0-3)

Description

Check for floating-point/media condition code instruction tests FCCi according to contents of #cond field shown by Table 12 . The result of test is written into the condition code register for conditional instruction CCx shown by Table 13.

Table 12 #cond field and Pseudo opecode

Pseudo opecode	#cond	Operation	FCC test
FCKNE	0111	Check Not Equal	L or G or U
FCKEQ	1000	Check Equal	E
FCKLG	0110	Check Less or Greater	L or G
FCKUE	1001	Check Unordered or Equal	E or U
FCKUL	0101	Check Unordered or Less	L or U
FCKGE	1010	Check Greater or Equal	E or G
FCKLT	0100	Check Less	L
FCKUGE	1011	Check Unordered or Greater or Equal	E or G or U
FCKUG	0011	Check Unordered or Greater	G or U
FCKLE	1100	Check Less or Equal	E or L
FCKGT	0010	Check Greater	G
FCKULE	1101	Check Unordered or Less or Equal	E or L or U
FCKU	0001	Check Unordered	U
FCKO	1110	Check Ordered	E or L or G
FCKNO	0000	Check Never	0
FCKRA	1111	Check Always	1

Table 13 Result of FCC test and value of CCCR(CCx)

Result of test	value of cccr
false	10
true	11

Registers altered (except destination register)

none

Occurrence Exceptions

fp_disabled

Detected Exceptions

none

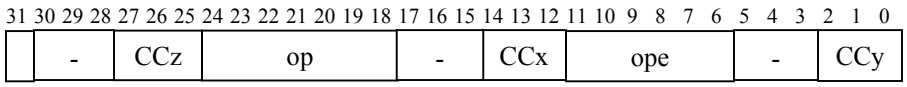
1.8.3. Condition Code Logical Operations

Ope-code	op	ope	Operation
ANDCR	0001010	001000	AND CR
ORCR	0001010	001001	OR CR
XORCR	0001010	001010	XOR CR
NOTCR	0001010	001011	NOT CR
NANDCR	0001010	001100	NAND CR
NORCR	0001010	001101	NOR CR
ANDNCR	0001010	010000	NOT-AND CR
ORNCR	0001010	010001	NOT-OR CR
NANDNCR	0001010	010100	NOT-NAND CR
NORNCR	0001010	010101	NOT-NOR CR

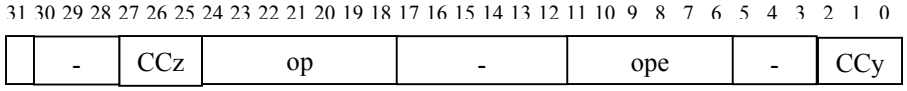
Category

Branch

Instruction Format (Condition code logic instruction)



Instruction Format (Condition code not instruction)



Assembler Syntax

ANDCR	CCx, CCy, CCz
ORCR	CCx, CCy, CCz
XORCR	CCx, CCy, CCz
NOTCR	CCy, CCz
NANDCR	CCx, CCy, CCz
NORCR	CCx, CCy, CCz
ANDNCR	CCx, CCy, CCz
ORNCR	CCx, CCy, CCz
NANDNCR	CCx, CCy, CCz
NORNCR	CCx, CCy, CCz

Description

Condition code logic instruction execute following logical operation between condition code for conditional instruction register specified by CCx and condition code for conditional instruction register specified by CCy. Condition code logic instruction deals with three kinds of values: true, false and undefined.

ANDCR is specified by Table 15.
ORCR is specified by Table 16
XORCR is specified by Table 17
NANDCR is specified by Table 18
NORCR is specified by Table 19.
ANDNCR is specified by Table 20
ORNCR is specified by Table 21
NANDNCR is specified by Table 22
NORNCR is specified by Table 23

The result of operation is written into condition code field specified by CCz.

NOTCR instruction is specified by Table 24 and executes not-operation. The result of operation is written into condition field specified by CCz.

Pay attention that the result will change when CCx is exchanged for CCy in ANDCR, NANDCR, ANDNCR, ORNCR, NANDNCR and NORNCR.

(Table 14 shows relation between values of CCCR and "true/false/undefined" in Table 15-Table 24.)

Table 14 Values of CCCR(CCx, CCy and CCz)

	mean
00	Undefined
01	Undefined
10	False
11	True

Table 15 ANDCR

CCx \ CCy	*true*	*false*	*undefined*
true	true	false	undefined
false	undefined	undefined	undefined
undefined	undefined	undefined	undefined

Table 16 ORCR

CCx \ CCy	*true*	*false*	*undefined*
true	true	true	true
false	true	false	false
undefined	true	false	undefined

Table 17 XORCR

CCx \ CCy	*true*	*false*	*undefined*
true	false	true	true
false	true	false	undefined
undefined	undefined	undefined	undefined

Table 18 NANDCR

CCx \ CCy	*true*	*false*	*undefined*
true	false	true	undefined
false	undefined	undefined	undefined
undefined	undefined	undefined	undefined

Table 19 NORCR

CCx \ CCy	*true*	*false*	*undefined*
true	false	false	false
false	false	true	true
undefined	false	true	undefined

Table 20 ANDNCR

CCx \ CCy	*true*	*false*	*undefined*
true	undefined	undefined	undefined
false	true	false	undefined
undefined	undefined	undefined	undefined

Table 21 ORNCR

CCx \ CCy	*true*	*false*	*undefined*
true	true	false	false
false	true	true	true
undefined	true	false	undefined

Table 22 NANDNCR

CCx \ CCy	*true*	*false*	*undefined*
true	undefined	undefined	undefined
false	false	true	undefined
undefined	undefined	undefined	undefined

Table 23 NORNCR

CCx \ CCy	*true*	*false*	*undefined*
true	false	true	true
false	false	false	false
undefined	false	true	undefined

Table 24 NOTCR

CCy	
true	false
false	true
undefined	undefined

Registers altered (except destination register)

CCCR

Occurrence Exceptions

none

Detected Exceptions

none

1.9. Special Operation Instructions

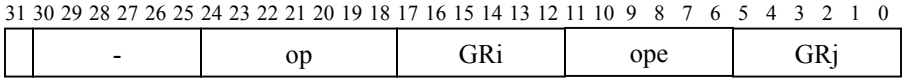
1.9.1. Instruction Cache Invalidate

Ope-code	op	ope	Operation
ICI	0000011	111000	Instruction Cache Invalidate

Category

Control

Instruction Format (Cache invalidate instruction (R-R))



Assembler Syntax

ICI @(GRi, GRj)

Description

Instruction Cache Invalidate instruction calculates "GRi+GRj" as an effective address and invalidates a block of instruction cache, which contains byte data specified by the effective address, if the block is in the instruction cache.
This instruction invalidates the instruction cache regardless of the cache enable bit (HSR0.ICE).

Registers altered (except destination register)

none

Occurrence Exceptions

instruction_access_error
register_exception (unimplement_exception)

Detected Exceptions

none

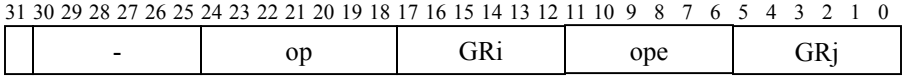
1.9.2. Data Cache Invalidate

Ope-code	op	ope	Operation
DCI	0000011	111100	Data Cache Invalidate

Category

Control

Instruction Format (Cache invalidate instruction (R-R))



Assembler Syntax

DCI @(GRi, GRj)

Description

Data Cache Invalidate instruction calculate “GRi+GRj” as an effective address and invalidates a block of data cache, which contains byte data specified by the effective address, if the block is in the data cache.
The block is invalidated even if the block in the data cache is the latest one.
This instruction invalidates the data cache regardless of the write-back/write-through mode (HSR0.CBM) and the state of the cache enable bit (HSR0.DCE).

Registers altered (except destination register)

none

Occurrence Exceptions

data_access_error
register_exception (unimplement_exception)

Detected Exceptions

none

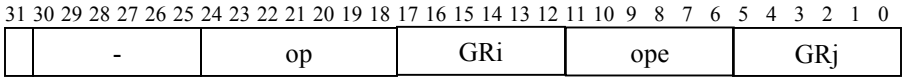
1.9.3. Data Cache Flush

Ope-code	op	ope	Operation
DCF	0000011	111101	Data Cache flush

Category

Control

Instruction Format (Cache invalidate instruction (R-R))



Assembler Syntax

DCF @(GRi, GRj)

Description

Data Cache Flash instruction calculate "GRi+GRj" as an effective address and invalidates a block of data cache, which contains byte data specified by the effective address, if the block is in the data cache. When the block in the data cache is the latest data, the data is copied from data cache to memory.
This instruction invalidates the data cache regardless of the write-back/write-through mode (HSR0.CBM) and the state of the cache enable bit (HSR0.DCE).

Registers altered (except destination register)

none

Occurrence Exceptions

data_access_error
register_exception (unimplement_exception)

Detected Exceptions

none

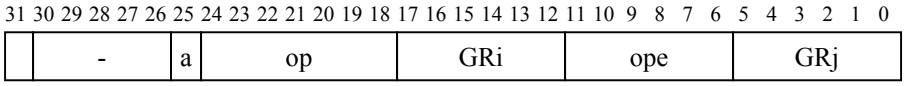
1.9.4. Instruction Cache Entry Invalidate Instruction

Ope code	op	ope	Operation
ICEI	0000011	111001	Instruction Cache Entry Invalidate

Category

Control

Instruction format (Instruction cache invalidate instruction (R – R))



Assembler description

ICEI @(GRi,GRj),a

Description

When a block containing byte data pointed to by the entry calculated using “GRi+GRj” exists in the instruction cache, the instruction cache invalidate instruction invalidates that block.
When bit “a”is 1, all entries in the instruction cache are invalidated.
This instruction invalidates the cache independently of the cache validate bit (HSR0.ICE).
This instruction is optional so it is not necessarily implemented.

Registers altered (except destination register)

None

Occurrence Exceptions

instruction_access_exception
instruction_access_MMU_miss
instruction_access_error
register_exception (unimplement_exception)

Detected Exceptions

None

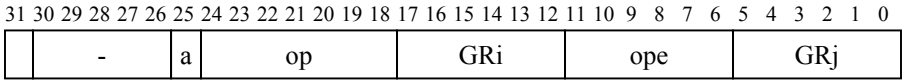
1.9.5. Data Cache Entry Invalidate Instruction

Ope code	op	ope	Operation
DCEI	0000011	111010	Data Cache Entry Invalidate

Category

Control

Instruction format (Data cache invalidate instruction (R – R))



Assembler description

DCEI @(GRi,GRj),a

Description

When a block containing byte data pointed to by the entry calculated using “GRi+GRj” exists in the data cache, the data cache invalidate instruction invalidates that block. When that block is the latest data, that block is discarded (but not flushed).
When bit “a” is 1, all entries in the data cache are invalidated.
This instruction invalidates the cache independently of the write back mode/write through mode (HSR0.CBM) and the cache validate bit (HSR0.DCE).
This instruction is optional so it is not necessarily implemented.

Registers altered (except destination register)

None

Occurrence Exceptions

data_access_exception
data_access_MMU_miss
data_access_error
register_exception (unimplement_exception)

Detected Exceptions

None

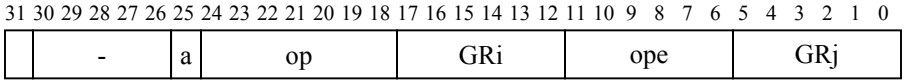
1.9.6. Data Cache Entry Flush Instruction

Ope code	op	ope	Operation
DCEF	0000011	111011	Data Cache Entry Flush

Category

Cotrol

Instruction format (Data cache invalidate instruction (R – R))



Assembler description

DCEF @(GRi,GRj),a

Description

When a block containing byte data pointed to by the entry calculated using “GRi+GRj” exists in the data cache, the data cache entry flush instruction invalidates that block. When that block is the latest data, that block is recorded in main memory.
When bit “a” is 1, all entries in the data cache are invalidated.
This instruction invalidates the cache independently of the write back mode/write through mode (HSR0.CBM) and the cache validate bit (HSR0.DCE).
This instruction is optional so it is not necessarily implemented.

Registers altered (except destination register)

None

Occurrence Exceptions

data_access_exception
data_access_MMU_miss
data_access_error
register_exception (unimplement_exception)

Detected Exceptions

None

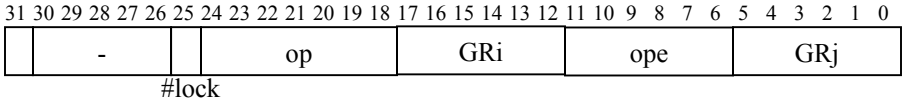
1.9.7. Instruction Cache Pre-Load

Ope-code	op	ope	Operation
ICPL	0000011	110000	Instruction Cache Pre-Load

Category

Control

Instruction Format (Instruction Cache Pre-Load Instruction (R-R))



Assembler Syntax

ICPL GRi, GRj, #lock

Description

The Instruction Cache Pre-Load Instruction pre-loads the data of the number of bytes which GRj shows from the address area specified by GRi to the instruction cache. When GRj is 0, the instruction pre-loads one block size of data which contains effective address shown by GRi.
If lock bit is 1 and instruction cache has lock mechanism, the instruction locks the pre-loaded data.
When the instruction cache for storage is already in the state of the lock, the instruction operates as NOP.
If instruction cache is disabled(HSR0.ICE=0), the instruction operates as NOP.
When the instruction_access_exception, instruction_access_MMU_miss, instruction_access_error is detected, the instruction operates as NOP and exception does not occur.

Programming Notes

In using this instruction, processor can lock instruction cache in every block. But lock mechanism of the instruction cache in an actual implementation often becomes the unit of way. Therefore it is effective to pre-load and to lock the capacity corresponding of each way of the instruction cache.

Registers altered (except destination register)

none

Occurrence Exceptions

register_exception

Detected Exceptions

none

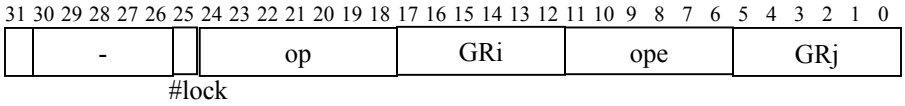
1.9.8. Data Cache Pre-Load

Ope-code	op	ope	Operation
DCPL	0000011	110100	Data Cache Pre-Load

Category

Control

Instruction Format (Data Cache Pre-Load Instruction(R-R))



Assembler Syntax

DCPL GRi, GRj, #lock

Description

The Data Cache Pre-Load Instruction pre-loads the data of the number of bytes which GRj shows from the address area specified by GRi to the data cache. When GRj is 0, the instruction pre-loads one block size of data which contains effective address shown by GRi.
If lock bit is 1 and data cache has lock mechanism, the instruction locks the pre-loaded data.
When the data cache for storage is already in the state of the lock, the instruction operates as NOP.
If data cache is disabled (HSR0.DCE=0), the instruction operates as NOP.
When the data_access_exception, data_access_MMU_miss, data_access_error is detected, the instruction operates as NOP and exception does not occur.

Programming Notes

In using this instruction, processor can lock instruction cache every block. But lock mechanism of the data cache in an actual implementation often becomes the unit of way. Therefore it is effective to pre-load and to lock the capacity corresponding of each way of the data cache.

Registers altered (except destination register)

none

Occurrence Exceptions

register_exception

Detected Exceptions

none

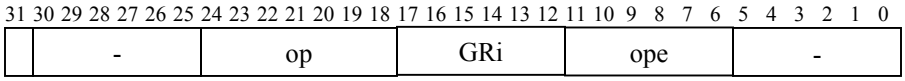
1.9.9. Instruction Cache UnLock

Ope-code	op	ope	Operation
ICUL	0000011	110001	Instruction Cache UnLock

Category

Control

Instruction Format (Instruction Cache UnLock Instruction (R-R))



Assembler Syntax

ICUL GRi

Description

The Instruction Cache UnLock Instruction unlocks instruction cache, which contain byte data specified by the effective address in GRi.
The unit of unlocking (All or way or block) depends on implementation.
If instruction cache is disabled (HSR0.ICE=0), the instruction operates as NOP.
When the instruction_access_exception, instruction_access_MMU_miss, instruction_access_error is detected, the instruction operates as NOP and exception does not occur.

Registers altered (except destination register)

none

Occurrence Exceptions

register_exception

Detected Exceptions

none

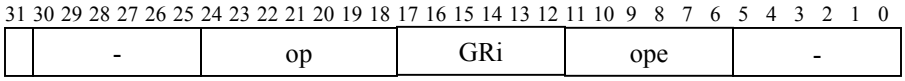
1.9.10. Data Cache UnLock

Ope-code	op	ope	Operation
DCUL	0000011	110101	Data Cache UnLock

Category

Control

Instruction Format (Data Cache UnLock Instruction (R-R))



Assembler Syntax

DCUL GRi

Description

The Data Cache UnLock Instruction unlocks data cache, which contain byte data specified by the effective address in GRi.
The unit of unlocking (All or way or block) depends on implementation.
If data cache is disabled (HSR0.DCE=0), the instruction operates as NOP.
When the data_access_exception, data_access_MMU_miss, data_access_error, is detected, the instruction operates as NOP and exception does not occur.

Registers altered (except destination register)

none

Occurrence Exceptions

register_exception

Detected Exceptions

none

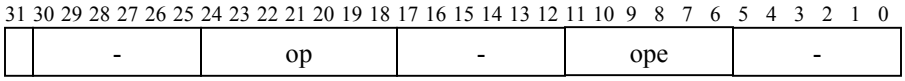
1.9.11. Barrier

Ope-code	op	ope	Operation
BAR	0000011	111110	Barrier

Category

Control

Instruction Format (Barrier Instruction (R-R))



Assembler Syntax

BAR

Description

The Barrier Instruction waits for the completion of the integer instructions, load/store instructions, floating-point instructions, and media instructions executed by processor before this instruction.
If co-processor is synchronized mode (PSR.ECS=1), the instruction waits for the completion of the instruction executed by co-processor, too.

Registers altered (except destination register)

none

Occurrence Exceptions

none

Detected Exceptions

none

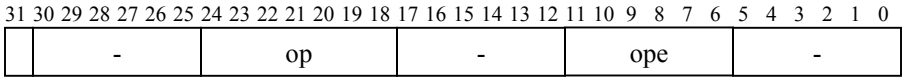
1.9.12. Memory Barrier

Ope-code	op	ope	Operation
MEMBAR	0000011	111111	Memory Barrier

Category

Control

Instruction Format (Memory Barrier Instruction (R-R))



Assembler Syntax

MEMBAR

Description

The Memory Barrier Instruction waits for the completion of the load/store instructions, issued load/store instructions executed by processor before this instruction.

Registers altered (except destination register)

none

Occurrence Exceptions

none

Detected Exceptions

none

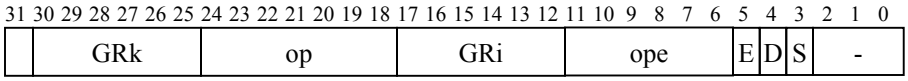
1.9.13. Load Real Address of Instruction (This instruction is available for MB93451.)

Ope-code	op	ope	Operation
LRAI	0000011	100000	Load Real Address of Instruction

Category

Control

Instruction Format (Load Real Address of Instruction (R-R))



Assembler Syntax

LRAI GRi, GRk, #E, #D, #S

Description

LRAI instruction takes GRi as "VA" for searching IAMR/TLB and Implicit SAT entries to generate corresponding "RA" which is then returned in GRk.
DAT mode is applied for the searching.

If an entry is found in IAMR/TLB or in Implicit SAT entries and corresponding "RA" generation is successful, high order 18bits of "RA" and 14bits status code are stored in GRk.

If IAMR/TLB and Implicit SAT entries are missed in search, "TLB miss exception" is reported when #D=0 or "MMU miss exception" is reported when #D=1 in STW(Software Table Walk) mode. Address translation and re-searching is performed by hardware in HTW(Hardware Table Walk) mode.
If #E is set (to 1), above actions (exception report or hardware translation) are suppressed and instead all-0 in high order 18bits and 14bits status code are stored in GRk.

#D/#S can be set (to 1) to suppress DAT/SAT entry search, respectively. If #D or #S is set then corresponding report of "MMU multiple_hit exception" is also suppressed. If both #D and #S are set then 0x00000000 is stored in GRk and there's no other side effect.

In the below, the detail of the output format in GRk is explained:

High order 18bits of GRk is set by high order 18bits of "RA" if its successfully generated, or is set by all-0 if failed
Low order 14bits of GRk is not a part of "RA" but is the code in following format :

bit10 : "1" indicates there was a hit in IAMR
"0" indicates there was no hit in IAMR
bit9-4 : If IAMR is hit, the IAMR number is set (0-63)
If IAMR is missed, following code is set :
0b000000 : all factors below was not detected
0b000001 : TLB is hit
0b000010 : Implicit SAT entry for FExxxxxx is hit

0b000011 : Implicit SAT entry for FFxxxxxx is hit
0b001110 : the cause of "MMU multiple-hit exception" is detected (but not reported)
*only possible when #E is 1
0b001111 : the cause of "TLB miss exception"(#D=0) or "MMU miss exception"(#D=1) in STW mode or Address translation in HTW mode are detected (but not reported/translated)
*only possible when #E is 1

bit3 : If "RA" is successfully generated :
"1" indicates the area is supervisor protected
"0" indicates the area is not supervisor protected
"0" is set if "RA" is not generated

bit2 : If "RA" is successfully generated :
"1" indicates the area is not cacheable
"0" indicates the area is cacheable
"0" is set if "RA" is not generated

bit0 : "1" indicates "RA" is successfully generated
"0" indicates "RA" is not successfully generated

else : Values to be set are "M-D"

HSR0.EDAT is ignored during the execution of this instruction.
GRk is not modified if the exception is reported.

If this instruction is executed in MMU off mode, #E is assumed to be 1 regardless of the setting. If this instruction is executed in Compatible SAT mode, 0x00000000 is stored in GRk regardless of #E/#D/#S, and has no side effect.
It will be "U-P" if state has changed from compatible SAT mode into MMU off mode without TLB/IAMR initialization.

Registers altered (except destination register)

Occurrence Exceptions

register_exception
privileged_exception
MMU_multiplehit_exception
TLB_miss_exception (STW mode)
MMU_miss_exception (STW mode)
DAT_translation_exception (HTW mode)

Detected Exceptions

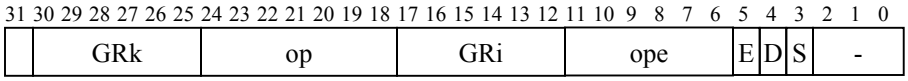
1.9.14. Load Real Address of Data (This instruction is available for MB93451.)

Ope-code	Op	ope	Operation
LRAD	0000011	100001	Load Real Address of Data

Category

Control

Instruction Format (Load Real Address of Instruction (R-R))



Assembler Syntax

LRAD GRi, GRk, #E, #D, #S

Description

LRAD instruction takes GRj as "VA" for searching DAMR/TLB and Implicit SAT entries to generate corresponding "RA" which is then returned in GRk.
DAT mode is applied for the searching.

If an entry is found in DAMR/TLB or in Implicit SAT entries and corresponding "RA" is successfully generated, high order 18bits of "RA" and 14bits status code are stored in GRk.

If DAMR/TLB and Implicit SAT entries are missed in search, "TLB miss exception" is reported when #D=0 or "MMU miss exception" is reported when #D=1 in STW(Software Table Walk) mode. Address translation and re-searching is performed by hardware in HTW(Hardware Table Walk) mode.
If #E is set (to 1), above actions (exception report or hardware translation) are suppressed and instead all-0 in high order 18bits and 14bits status code are stored in GRk.

#D/#S can be set (to 1) to suppress DAT/SAT entry search, respectively. If #D or #S is set then corresponding report of "MMU multiple_hit exception" is also suppressed. If both #D and #S are set then 0x00000000 is stored in GRk and there's no other side effect.

In the below, the detail of the output format in GRk is explained:

High order 18bits of GRk is set by high order 18bits of "RA" if its successfully generated, or is set by all-0 if its failed
Low order 14bits of GRk is not a part of "RA" but is the code in following format :

bit10 : "1" indicates there was a hit in DAMR
"0" indicates there was no hit in DAMR
bit9-4 : If DAMR is hit, the DAMR number is set (0-63)
If DAMR is missed, following code is set :
0b000000 : all factors below was not detected
0b000001 : TLB is hit
0b000010 : Implicit SAT entry for FExxxxxx is hit

0b000011 : Implicit SAT entry for FFxxxxxx is hit
0b001110 : the cause of "MMU multiple-hit exception" is detected. (but not reported)
*only possible when #E is 1
0b001111 : the cause of "TLB miss exception"(#D=0) or "MMU miss exception"(#D=1) in STW mode or Address translation in HTW mode are detected (but not reported/translated)
*only possible when #E is 1

bit3 : If "RA" is successfully generated :
"1" indicates the area is supervisor protected
"0" indicates the area is not supervisor protected
"0" is set if "RA" is not generated

bit2 : If "RA" is successfully generated :
"1" indicates the area is not cacheable
"0" indicates the area is cacheable
"0" is set if "RA" is not generated

bit1 : If "RA" is successfully generated :
"1" indicates the area is write protected
"0" indicates the area is not write protected
"0" is set if "RA" is not generated

bit0 : "1" indicates "RA" is successfully generated
"0" indicates "RA" is not successfully generated

else : Values to be set is "M-D"

HSR0.EDAT is ignored during the execution of this instruction.
GRk is not modified if the exception is reported.

If this instruction is executed in MMU off mode, #E is assumed to be 1 regardless of the setting. If this instruction is executed in Compatible SAT mode, 0x00000000 is stored in GRk regardless of #E/#D/#S, and has no side effect.
It will be "U-P" if state has changed from compatible SAT mode into MMU off mode without TLB/DAMR initialization.

Registers altered (except destination register)

Occurrence Exceptions

register_exception
privileged_exception
MMU_multiplehit_exception
TLB_miss_exception (STW mode)
MMU_miss_exception (STW mode)
DAT_translation_exception (HTW mode)

Detected Exceptions

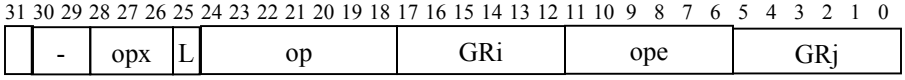
1.9.15. TLB Probe (This instruction is available for MB93451.)

Ope-code	Op	ope	Operation
TLBPR	0000011	100100	TLB Probe

Category

Control

Instruction Format (Load Real Address of Instruction (R-R))



Assembler Syntax

TLBPR GRi, GRj, #opx, #L

Description

TLB is operated depending on the value of #opx.

#opx=000:

TLB is searched by "VA" (GRi+GRj) and Context number (TPLR.CXN), and the matched entry's content is read out to TPR.
If there is no entry matched, 0 is set to TPPR.V and all other fields of TPPR/TPLR are unchanged.
Way number of read-out entry is set to TPXR.WAY.
If no read-out occurs then 1 is set to TPXR.E, otherwise 0 is set to TPXR.E.
* If multiple entries are hit on search, "MMU Multple-hit exception" is reported and TPXR.E is remained unchanged.
* NG(Non global-share)-bit is refered but D(DAT entry)-bit is not refered.
* If TLB is not mounted, it is treated as NOP

#opx=001 :

The entry, specified by "VA" (GRi+GRj) and Way number (TPXR.WAY), is read out from TLB to TPR.
0 is always set to TPXR.E.
* "VA" is used only to specify the line of TLB.
*If TLB is not mounted, it is treated as NOP.

#opx=010:

TLB is searched by "VA" (GRi+GRj) and Context number (TPLR.CXN), and the content of TPR is written into "selected-entry" in TLB.
If the search succeeded, the TPR is written into the hit-entry regardless of #L.
If the search missed, the TPR is written into the entry of the line specified by "VA" and the "hardware-selected" way. The selection of line and way are "M-D." The locked entry (L(Lock)-bit is set) may be selected if #L is 1 but no entry is selected if #L is 0.
If TPR is successfully written into TLB, old content of the TLB entry is written back to TPLR/TPPR and the way number of the entry is set to TPXR.WAY. TPXR.E is also set to 0.

If TPR is not written into TLB, due for all entries locked when #L is 0, TPR are not modified except TPXR.E which is set to 1.
* If multiple entries are hit on search, “MMU Multple-hit exception” is reported and no TPR(including TPXR.E) are modified.
* NG(Non Global-share)-bit is refered but D(DAT entry)-bit is not refered.
* “VA” (GRi+GRj) should be corresponding to TPLR.VPFN otherwise it is “U-P”
* If TLB is not mounted, it is treated as NOP

#opx=011:
The content of TPR is written into the entry specified by “VA” (GRi+GRj) and the Way number (TPXR.WAY), regardless of #L.
0 is always set to TPXR.E and no other TPR is modified.
* ”VA” is used only to specify the line of TLB
* ”VA” (GRi+GRj) should be corresponding to TPLR.VPFN otherwise is “U-P”
* If TLB is not mounted, it is treated as NOP

* Hereinafter “the deletion of entry” means to make V(Valid)-bit of corresponding entry to 0.
* It is “M-D” whether TPXR.E is modified for #opx=100-111.

#opx=100:
TLB is searched by “VA” (GRi+GRj) and Context number (TPLR.CXN), and all the hit-entries are deleted.
But the global entries (NG(Non-global)-bit is 0) are not deleted.
And if #L is 0, the locked entries (L(Lock)-bit set) are not deleted.
* D(DAT entry)-bit is not refered.
* “MMU Multple-hit exceptions” is not reported
* If TLB is not mounted, it is treated as NOP

#opx=101:
Regarding the line specified by “VA” (GRi+GRj) of TLB, all the entries matched with Context number (TPLR.CXN) are deleted.
All the entries matched with Context number (TPLR.CXN) of IAMR/DAMR are deleted at the same time.
But the global entries (NG(Non-global)-bit is 0) are not deleted.
And if #L is 0, the locked entries (L(Lock)-bit set) are not deleted.
* “MMU Multple-hit exceptions” is not reported
* If TLB is not mounted, it is treated as NOP

#opx=110:
Regarding the line specified by “VA” (GRi+GRj) of TLB, all the entries are deleted.
But if #L is 0, the locked entries (L(Lock)-bit is set) are not deleted.
* “MMU Multple-hit exceptions” is not reported
* If TLB is not mounted, it is treated as NOP

#opx=111:
All the entries of IAMR/DAMR are deleted.
But if #L is 0, the locked entries (L(Lock)-bit is set) are not deleted.
* “MMU Multple-hit exceptions” is not reported
* If TLB is not mounted, it is treated as NOP

The value of HSR0.EXMMU/EIMMU/EDMMU are ignored when this instruction is performed.

Registers altered (except destination register)

TPR

Occurrence Exceptions

register_exception
privileged_instruction_exception
MMU_multiplehit_exception

Detected Exceptions

1.10. Media Instructions

1.10.1. Media Nop Instruction (M -Type Instruction)

Ope-code	op	ope	Operation
MNOP	1111011	111011	Media No Operation

Category

Media

Instruction Format (Media instruction (R-R))

31	30 29 28 27 26 25	24 23 22 21 20 19 18	17	16 15 14 13 12	11 10 9 8 7 6	5 4 3 2 1 0
	111111	op	1	-	ope	-

Assembler Syntax

MNOP

Description

The MNOP instruction is a media type instruction. This instruction does nothing.

Registers altered (except destination register)

None

Occurrence Exceptions

mp_disabled

Detected Exceptions

None

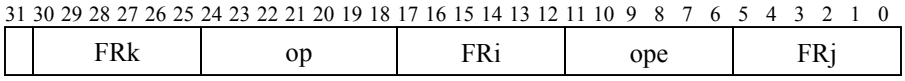
1.10.2. Logical Operations

Ope-code	op	ope	Operation
MAND	1111011	000000	Media And
MOR	1111011	000001	Media OR
MXOR	1111011	000010	Media XOR
MNOT	1111011	000011	Media Not

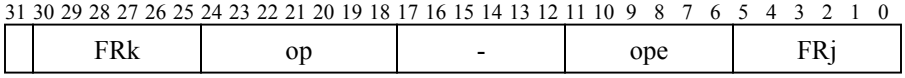
Category

Media
Single word

Instruction Format (Media instruction (R-R))



Instruction Format (Media NOT Operation (R-R))



Assembler Syntax

MAND	FRi, FRj, FRk
MOR	FRi, FRj, FRk
MXOR	FRi, FRj, FRk
MNOT	FRj, FRk

Description

The media logical instructions execute logical operations of FRi and FRj bit by bit and write the result in FRk.

Registers altered (except destination register)

none

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception)

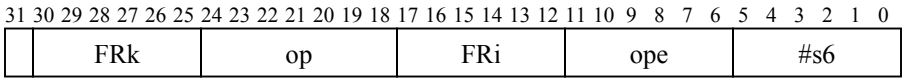
1.10.3. Rotate

Ope-code	op	ope	Operation
MROTLI	1111011	000100	Rotate Left Immediate
MROTRI	1111011	000101	Rotate Right Immediate

Category

Media
Single word

Instruction Format (Media instruction (R-imm6))



Assembler Syntax

MROTLI FRi, #s6, FRk
MROTRI FRi, #s6, FRk

Description

The media rotate (Word) instruction rotates FRi by the number specified by low-order five bits of #s6 and writes the results in the FRk.
The MROTLI instruction rotates FRi to the left.
The MROTRI instruction rotates FRi to the right.
The processing exception detected by the instruction is initiated by executing MTRAP instruction.

Registers altered (except destination register)

none

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception)

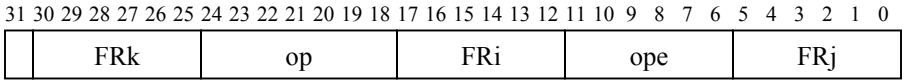
1.10.4. Word Cut

Ope-code	op	ope	Operation
MWCUT	1111011	000110	Word Data CUT from Doubleword Data
MWCUTI	1111011	000111	Word Data CUT from Doubleword Data (Immediate)

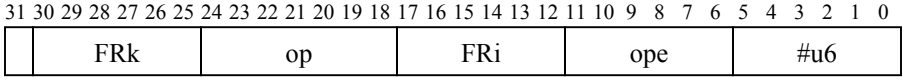
Category

Media
Single word

Instruction Format (Media instruction (R))



Instruction Format (Media instruction (imm))



Assembler Syntax

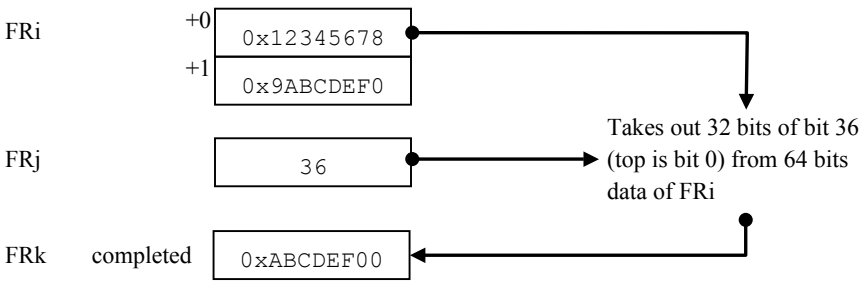
MWCUT FRi, FRj, FRk
MWCUTI FRi, #u6, FRk

Description

The MWCUT instruction concatenates FRi and FRi+1, following FRi with FRi+1, and takes out 32 bits from the bit position specified by the lower 6 bits of FRj in the made 64 bits data, and writes the result in FRk.
The MWCUTI instruction concatenates FRi and FRi+1, following FRi with FRi+1, and takes out 32 bits from the bit position specified by #u6 in the made 64 bits data, and writes the result in FRk.
MSB is 0 and LSB is 63, and according to the specified bit position, 32 bits data is taken out from the 64 bits data which is made by concatenating FRi and FRi+1. When FRj is greater than 32, "0" is written in the bits which are rest in low-order.
The processing exception detected by the instruction is initiated by executing MTRAP instruction.

Operation example

MWCUT FRi, FRj, FRk



Registers altered (except destination register)

None

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception)

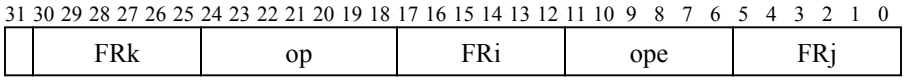
1.10.5. Average (Halfword Dual)

Ope-code	op	ope	Operation
MAVEH	1111011	001000	Dual Average Halfword

Category

Media

Instruction Format (Media instruction (R-R))



Assembler Syntax

MAVEH FRi, FRj, FRk

Description

The media average instruction calculates “FRihi + FRjhi” and writes the result in FRkhi as 16-bit integer after 1 bit right arithmetic shifting and calculates “FRilo + FRjlo” and writes the result in FRklo as 16-bit integer after 1 bit right arithmetic shifting.
When the result is an odd number, it is rounded to negative, minimum number.
For example,

3 -> (1.5) -> 1, 1 -> (0.5) -> 0, -1 -> (-0.5) -> -1,
-3 -> (-1.5) -> -2

The processing exception detected by the instruction is initiated by executing MTRAP instruction.
(“FRihi” indicates the high-order 16 bits of FRi and “FRilo” indicates the low-order 16 bits)

Registers altered (except destination register)

none

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception)

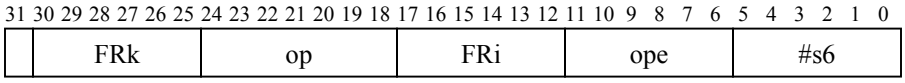
1.10.6. Shift (Halfword Dual)

Ope-code	op	ope	Operation
MSLLHI	1111011	001001	Dual Shift Left Logical Halfword (Immediate)
MSRLHI	1111011	001010	Dual Shift Right Logical Halfword (Immediate)
MSRAHI	1111011	001011	Dual Shift Right Arithmetic Logical Halfword (Immediate)

Category

Media

Instruction Format (Media instruction (R-R))



Assembler Syntax

MSLLHI FRi, #s6, FRk
MSRLHI FRi, #s6, FRk
MSRAHI FRi, #s6, FRk

Description

The media shift instruction (Halfword Dual) shifts FRi_{hi} and FRi_{lo} by the number specified by the shift-count, and writes the result in the FRk_{hi} and FRk_{lo}. The shift-count is specified by low-order four bits of #s6.
The MSLLHI instruction shifts FRi_{hi} and FRi_{lo} to the left by the value specified by the shift-count, replacing the vacated positions with zeros.
The MSRLHI instruction shifts FRi_{hi} and FRi_{lo} to the right by the value specified by the shift-count, replacing the vacated positions with zeros.
The MSRAHI instruction shifts FRi_{hi} and FRi_{lo} to the right by the value specified by the shift-count, replacing the vacated positions with highest bit of FRi_{hi} and FRi_{lo}. After that, this instruction rounds the result by the way specified by MSR0.
The processing exception detected by the instruction is initiated by executing MTRAP instruction.
("FRi_{hi}" indicates the high-order 16 bits of FRi and "FRi_{lo}" indicates the low-order 16 bits)

Registers altered (except destination register)

none

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception)

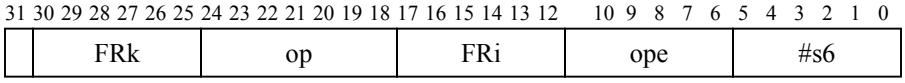
1.10.7. Media Dual Rotate (Word Dual) Instruction

Ope code	op	ope	Operation
MDROTLI	1111000	001011	Dual Rotate Left Immediate

Category

Media

Instruction format (Media instruction)



Assembler description

MDROTLI FRi,#s6,FRk

Description

The media dual rotate (Word Dual) instruction rotates FRi by the bit counts specified for lower 5 bits of #s6 and then stores the result of this rotation in FRk. Simultaneously, the instruction rotates "FRi+1" by the bit counts specified for lower 5 bits of #s6 and then stores the result of this rotation in "FRk+1".
The MDROTLI instruction rotates FRi and "FRi+1" to the left.
The register numbers of the Fri and FRk registers must be set to even numbers previously by software; when they are odd numbers, mp_exception (register_not_aligned) is detected.
For exceptions detected by this instruction, exception handling can be started by executing the MTRAP instruction.

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned)

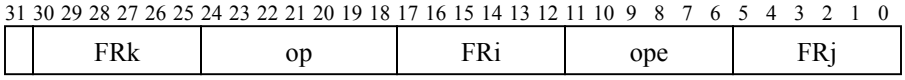
1.10.8. Saturate (Halfword Dual)

Ope-code	op	ope	Operation
MSATHS	1111011	001100	Dual Saturation Halfword for Signed
MSATHU	1111011	001101	Dual Saturation Halfword for Unsigned

Category

Media

Instruction Format (Media instruction)



Assembler Syntax

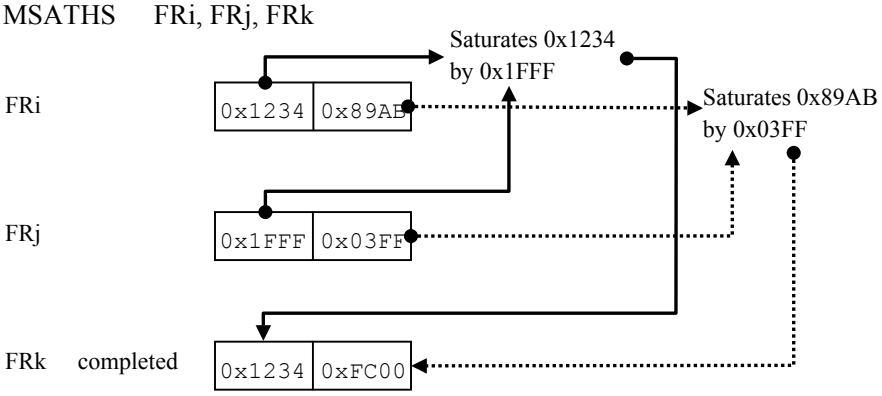
MSATHS FRi, FRj, FRk
MSATHU FRi, FRj, FRk

Description

The MSATHS instruction saturates 16-bit signed integer data of FRi$_{hi}$ within the range indicated by FRj$_{hi}$ and writes the result in FRk$_{hi}$ and saturates 16-bit signed integer data of FRi$_{lo}$ within the range indicated by FRj$_{lo}$ and writes the result in FRk$_{lo}$.
The data of FRi has two 16-bit signed integers. The data of FRj should be positive maximum numbers which are under 0x7fff and they should be "2^n-1" as well. The result is undefined when FRj is not "2^n-1".
The MSATHU instruction saturates 16-bit unsigned integer data of FRi$_{hi}$ within the range indicated by FRj$_{hi}$ and writes the result in FRk$_{hi}$ and saturates 16-bit unsigned integer data of FRi$_{lo}$ within the range indicated by FRj$_{lo}$ and writes the result in FRk$_{lo}$.
The data of FRi has two 16-bit unsigned integers. The data of FRj should be positive maximum numbers which are under 0xffff and they should be "2^n-1" as well. The result is undefined when FRj is not "2^n-1".
The processing exception detected by the instruction is initiated by executing MTRAP instruction.
("FRi$_{hi}$" indicates the high-order 16 bits of FRi and "FRi$_{lo}$" indicates the low-order 16 bits)

Operation example

MSATHS FRi, FRj, FRk



Registers altered (except destination register)

none

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception)

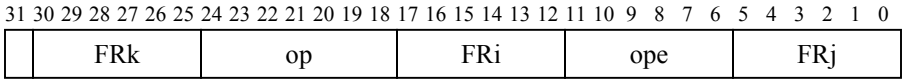
1.10.9. Media Quad Saturation Operation (Halfword Quad) Instruction

Ope code	op	ope	Operation
MQSATHS	1111000	001111	Quad Saturation Halfword for Signed

Category

Media

Instruction format (Media instruction)



Assembler description

MQSATHS FRi,FRj,FRk

Description

The MQSATHS instruction performs signed saturating for FRihi in the range shown by FRjhi and then stores the result in FRkhi. Simultaneously, the instruction performs signed saturating for FRilo in the range shown by FRjlo and then stores the result in FRklo. Simultaneously, the instruction performs signed saturating for "FRi+1hi" in the range shown by "FRj+1hi" and then stores the result in "FRk+1hi". Simultaneously, the instruction performs signed saturating for "FRi+1lo" in the range shown by "FRj+1lo" and then stores the result in "FRk+1lo".
FRi data and "FRi+1" data are two pairs of signed integer halfwords. FRj data and "FRj+1" data are two pairs of maximum positive values below 0x7fff to be subjected to saturation, and must be "$2^n - 1$". When FRjhi, FRjlo, "FRj+1hi", and "FRj+1lo" are not "$2^n - 1$", the result is undefined.
The register numbers of the FRi, FRj, and FRk registers must be set to even numbers previously by software; when they are set to an odd numbers, mp_exception (register_not_aligned) is detected.
For exceptions detected by this instruction, exception handling can be started by executing the MTRAP instruction.

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned)

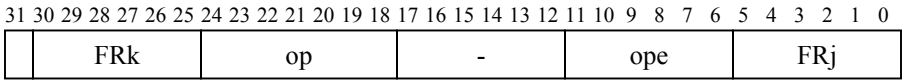
1.10.10. Media Absolute Value Operation (Halfword Dual) Instruction

Ope code	op	ope	Operation
MABSHS	1111000	001010	Dual Absolute

Category

Media

Instruction format (Media instruction)



Assembler description

MABSHS FRj,FRk

Description

The MABSHS instruction stores the absolute value of FRjhi in FRkhi and simultaneously stores the absolute value of FRjlo in FRklo. When the input value is 0x8000 (maximum negative value), mp_exception (overflow) is detected, the result (0x7FFF) obtained after signed saturating is stored in the register, and msr.ovf is set to 1.
For exceptions detected by this instruction, exception handling can be started by executing the MTRAP instruction.

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, overflow)

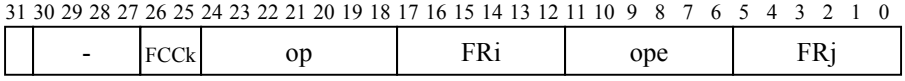
1.10.11. Compare (Halfword Dual)

Ope-code	op	ope	Operation
MCMPSH	1111011	001110	Compare Signed Halfword (Halfword Dual)
MCMPUH	1111011	001111	Compare Unsigned (Halfword Dual)

Category

Media

Instruction Format (Media instruction)



Assembler Syntax

MCMPSH FRi, FRj, FCCk
MCMPUH FRi, FRj, FCCk

Description

The MCMPSH instruction compares FRi_{hi} with FRj_{hi} as 16-bit signed integer and sets FCCk , and compares FRi_{lo} with FRj_{lo} as 16-bit signed integer and sets FCCk+1. The FCC is set according to the following table.
The MCMPUH instruction compares FRi_{hi} with FRj_{hi} as 16-bit unsigned integer and sets FCCk , and compares FRi_{lo} with FRj_{lo} as 16-bit unsigned integer and FCCk+1. The FCC is set according to the following table.
It is necessary to set FCCk as an even number with software. If FCCk is an odd number, mp_exception(cr_not_aligned) occur.
The processing exception detected by the instruction is initiated by executing MTRAP instruction.

FCCk	condition
3(EQ)	FRi_x = FRj_x (“x” is “ hi” or “ lo”)
2(LT)	FRi_x < FRj_x (“x” is “ hi” or “ lo”)
1(GT)	FRi_x > FRj_x (“x” is “hi” or “lo”)
0(UO)	(None)

Registers altered (except destination register)

FCCk,FCCk+1

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, cr_not_aligned)

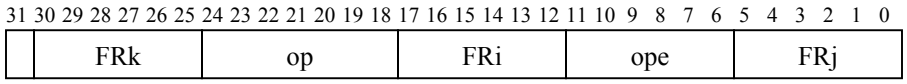
1.10.12. Add / Subtract with Saturation (Halfword Dual)

Ope-code	op	ope	Operation
MADDHSS	1111011	010000	Dual Add Signed Halfword with Saturation
MADDHUS	1111011	010001	Dual Add Unsigned Halfword with Saturation
MSUBHSS	1111011	010010	Dual Subtract Signed Halfword with Saturation
MSUBHUS	1111011	010011	Dual Subtract Unsigned Halfword with Saturation

Category

Media

Instruction Format (Media instruction)



Assembler Syntax

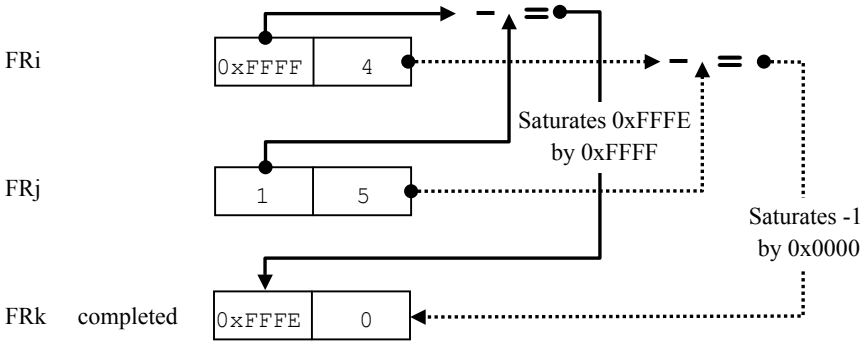
MADDHSS FRi, FRj, FRk
MADDHUS FRi, FRj, FRk
MSUBHSS FRi, FRj, FRk
MSUBHUS FRi, FRj, FRk

Description

The MADDHSS instruction calculates "FRi_{hi} +FRj_{hi} " and writes the result in FRk_{hi} as 16-bit integer after signed saturation processing, and calculates "FRi_{lo} +FRj_{lo} " and writes the result in FRk_{lo} as 16-bit integer after signed saturation processing. When the operation results overflow and the result is a positive number, the instruction writes positive, maximum number (0x7fff) in each halfword of FRk, or when the operation results overflow and the result is a negative number, the instruction writes negative, minimum number (0x8000) in each halfword of FRk.
The MADDHUS instruction calculates "FRi_{hi} +FRj_{hi} " and writes the result in FRk_{hi} as 16-bit integer after unsigned saturation processing, and calculates "FRi_{lo} +FRj_{lo} " and writes the result in FRk_{lo} as 16-bit integer after unsigned saturation processing. When the operation results overflow, the instruction writes positive, maximum number (0xffff) in each halfword of FRk.
The MSUBHSS instruction calculates "FRi_{hi} - FRj_{hi} " and writes the result in FRk_{hi} as 16-bit integer after signed saturation processing, and calculates "FRi_{lo} -FRj_{lo} " and writes the result in FRk_{lo} as 16-bit integer after signed saturation processing. When the operation results overflow and the result is positive number, the instruction writes positive, maximum number (0x7fff) in each halfword of FRk , or when the operation results overflow and the result is negative number, the instruction writes negative, minimum number (0x8000) in each halfword of FRk.
The MSUBHUS instruction calculates "FRi_{hi} - FRj_{hi} " and writes the result in FRk_{hi} as 16-bit integer after unsigned saturation processing, and calculates "FRi_{lo} -FRj_{lo}" and writes the result in FRk_{lo} as 16-bit integer after unsigned saturation processing. When the operation results overflow, the instruction writes minimum number (0) in each halfword of FRk
When the result is overflow, "1" is set in the msr.ovf. The processing of the exception detected by the instruction is initiated by executing MTRAP instruction.
("FRi_{hi}" indicates the high-order 16 bits of FRi, and "FRi_{lo}" indicates the low-order 16 bits of FRi.)

Operation example

MSUBHUS FRi, FRj, FRk



Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, overflow)

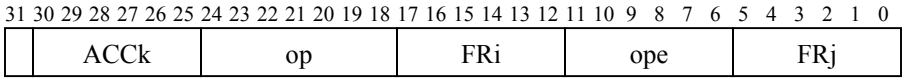
1.10.13. Multiply (Halfword Dual)

Ope-code	op	ope	Operation
MMULHS	1111011	010100	Dual Multiply Signed Halfword
MMULHU	1111011	010101	Dual Multiply Unsigned Halfword

Category

Media

Instruction Format (Media instruction)



Assembler Syntax

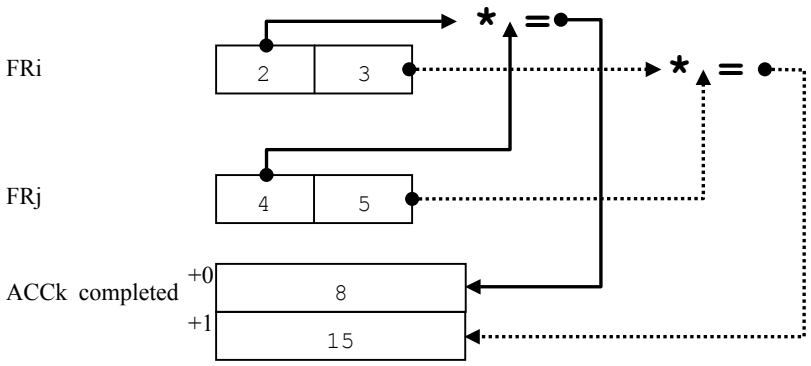
MMULHS FRi, FRj, ACCk
MMULHU FRi, FRj, ACCk

Description

The MMULHS instruction signed multiplies FRi$_{hi}$ by FRj$_{hi}$ and writes the result in the accumulator which concatenates ACCGk and ACCk as 40-bit integer, and signed multiplies FRi$_{lo}$ by FRj$_{lo}$ and writes the result in the accumulator which concatenates ACCGk+1 and ACCk+1 as 40-bit integer.
The MMULHU instruction unsigned multiplies FRi$_{hi}$ by FRj$_{hi}$ and writes the result in the accumulator which concatenates ACCGk and ACCk as 40-bit integer, and unsigned multiplies FRi$_{lo}$ by FRj$_{lo}$ and writes the result in the accumulator which concatenates ACCGk+1 and ACCk+1 as 40-bit integer.
It is necessary to set the accumulator number of ACCk as an even number with software. If the accumulator number of ACCk is an odd number, the mp_exception (acc_not_aligned) occurs.
The instruction can not change the value of msr.ovf. The processing of the exception detected by the instruction is initiated by executing MTRAP instruction.
("FRi$_{hi}$" indicates the high-order 16 bits of FRi, and "FRi$_{lo}$" indicates the low-order 16 bits of FRi.)

Operation example

MMULHS FRi, FRj, ACCk



Registers altered (except destination register)

none

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, acc_not_aligned, overflow)

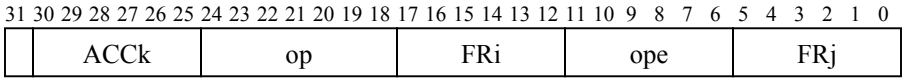
1.10.14. Cross Multiply (Halfword Dual)

Ope-code	op	ope	Operation
MMULXHS	1111011	101000	Dual Cross Multiply Signed Halfword
MMULXHU	1111011	101001	Dual Cross Multiply Unsigned Halfword

Category

Media

Instruction Format (Media instruction)



Assembler Syntax

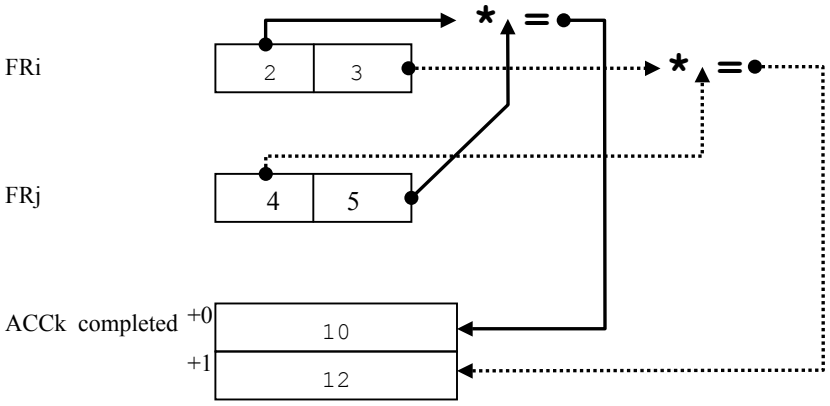
MMULXHS FRi, FRj, ACCk
MMULXHU FRi, FRj, ACCk

Description

The MMULXHS instruction signed multiplies FRi$_{hi}$ by FRj$_{lo}$ and writes the result in the accumulator which concatenates ACCGk and ACCk as 40-bit integer, and signed multiplies FRi$_{lo}$ by FRj$_{hi}$ and writes the result in the accumulator which concatenates ACCGk+1 and ACCk+1 as 40-bit integer.
The MMULXHU instruction unsigned multiplies FRi$_{hi}$ by FRj$_{lo}$ and writes the result in the accumulator which concatenates ACCGk and ACCk as 40-bit integer, and unsigned multiplies FRi$_{lo}$ by FRj$_{hi}$ and writes the result in the accumulator which concatenates ACCGk+1 and ACCk+1 as 40-bit integer.
It is necessary to set the accumulator number of ACCk as an even number with software. If the accumulator number of ACCk is an odd number, the mp_exception (acc_not_aligned) occurs.
The instruction can not change the value of msr.ovf. The processing of the exception detected by the instruction is initiated by executing MTRAP instruction.
("FRi$_{hi}$" indicates the high-order 16 bits of FRi, and "FRi$_{lo}$" indicates the low-order 16 bits of FRi.)

Operation example

MMULXHS FRi, FRj, ACCk



Registers altered (except destination register)

none

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, acc_not_aligned, overflow)

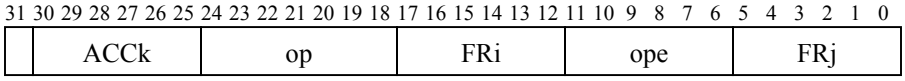
1.10.15. Multiply and Accumulate (Halfword Dual)

Ope-code	op	ope	Operation
MMACHS	1111011	010110	Dual Multiply and Accumulate Signed Halfword
MMACHU	1111011	010111	Dual Multiply and Accumulate Unsigned Halfword

Category

Media

Instruction Format (Media instruction)



Assembler Syntax

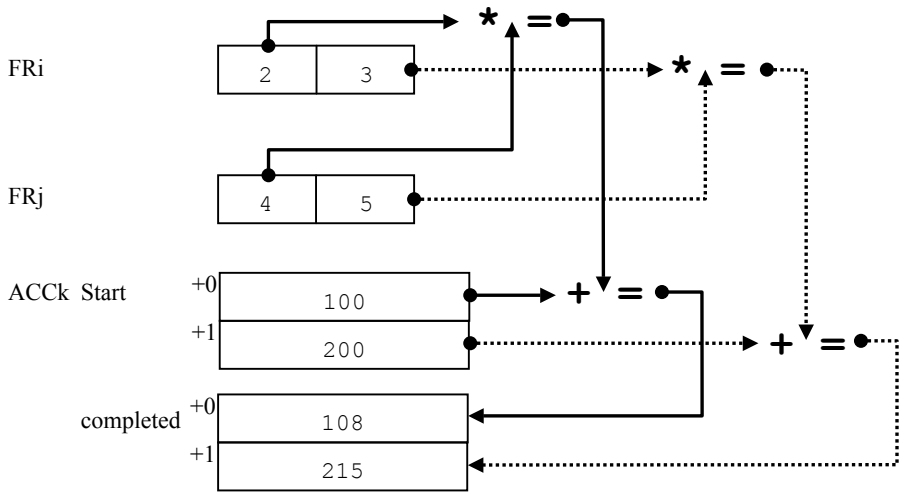
MMACHS FRi, FRj, ACCk
MMACHU FRi, FRj, ACCk

Description

The MMACHS instruction singed multiplies FRi_{hi} by FRj_{hi} and adds the result to the 40-bit accumulator which concatenates ACCGk and ACCk, and signed multiplies FRi_{lo} by FRj_{lo} and adds the result to the 40-bit accumulator which concatenates ACCGk+1 and ACCk+1. When the result is overflow, the result is written in the accumulator after saturated to signed value and msr.ovf is set to 1.
The MMACHU instruction unsigned multiplies FRi_{hi} by FRj_{hi} and adds the result to the 40-bit accumulator which concatenates ACCGk and ACCk, and unsigned multiplies FRi_{lo} by FRj_{lo} and adds the result to the 40-bit accumulator which concatenates ACCGk+1 and ACCk+1. When the result is overflow, the result is written in the accumulator after saturated to unsigned value and msr.ovf is set to 1.
It is necessary to set the accumulator number of ACCk as an even number with software. If the accumulator number of ACCk is an odd number, the mp_exception (acc_not_aligned) occurs.
When the result of addition cause the overflow, "1" is set in the msr.ovf. The processing of the exception detected by the instruction is initiated by executing MTRAP instruction.
("FRi_{hi}" indicates the high-order 16 bits of FRi, and "FRi_{lo}" indicates the low-order 16 bits of FRi.)

Operation example

MMACHS FRi, FRj, ACCk



Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, acc_not_aligned, overflow)

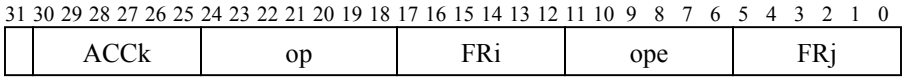
1.10.16. Multiply and Subtract (Halfword Dual)

Ope-code	op	ope	Operation
MMRDHS	1111011	110000	Dual Multiply and Subtract Signed Halfword
MMRDHU	1111011	110001	Dual Multiply and Subtract Unsigned Halfword

Category

Media

Instruction Format (Media instruction)



Assembler Syntax

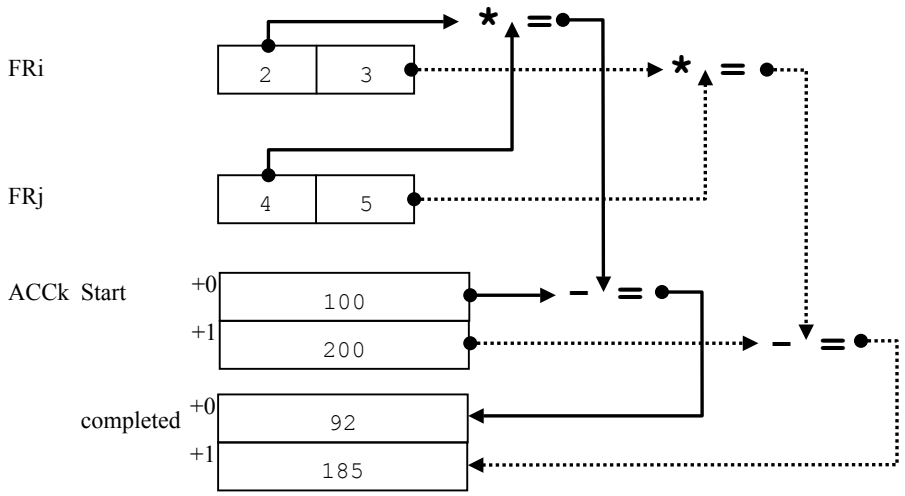
MMRDHS FRi, FRj, ACCk
MMRDHU FRi, FRj, ACCk

Description

The MMRDHS instruction signed multiplies FRi_{hi} by FRj_{hi} and subtracts the result from the 40-bit accumulator which concatenates ACCGk and ACCk, and signed multiplies FRi_{lo} by FRj_{lo} and subtracts the result from the 40-bit accumulator which concatenates ACCGk+1 and ACCk+1. When the result is overflow, the result is written in the accumulator after saturated to signed value and msr.ovf is set to 1.
The MMRDHU instruction unsigned multiplies FRi_{hi} by FRj_{hi} and subtracts the result from the 40-bit accumulator which concatenates ACCGk and ACCk, and unsigned multiplies FRi_{lo} by FRj_{lo} and subtracts the result from the 40-bit accumulator which concatenates ACCGk+1 and ACCk+1. When the result is overflow, the result is written in the accumulator after saturated to unsigned value(0) and msr.ovf is set to 1.
It is necessary to set the accumulator number of ACCk as an even number with software. If the accumulator number of ACCk is an odd number, the mp_exception(acc_not_aligned) occurs.
When the result of subtraction cause the overflow, "1" is set in the msr.ovf. The processing of the exception detected by the instruction is initiated by executing MTRAP instruction.
("FRi_{hi}" indicates the high-order 16 bits of FRi, and "FRi_{lo}" indicates the low-order 16 bits of FRi.)

Operation example

MMRDHS FRi, FRj, ACCk



Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, acc_not_aligned, overflow)

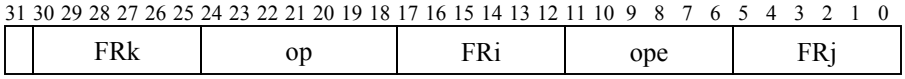
1.10.17. Add / Subtract with Saturation (Halfword Quad)

Ope-code	op	ope	Operation
MQADDHSS	1111011	011000	Quad Add Signed Halfword with Saturation
MQADDHUS	1111011	011001	Quad Add Unsigned Halfword with Saturation
MQSUBHSS	1111011	011010	Quad Subtract Signed Halfword with Saturation
MQSUBHUS	1111011	011011	Quad Subtract Unsigned Halfword with Saturation

Category

Media

Instruction Format (Media instruction)



Assembler Syntax

MQADDHSS FRi, FRj, FRk
MQADDHUS FRi, FRj, FRk
MQSUBHSS FRi, FRj, FRk
MQSUBHUS FRi, FRj, FRk

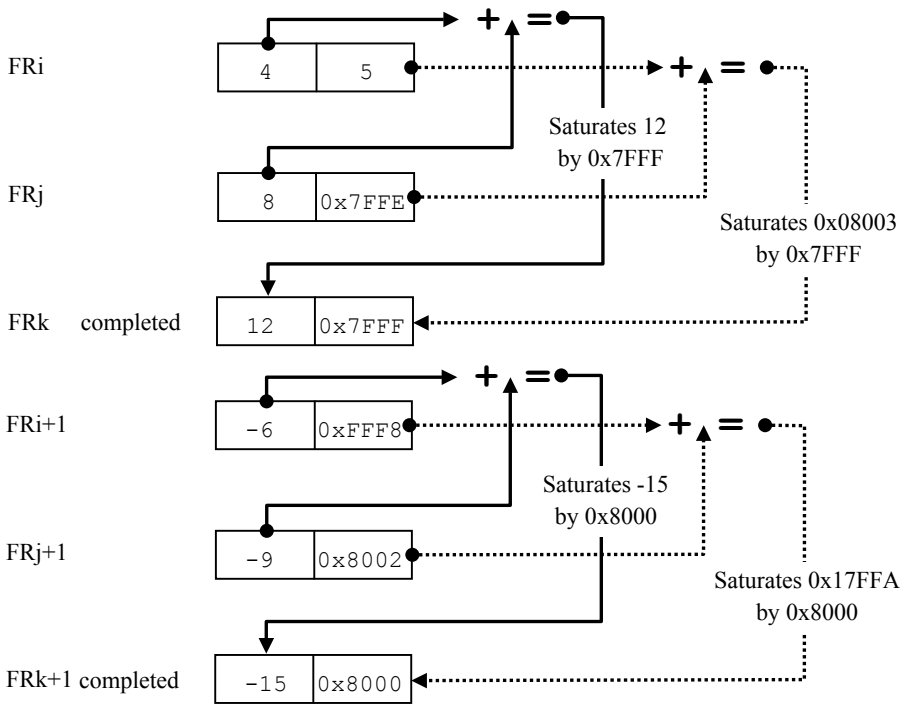
Description

The MQADDHSS instruction calculates "FRi_{hi} +FRj_{hi}" and writes the result in FRk_{hi} as 16-bit integer after signed saturation processing, and calculates "FRi_{lo} +FRj_{lo}" and writes the result in FRk_{lo} as 16-bit integer after signed saturation processing. When the operation results overflow and the result is positive number, the instruction writes positive, maximum number (0x7fff) in each halfword of FRk, or when the operation results overflow and the result is negative number, the instruction writes negative, minimum number (0x8000) in each halfword of FRk.
It also calculates "$FRi+1_{hi}$ +$FRj+1_{hi}$" and writes the result in $FRk+1_{hi}$ as 16-bit integer after signed saturation processing, and calculates "$FRi+1_{lo}$ +$FRj+1_{lo}$" and writes the result in $FRk+1_{lo}$ as 16-bits integer after signed saturation processing. When the operation results overflow and the result is positive number, the instruction writes positive, maximum number (0x7fff) in each halfword of FRk+1, or when the operation results overflow and the result is negative number, the instruction writes negative, minimum number (0x8000) in each halfword of FRk+1.
The MQADDHUS instruction calculates "FRi_{hi} +FRj_{hi}" and writes the result in FRk_{hi} as 16-bit integer after unsigned saturation processing, and calculates "FRi_{lo} +FRj_{lo}" and writes the result in FRk_{lo} as 16-bit integer after unsigned saturation processing. When the operation results overflow, the instruction writes positive, maximum number (0xffff) in each halfword of FRk.
It also calculates "$FRi+1_{hi}$ +$FRj+1_{hi}$" and writes the result in $FRk+1_{hi}$ as 16-bit integer after unsigned saturation processing, and calculates "$FRi+1_{lo}$ +$FRj+1_{lo}$" and writes the result in $FRk+1_{lo}$ as 16-bit integer after unsigned saturation processing. When the operation results overflow, the instruction writes positive, maximum number (0xffff) in each halfword of FRk+1.
The MQSUBHSS instruction calculates "FRi_{hi} -FRj_{hi}" and writes the result in FRk_{hi} as 16-bit integer after signed saturation processing, and calculates "FRi_{lo} -FRj_{lo}" and writes the result in FRk_{lo} as 16-bit integer after signed saturation processing. When the operation results overflow and the result is positive number, the instruction writes positive, maximum number (0x7fff) in

each halfword of FRk, or when the operation results overflow and the result is negative number, the instruction writes negative, minimum number (0x8000) in each halfword of FRk.
It also calculates "FRi+1$_{hi}$ -FRj+1$_{hi}$" and writes the result in FRk+1$_{hi}$ as 16-bit integer after signed saturation processing, and calculates "FRi+1$_{lo}$ -FRj+1$_{lo}$" and writes the result in FRk+1$_{lo}$ as 16-bit integer after signed saturation processing. When the operation results overflow and the result is positive number, the instruction writes positive, maximum number (0x7fff) in each halfword of FRk+1, or when the operation results overflow and the result is negative number, the instruction writes negative, minimum number (0x8000) in each halfword of FRk+1.
The MQSUBHUS instruction calculates "FRi$_{hi}$ -FRj$_{hi}$" and writes the result in FRk$_{hi}$ as 16-bit integer after unsigned saturation processing, and calculates "FRi$_{lo}$ -FRj$_{lo}$" and writes the result in FRk$_{lo}$ as 16-bit integer after unsigned saturation processing. When the operation results overflow and the result is a negative number , the instruction writes minimum number (0x0) in each halfword of FRk.
It also calculates "FRi+1$_{hi}$ -FRj+1$_{hi}$" and writes the result in FRk+1$_{hi}$ as 16-bit integer after unsigned saturation processing, and calculates "FRi+1$_{lo}$ -FRj+1$_{lo}$" and writes the result in FRk+1$_{lo}$ as 16-bit integer after unsigned saturation processing. when the operation results overflow and the result is a negative number, the instruction writes minimum number (0x0) in each halfword of FRk+1.
It is necessary to set the register number of FRi, FRj and FRk in an even number with software. If the register number of FRi, FRj or FRk is an odd number, the mp_exception (register_not_aligned) occur.
When the operation results overflow, the instruction writes the saturated number in each FRk or FRk+1 and "1" is set in the msr.ovf. The processing of the exception detected by the instruction is initiated by executing MTRAP instruction.
("FRi$_{hi}$" indicates the high-order 16 bits of FRi, and "FRi$_{lo}$" indicates the low-order 16 bits of FRi.)

Operation example

MQADDHSS FRi, FRj, FRk



Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned, overflow)

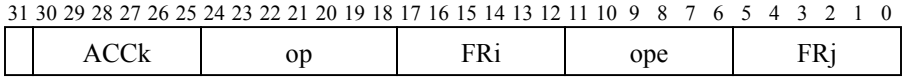
1.10.18. Multiply (Halfword Quad)

Ope-code	op	ope	Operation
MQMULHS	1111011	011100	Quad Multiply Signed Halfword
MQMULHU	1111011	011101	Quad Multiply Unsigned Halfword

Category

Media

Instruction Format (Media accumulator instruction)



Assembler Syntax

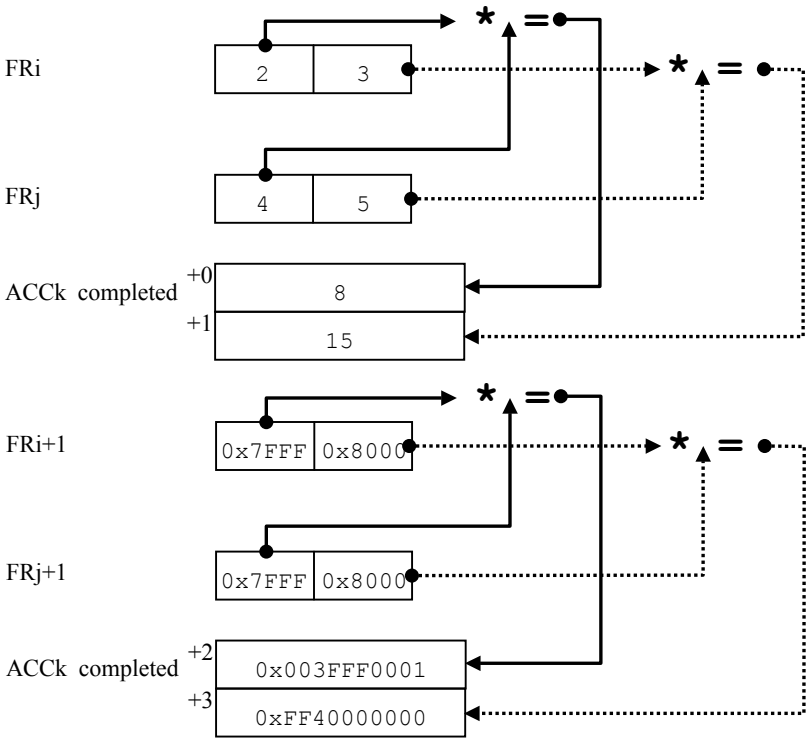
MQMULHS FRi, FRj, ACCk
MQMULHU FRi, FRj, ACCk

Description

The MQMULHS instruction signed multiplies FRi_{hi} by FRj_{hi} and writes the result in the 40-bit accumulator which concatenates ACCGk and ACCk, and signed multiplies FRi_{lo} by FRj_{lo} and writes the result in the 40-bit accumulator which concatenates ACCGk+1 and ACCk+1, and signed multiplies $FRi+1_{hi}$ by $FRj+1_{hi}$ and writes the result in the 40-bit accumulator which concatenates ACCGk+2 and ACCk+2, and signed multiplies $FRi+1_{lo}$ by $FRj+1_{lo}$ and writes the result in the 40-bit accumulator which concatenates ACCGk+3 and ACCk+3.
The MQMULHU instruction unsigned multiplies FRi_{hi} by FRj_{hi} and writes the result in the 40-bit accumulator which concatenates ACCGk and ACCk, and unsigned multiplies FRi_{lo} by FRj_{lo} and writes the result in the 40-bit accumulator which concatenates ACCGk+1 and ACCk+1, and unsigned multiplies $FRi+1_{hi}$ by $FRj+1_{hi}$ and writes the result in the 40-bit accumulator which concatenates ACCGk+2 and ACCk+2, and unsigned multiplies $FRi+1_{lo}$ by $FRj+1_{lo}$ and writes the result in the 40-bit accumulator which concatenates ACCGk+3 and ACCk+3.
It is necessary to set the register number of FRi and FRj as an even number with software. If the register number of FRi or FRj is an odd number, the mp_exception (register_not_aligned) occurs.
It is necessary to set the accumulator number of ACCk as a multiple of four. If the accumulator number of ACCk is not a multiple of four, the mp_exception (acc_not_aligned) occurs.
The instructions can not change the value of msr.ovf.
("FRi_{hi}" indicates the high-order 16 bits of FRi, and "FRi_{lo}" indicates the low-order 16 bits of FRi.)

Operation example

MQMULHS FRi, FRj, ACCk



Registers altered (except destination register)

none

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned, acc_not_aligned)

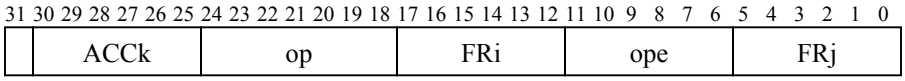
1.10.19. Cross Multiply (Halfword Quad)

Ope-code	op	ope	Operation
MQMULXHS	1111011	101010	Quad Cross Multiply Signed Halfword
MQMULXHU	1111011	101011	Quad Cross Multiply Unsigned Halfword

Category

Media

Instruction Format (Media accumulator instruction)



Assembler Syntax

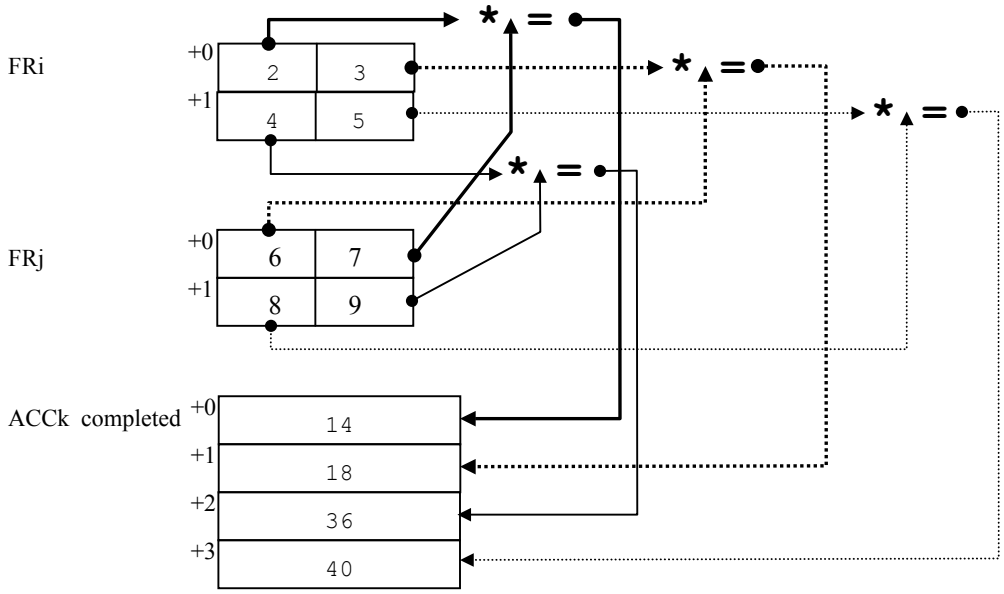
MQMULXHS FRi, FRj, ACCk
MQMULXHU FRi, FRj, ACCk

Description

The MQMULXHS instruction signed multiplies FRi_{hi} by FRj_{lo} and writes the result in the 40-bit accumulator which concatenates ACCGk and ACCk, and signed multiplies FRi_{lo} by FRj_{hi} and writes the result in the 40-bit accumulator which concatenates ACCGk+1 and ACCk+1, and signed multiplies $FRi+1_{hi}$ by $FRj+1_{lo}$ and writes the result in the 40-bit accumulator which concatenates ACCGk+2 and ACCk+2, and signed multiplies $FRi+1_{lo}$ by $FRj+1_{hi}$ and writes the result in the 40-bit accumulator which concatenates ACCGk+3 and ACCk+3.
The MQMULXHU instruction unsigned multiplies FRi_{hi} by FRj_{lo} and writes the result in the 40-bit accumulator which concatenates ACCGk and ACCk, and unsigned multiplies FRi_{lo} by FRj_{hi} and writes the result in the 40-bit accumulator which concatenates ACCGk+1 and ACCk+1, and unsigned multiplies $FRi+1_{hi}$ by $FRj+1_{lo}$ and writes the result in the 40-bit accumulator which concatenates ACCGk+2 and ACCk+2, and unsigned multiplies $FRi+1_{lo}$ by $FRj+1_{hi}$ and writes the result in the 40-bit accumulator which concatenates ACCGk+3 and ACCk+3.
It is necessary to set the register number of FRi and FRj as an even number with software. If the register number of FRi or FRj is an odd number, the mp_exception (register_not_aligned) occurs.
It is necessary to set the accumulator number of ACCk as a multiple of four. If the accumulator number of ACCk is not a multiple of four, the mp_exception (acc_not_aligned) occurs.
The instructions can not change the value of msr.ovf, holding the previous value.
("FRi_{hi}" indicates the high-order 16 bits of FRi, and "FRi_{lo}" indicates the low-order 16 bits of FRi.)

Operation example

MQMULXHS FRi, FRj, ACCk



Registers altered (except destination register)

none

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned, acc_not_aligned)

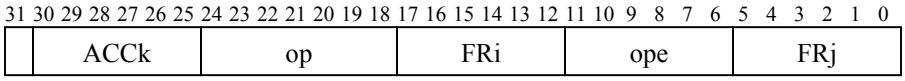
1.10.20. Multiply and Accumulate (Halfword Quad)

Ope-code	op	ope	Operation
MQMACHS	1111011	011110	Quad Multiply and Accumulate Signed Halfword
MQMACHU	1111011	011111	Quad Multiply and Accumulate Unsigned Halfword

Category

Media

Instruction Format (Media accumulator instruction)



Assembler Syntax

MQMACHS FRi, FRj, ACCk
MQMACHUFRi, FRj, ACCk

Description

The MQMACHS instruction signed multiplies FRi_{hi} by FRj_{hi} and adds the result to the 40-bit accumulator which concatenates ACCGk and ACCk, and signed multiplies FRi_{lo} by FRj_{lo} and adds the result to the 40-bit accumulator which concatenates ACCGk+1 and ACCk+1, and signed multiplies $FRi+1_{hi}$ by $FRj+1_{hi}$ and adds the result to the 40-bit accumulator which concatenates ACCGk+2 and ACCk+2, and signed multiplies $FRi+1_{lo}$ by $FRj+1_{lo}$ and adds the result to the 40-bit accumulator which concatenates ACCGk+3 and ACCk+3. When the result is overflow, the result is written in the accumulator after saturated to signed value and msr.ovf is set to 1.

The MQMACHU instruction unsigned multiplies FRi_{hi} by FRj_{hi} and adds the result to the 40-bit accumulator which concatenates ACCGk and ACCk, and unsigned multiplies FRi_{lo} by FRj_{lo} and adds the result to the 40-bit accumulator which concatenates ACCGk+1 and ACCk+1, and unsigned multiplies $FRi+1_{hi}$ by $FRj+1_{hi}$ and adds the result to the 40-bit accumulator which concatenates ACCGk+2 and ACCk+2, and unsigned multiplies $FRi+1_{lo}$ by $FRj+1_{lo}$ and adds the result to the 40-bit accumulator which concatenates ACCGk+3 and ACCk+3. When the result is overflow, the result is written in the accumulator after saturated to unsigned value and msr.ovf is set to 1.

It is necessary to set the register number of FRi and FRj as an even number with software. If the register number of FRi or FRj is an odd number, the mp_exception (register_not_aligned) occurs.

It is necessary to set the accumulator number of ACCk as a multiple of four with software. If the accumulator number of ACCk is not a multiple of four, the mp_exception (acc_not_aligned) occurs.

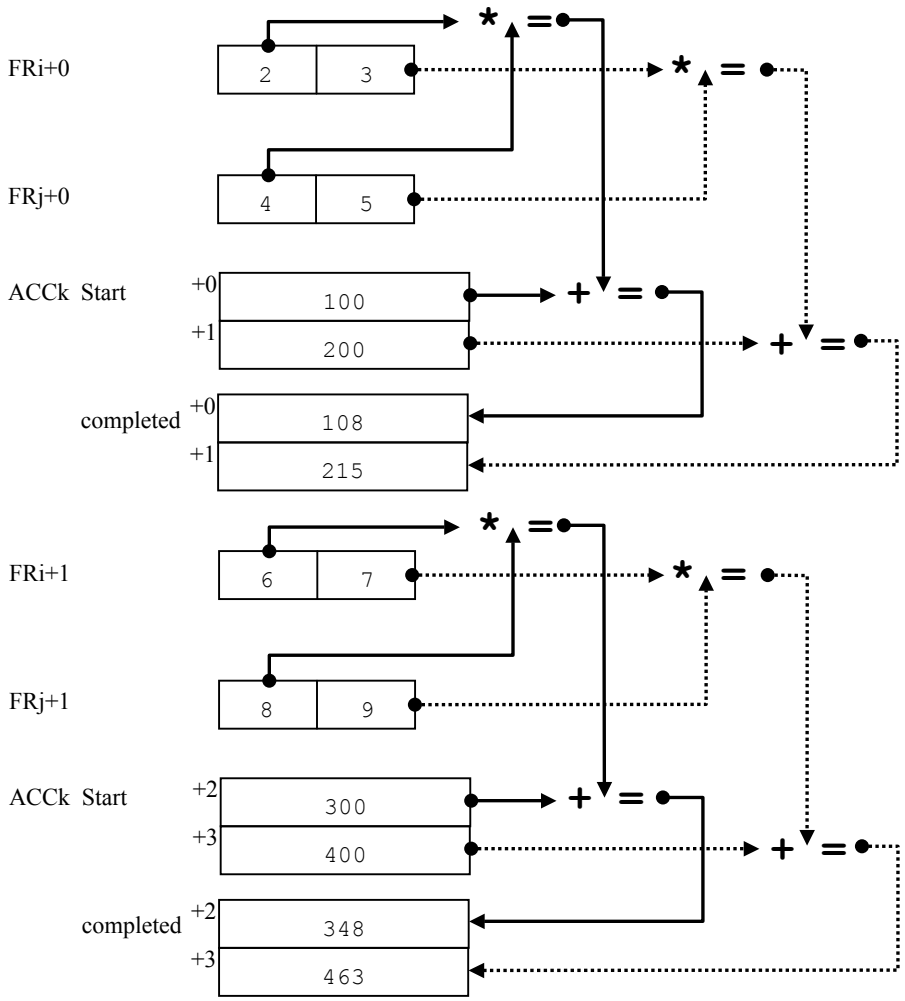
When the result of addition is overflow, "1" is set in msr.ovf.

The processing of the exception detected by the instruction is initiated by executing MTRAP instruction.

("FRi_{hi}" indicates the high-order 16 bits of FRi, and "FRi_{lo}" indicates the low-order 16 bits of FRi.)

Operation example

MQMACHS FRi, FRj, ACCk



Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned, acc_not_aligned , overflow)

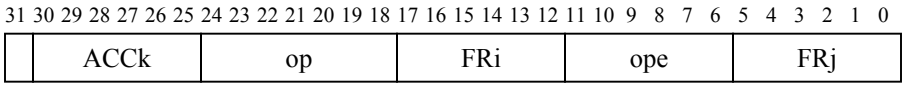
1.10.21. Media ACC Cross Quad Multiply and Accumulation (Halfword Quad) Instruction)

Ope code	op	ope	Operation
MQXMACHS	1111000	000000	Quad Multiply and Cross Accumulate Signed Halfword

Category

Media

Instruction format (Media accumulator instruction)



Assembler description

MQXMACHS FRi,FRj,ACCk

Description

The MQXMACHS instruction calculates FRihi*FRjhi, adds the result obtained from this calculation to the 40-bit integer data in which “ACCGk+2” and “ACCk+2” are concatenated, and then stores the result of this addition in the 40-bit register in which “ACCGk+2” and “ACCk+2” are concatenated, as a 40-bit signed integer. Simultaneously, the MQXMACHS instruction calculates FRilo*FRjlo, adds the result obtained from this calculation to the 40-bit integer data in which “ACCGk+3” and “ACCk+3” are concatenated, and then stores the result of this addition in the 40-bit register in which “ACCGk+3” and “ACCk+3” are concatenated, as a 40-bit signed integer. Simultaneously, the MQXMACHS instruction calculates FRi+1hi*FRj+1hi, adds the result obtained from this calculation to the 40-bit integer data in which ACCGk and ACCk are concatenated, and then stores the result of this addition in the register in which ACCGk and ACCk are concatenated, as a 40-bit signed integer. Simultaneously, the MQXMACHS instruction calculates FRi+1lo*FRj+1lo, adds the result obtained from this calculation to the 40-bit integer data in which “ACCGk+1” and “ACCk+1” are concatenated, and then stores the result of this addition in the 40-bit register in which “ACCGk+1” and “ACCk+1” are concatenated, as a 40-bit signed integer. When the result overflows, the result obtained after signed saturating is stored in the register, and `msr.ovf` is set to 1.

The register numbers of the Fri and FRj registers must be set to even numbers previously by software; if they are set to odd numbers, mp_exception (register_not_aligned) is detected.

The register number of the ACCk register must be set to a multiple of 4 previously by software; otherwise, mp_exception (acc_not_aligned) is detected.

For exceptions detected by this instruction, exception handling can be started by executing the MTRAP instruction.

(FRihi shows upper 16 bits of FRi; and FRilo shows lower 16 bits of FRi.)

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned,
acc_not_aligned , overflow)

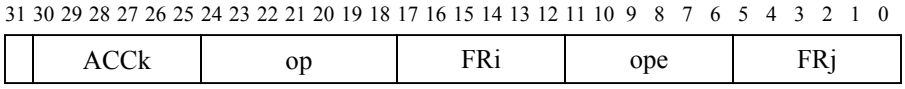
1.10.22. Media ACC Cross Quad Cross Multiply and Accumulation (Halfword Quad) Instruction

Ope code	op	ope	Operation
MQXMACXHS	1111000	000001	Quad Cross Multiply and Cross Accumulate Signed Halfword

Category

Media

Instruction format (Media accumulator instruction)



Assembler description

MQXMACXHS FRi,FRj,ACCk

Description

The MQXMACXHS instruction calculates FRihi*FRjlo, adds the result obtained from this calculation to the 40-bit integer data in which “ACCGk+2” and “ACCk+2” are concatenated, and then stores the result of this addition in the 40-bit register in which “ACCGk+2” and “ACCk+2” are concatenated, as a 40-bit signed integer. Simultaneously, the instruction calculates FRilo*FRjhi, adds the result obtained from this calculation to the 40-bit integer data in which ACCGk+3 and ACCk+3 are concatenated, and then stores the result of this addition in the 40-bit register in which “ACCGk+3” and “ACCk+3” are concatenated, as a 40-bit signed integer. Simultaneously, the instruction calculates FRi+1hi*FRj+1lo, adds the result obtained from this calculation to the 40-bit integer data in which ACCGk and ACCk are concatenated, and then stores the result of this addition in the register in which ACCGk and ACCk are concatenated, as a 40-bit signed integer. Simultaneously, the instruction calculates FRi+1lo*FRj+1hi, adds the result obtained from this calculation to the 40-bit integer data in which “ACCGk+1” and “ACCk+1” are concatenated, and then stores the result of this addition in the 40-bit register in which “ACCGk+1” and “ACCk+1” are concatenated, as a 40-bit signed integer. When the result overflows, the result obtained after signed saturating is stored in the register, and msr.ovf is set to 1.
The register numbers of the Fri and FRj registers must be set previously to even numbers by software; if they are odd numbers, mp_exception (register_not_aligned) is detected.
The register number of the ACCk register must be set to a multiple of 4 previously by software; otherwise, mp_exception (acc_not_aligned) is detected.
For exceptions detected by this instruction, exception handling can be started by executing the MTRAP instruction.
(FRihi shows upper 16 bits of FRi; and FRilo shows lower 16 bits of FRi.)

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned,
acc_not_aligned , overflow)

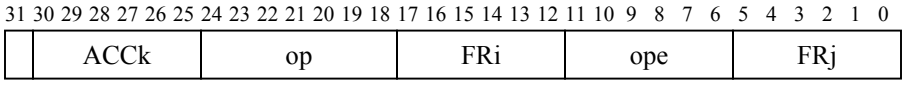
1.10.23. Media Quad Cross Multiply and Accumulation (Halfword Quad) Instruction

Ope code	op	ope	Operation
MQMACXHS	1111000	000010	Quad Cross Multiply and Cross Accumulate Signed Halfword

Category

Media

Instruction format (Media accumulator instruction)



Assembler description

MQMACXHS FRi,FRj,ACCk

Description

The MQMACXHS instruction calculates FRihi*FRjlo, adds the result obtained from this calculation to the 40-bit integer data in which ACCGk and ACCk are concatenated, and then stores the result of this addition in the 40-bit register in which ACCGk and ACCk are concatenated, as a 40-bit signed integer. Simultaneously, the instruction calculates FRilo*FRjhi, adds the result obtained from this calculation to the 40-bit integer data in which "ACCGk+1" and "ACCk+1" are concatenated, and then stores the result of this addition in the 40-bit register in which "ACCGk+1" and "ACCk+1" are concatenated, as a 40-bit signed integer. Simultaneously, the instruction calculates FRi+1hi*FRj+1lo, adds the result obtained from this calculation to the 40-bit integer data in which "ACCGk+2" and "ACCk +2" are concatenated, and then stores the result of this addition in the register in which "ACCGk+2" and "ACCk+2" are concatenated, as a 40-bit signed integer. Simultaneously, the instruction calculates FRi+1lo*FRj+1hi, adds the result obtained from this calculation to the 40-bit integer data in which "ACCGk+3" and "ACCk+3" are concatenated, and then stores the result of this addition in the 40-bit register in which "ACCGk+3" and "ACCk+3" are concatenated, as a 40-bit signed integer. When the result overflows, a value obtained after signed saturating is stored in the register, and msr.ovf is set to 1.
The register numbers of the Fri and FRj registers must be set to even numbers previously by software; if they are set to odd numbers, mp_exception (register_not_aligned) is detected.
The register number of the ACCk register must be set to a multiple of 4 previously by software; otherwise, mp_exception (acc_not_aligned) is detected.
For exceptions detected by this instruction, exception handling can be started by executing the MTRAP instruction.
(FRihi shows upper 16 bits of FRi; and FRilo shows lower 16 bits of FRi.)

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned,
acc_not_aligned , overflow)

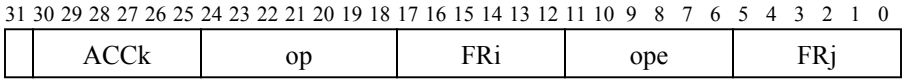
1.10.24. Complex Multiply (Halfword Dual)

Ope-code	Op	ope	Operation
MCPXRS	1111011	100000	Dual Complex Real Signed Halfword
MCPXRU	1111011	100001	Dual Complex Real Unsigned Halfword
MCPXIS	1111011	100010	Dual Complex Imaginary Signed Halfword
MCPXIU	1111011	100011	Dual Complex Imaginary Unsigned Halfword

Category

Media

Instruction Format (Media instruction)



Assembler Syntax

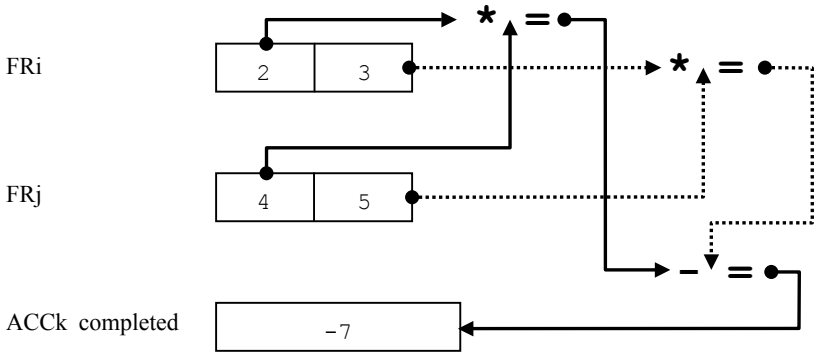
MCPXRS FRi, FRj, ACCk
MCPXRU FRi, FRj, ACCk
MCPXIS FRi, FRj, ACCk
MCPXIU FRi, FRj, ACCk

Description

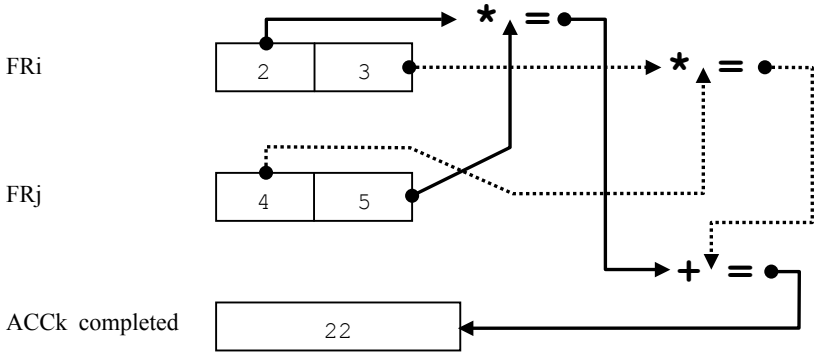
The MCPXRS instruction calculates "(FRi_{hi} * FRj_{hi}) – (FRi_{lo} * FRj_{lo})" as a signed integer and writes the result in the 40-bit accumulator which concatenates ACCGk and ACCk.
The MCPXRU instruction calculates "(FRi_{hi} * FRj_{hi}) – (FRi_{lo} * FRj_{lo})" as an unsigned integer and writes the result in the 40-bit accumulator which concatenates ACCGk and ACCk. The accumulator is set to 0 and msr.ovf is set to 1 when the subtracted result is minus.
The MCPXIS instruction calculates "(FRi_{hi} * FRj_{lo}) + (FRi_{lo} * FRj_{hi})" as a singed integer and writes the result in the 40-bit accumulator which concatenates ACCGk and ACCk.
The MCPXIU instruction calculates "(FRi_{hi} * FRj_{lo}) + (FRi_{lo} * FRj_{hi})" as an unsigned integer and writes the result in the 40-bit accumulator which concatenates ACCGk and ACCk.
The processing of the exception detected by the instruction is initiated by executing MTRAP instruction.
("FRi_{hi}" indicates the high-order 16 bits of FRi, and "FRi_{lo}" indicates the low-order 16 bits of FRi.)

Operation example

MCPXRS FRi, FRj, ACCk



MCPXIS FRi, FRj, ACCk



Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, acc_not_aligned, overflow)

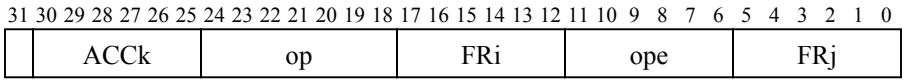
1.10.25. Complex Multiply (Halfword Quad)

Ope-code	Op	Ope	Operation
MQCPXRS	1111011	100100	Quad Complex Real Signed Halfword
MQCPXRU	1111011	100101	Quad Complex Real Unsigned Halfword
MQCPXIS	1111011	100110	Quad Complex Imaginary Signed Halfword
MQCPXIU	1111011	100111	Quad Complex Imaginary Unsigned Halfword

Category

Media

Instruction Format (Media instruction)



Assembler Syntax

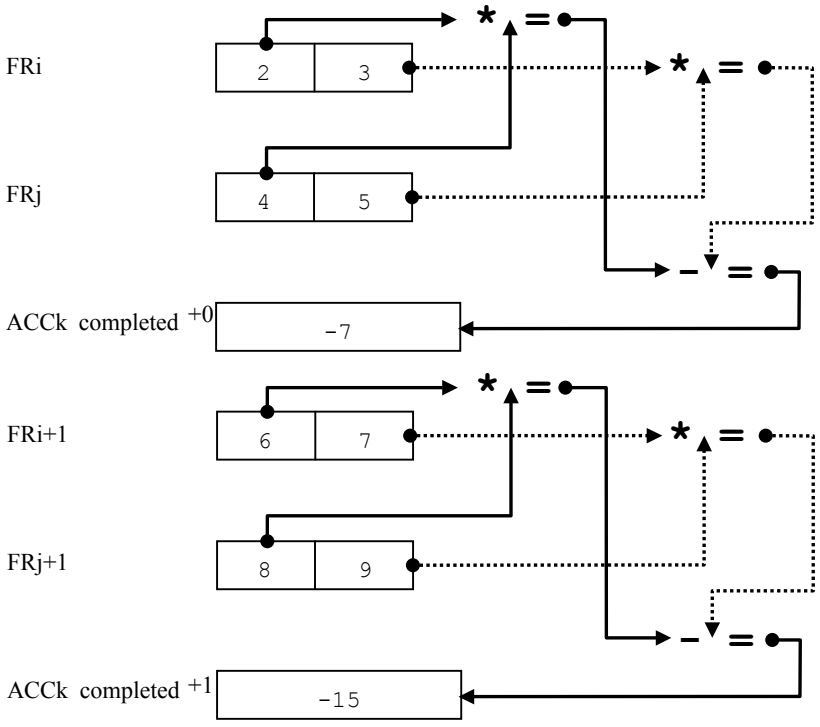
MQCPXRS FRi, FRj, ACCk
MQCPXRU FRi, FRj, ACCk
MQCPXIS FRi, FRj, ACCk
MQCPXIU FRi, FRj, ACCk

Description

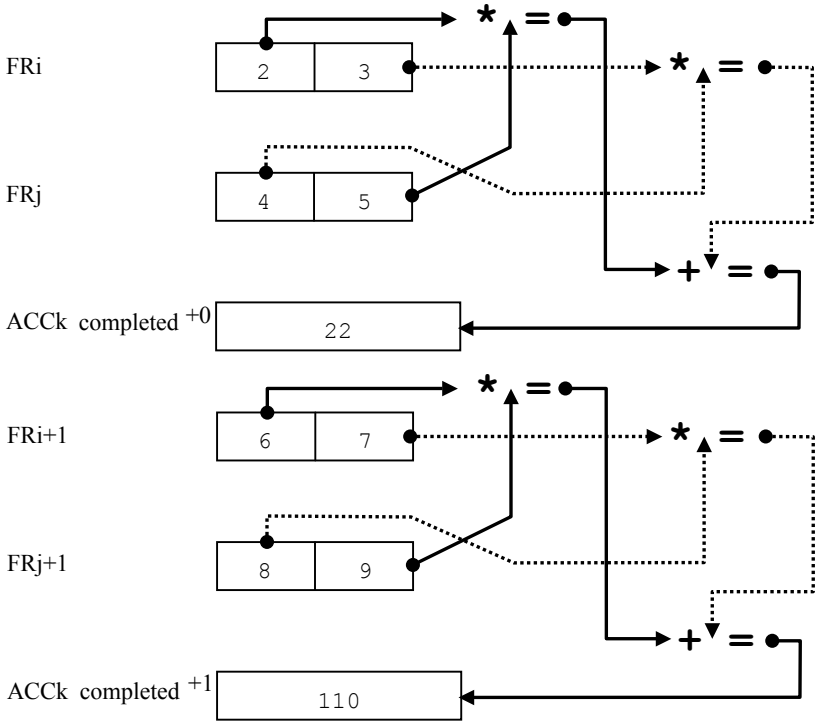
The MQCPXRS instruction calculates "(FRi$_{hi}$ * FRj$_{hi}$) – (FRi$_{lo}$ * FRj$_{lo}$)" as a signed integer and writes the result in the 40-bit accumulator which concatenates ACCGk and ACCk, and calculates "(FRi+1$_{hi}$ * FRj+1$_{hi}$) – (FRi+1$_{lo}$ * FRj+1$_{lo}$)" as a signed integer and writes the result in the 40-bit accumulator which concatenates ACCGk+1 and ACCk+1.
The MQCPXRU instruction calculates "(FRi$_{hi}$ * FRj$_{hi}$) – (FRi$_{lo}$ * FRj$_{lo}$)" as an unsigned integer and writes the result in the 40-bit accumulator which concatenates ACCGk and ACCk, and calculates "(FRi+1$_{hi}$ * FRj+1$_{hi}$) – (FRi+1$_{lo}$ * FRj+1$_{lo}$)" as an unsigned integer and writes the result in the 4-bit accumulator which concatenates ACCGk+1 and ACCk+1. The accumulator is set to 0 and msr.ovf is set to 1 when the subtracted result is minus.
The MQCPXIS instruction calculates "(FRi$_{hi}$ * FRj$_{lo}$) + (FRi$_{lo}$ * FRj$_{hi}$)" as a signed integer and writes the result in the 40-bit accumulator which concatenates ACCGk and ACCk, and calculates "(FRi+1$_{hi}$ * FRj+1$_{lo}$) + (FRi+1$_{lo}$ * FRj+1$_{hi}$)" as a signed integer and writes the result in the 40-bit accumulator which concatenates ACCGk+1 and ACCk+1.
The MQCPXIU instruction calculates "(FRi$_{hi}$ * FRj$_{lo}$) + (FRi$_{lo}$ * FRj$_{hi}$)" as an unsigned integer and writes the result in the 40-bit accumulator which concatenates ACCGk and ACCk, and calculates "(FRi+1$_{hi}$ * FRj+1$_{lo}$) + (FRi+1$_{lo}$ * FRj+1$_{hi}$)" as an unsigned integer and writes the result in the 40-bit accumulator which concatenates ACCGk+1 and ACCk+1.
When the operation results overflow, "1" is set in the msr.ovf. The processing of the exception detected by the instruction is initiated by executing MTRAP instruction.
("FRi$_{hi}$" indicates the high-order 16 bits of FRi, and "FRi$_{lo}$" indicates the low-order 16 bits of FRi.)

Operation example

MQCPXRS FRi, FRj, ACCk



MQCPXIS FRi, FRj, ACCk



Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, acc_not_aligned, overflow)

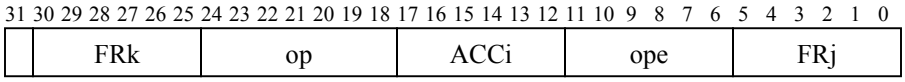
1.10.26. Cut

Ope-code	op	ope	Operation
MCUT	1111011	101100	CUT Accumulator
MCUTI	1111011	101110	CUT Accumulator (Immediate)

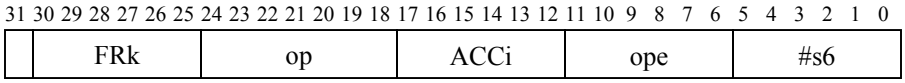
Category

Media
Single word

Instruction Format (Media instruction (R))



Instruction Format (Media instruction (imm))



Assembler Syntax

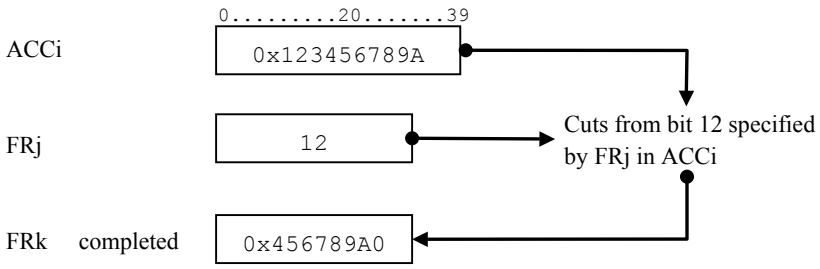
MCUT ACCi, FRj, FRk
MCUTI ACCi, #s6, FRk

Description

The MCUT instruction cuts the accumulator which concatenates ACCGi and ACCi as 32-bit integer from bit position which is indicated by the shift count and writes the result in the FRk.
MSB of the 40 bit data, which is made by concatenating ACCGi and ACCi is 0 and LSB is 39, and according to specified bit position, 32-bit data is taken out from the 40-bit data. The value greater than 31 can not be indicated as the shift count. But the negative number can be indicated as the shift count. When the negative number is indicated as the shift count, the sign bit is written into the negative bit positions.
The shift count is indicated by the lower 6 bits of FRj (immediate instruction: the 6 bits of #s6) as a signed integer.
The processing exception detected by the instruction is initiated by executing MTRAP instruction.

Operation example

MCUT ACCi, FRj, FRk



Registers altered (except destination register)

none

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception)

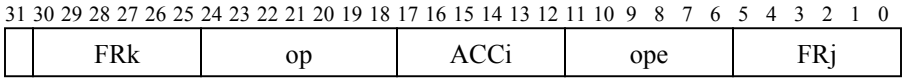
1.10.27. Cut

Ope-code	op	ope	Operation
MCUTSS	1111011	101101	CUT with Signed Saturation Accumulator
MCUTSSI	1111011	101111	CUT with Signed Saturation Accumulator (Immediate)

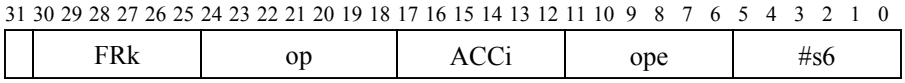
Category

Media
Single word

Instruction Format (Media instruction (R))



Instruction Format (Media instruction (imm))



Assembler Syntax

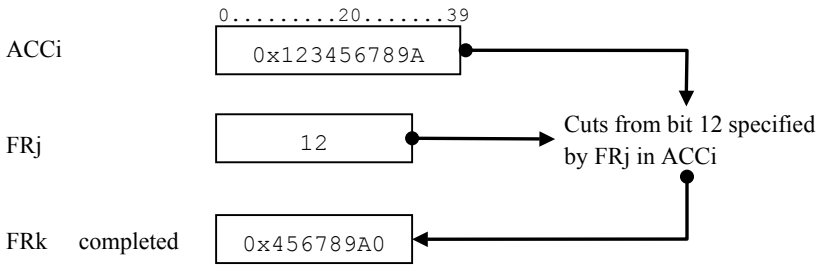
MCUTSS ACCi, FRj, FRk
MCUTSSI ACCi, #s6, FRk

Description

The MCUTSS instruction cuts the accumulator which concatenates ACCGi and ACCi as 32-bit integer from bit position which is indicated by the shift count and writes the result in the FRk after signed saturation processing.
MSB of the 40 bit data, which is made by concatenating ACCGi and ACCi is 0 and LSB is 39, and according to specified bit position, 32-bit data is taken out from the 40-bit data. The value greater than 31 can not be indicated as the shift count. But the negative number can be indicated as the shift count. When the negative number is indicated as the shift count, the sign bit is written into the negative bit positions.
The shift count is indicated by the lower 6 bits of FRj (immediate instruction: the 6 bits of #s6) as a signed integer.
When the result overflows, the result obtained after signed saturating is stored in the register, and `msr.ovf` is set to 1.
The processing exception detected by the instruction is initiated by executing MTRAP instruction.

Operation example

MCUTSS ACCi, FRj, FRk



Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception,overflow)

1.10.28. Media Dual Cut Instruction

Ope code	op	ope	Operation
MDCUTSSI	1111000	001110	Dual CUT and Signed Saturation Accumulator (Immediate)

Category

Media

Instruction format (Media instruction)

31	30 29 28 27 26 25	24 23 22 21 20	19 18 17 16 15 14	13 12 11 10 9 8 7 6	5 4 3 2 1 0
	FRk	op	ACCi	ope	#s6

Assembler description

MDCUTSSI ACCi,#s6,FRk

Description

The MDCUTSSI instruction extracts 32 bits out of 40-bit data in which ACCGi and ACCi are concatenated, beginning at the bit position specified for the shift count, performs signed saturating, and then stores the result in FRk. Simultaneously, the instruction extracts 32 bits out of 40-bit data in which "ACCGi+1" and "ACCi+1" are concatenated, beginning at the bit position specified for the shift count, performs signed saturating, and then stores the result in "FRk+1". ACCi (ACCi+1) data extracts 32 bits in 40-bit data (in which ACCGi (ACCGi+1) and ACCi (ACCi+1) are concatenated in this order), beginning at the specified bit position, with the MSB (left) side of 40-bit data allocated to bit 0 followed by "bit 1 to bit 31" allocated rightwards. It is not possible to specify extraction of 32 or larger bits. Also, a negative value can be specified for the shift count; when a negative value is specified, the bit position indicating the negative value is set to a sign. The shift count is specified for 6 bits of #s6.
When the result overflows, the result obtained after signed saturating is stored in the register, and `msr.ovf` is set to 1.
The register numbers of the ACCi and FRk registers must be set to even numbers previously by software; when the register number of the ACCi register is set to an odd number, mp_exception (acc_not_aligned) is detected. When the register number of the FRk register is set to an odd number, `mp`_exception (register_not_aligned) is detected.
For exceptions detected by this instruction, exception handling can be started by executing the MTRAP instruction.

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned,
acc_not_aligned, overflow)

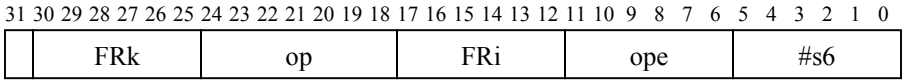
1.10.29. Expand (Halfword)

Ope-code	op	ope	Operation
MEXPDHW	1111011	110010	Expand Halfword to Word
MEXPDHD	1111011	110011	Expand Halfword to Double-Word

Category

Media

Instruction Format (Media instruction)



Assembler Syntax

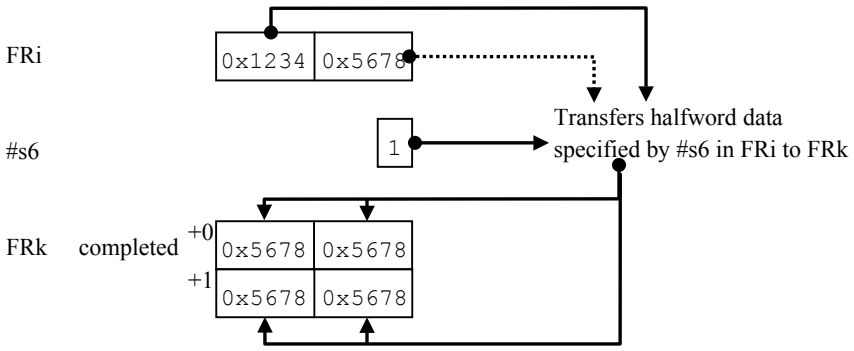
MEXPDHW FRi, #s6, FRk
MEXPDHD FRi, #s6, FRk

Description

The MEXPDHW instruction copies the halfword data of FRi indicated by the LSB of #s6 to FRk$_{hi}$ and FRk$_{lo}$.
The MEXPDHD instruction copies the halfword data of FRi indicated by the LSB of #s6 to FRk$_{hi}$, FRk$_{lo}$, FRk+1$_{hi}$ and FRk+1$_{lo}$.
The LSB of #s6 indicates that 0 is high-order halfword and 1 is low-order halfword.
It is necessary to set the register number of FRk of the MEXPDHD instruction as an even number with software. If the register number of FRk is an odd number, the mp_exception (register_not_aligned) occurs.
The processing exception detected by the instruction is initiated by executing MTRAP instruction.
("FRk$_{hi}$" indicates the high-order 16 bits of FRk and "FRk$_{lo}$" indicates the low-order 16 bits of FRk.)

Operation example

MEXPDHD FRi, #s6, FRk



Registers altered (except destination register)

none

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned)

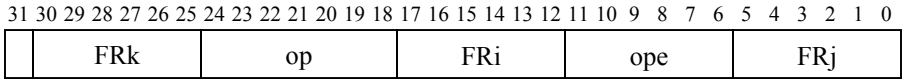
1.10.30. Pack/Unpack (Halfword)

Ope-code	op	ope	Operation
MPACKH	1111011	110100	Pack Halfword to Word
MUNPACKH	1111011	110101	Unpack Word to Halfword

Category

Media

Instruction Format (Media instruction)



Assembler Syntax

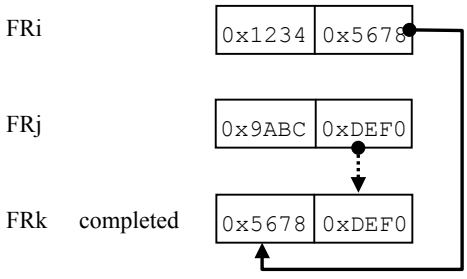
```
MPACKH      FRi,FRj,FRk
MUNPACKH    FRi,FRk
```

Description

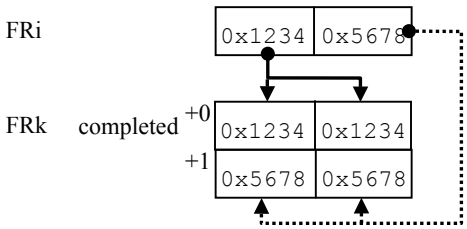
The MPACKH instruction writes FRi_{lo} in FRk_{hi} and writes FRj_{lo} in FRk_{lo}.
The MUNPACKH instruction copies the contents of FRi_{hi} to FRk_{hi} and FRk_{lo}, and copies the contents of FRi_{lo} to $FRk+1_{hi}$ and $FRk+1_{lo}$.
It is necessary to set the register number of FRk as an even number with software. If the register number of FRk in the MUNPACKH instruction is an odd number, the mp_exception (register_not_aligned) occurs.
The processing exception detected by the instruction is initiated by executing MTRAP instruction.
("FRk_{hi}" indicates the high-order 16 bits of FRk and "FRk_{lo}" indicates the low-order 16 bits of FRk.)

Operation example

MPACKH FRi,FRj,FRk



MUNPACKH FRi,FRk



Registers altered (except destination register)

none

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned)

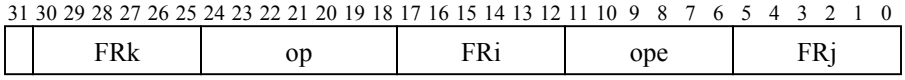
1.10.31. Pack (Halfword Dual)

Ope-code	op	ope	Operation
MDPACKH	1111011	110110	Dual Pack Halfword to Word

Category

Media

Instruction Format (Media instruction)



Assembler Syntax

MDPACKH FRi, FRj, FRk

Description

The MDPACKH instruction writes FRi_{lo} in FRk_{hi} and FRj_{lo} in FRk_{lo} and also writes $FRi+1_{lo}$ in $FRk+1_{hi}$ and $FRj+1_{lo}$ in $FRk+1_{lo}$

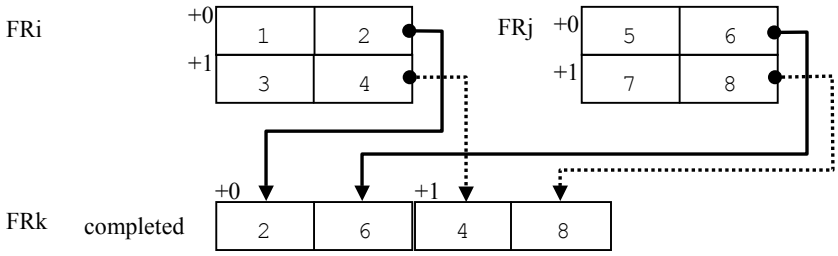
It is necessary to set the register number of FRi, FRj and FRk of the MDPACKH instruction as an even number with software. If the register number of FRi, FRj or FRk in the MDPACKH instruction is an odd number, the mp_exception (register_not_aligned) occurs.

The processing exception detected by the instruction is initiated by executing MTRAP instruction.

("FRi_{hi}" indicates the high-order 16 bits of FRi, and "FRi_{lo}" indicates the low-order 16 bits of FRi.)

Operation example

MDPACKH FRi, FRj, FRk



Registers altered (except destination register)

none

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned)

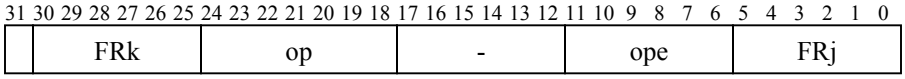
1.10.32. Convert Byte to/from Halfword

Ope-code	op	ope	Operation
MBTOH	1111011	111000	Byte To Halfword
MHTOB	1111011	111001	Halfword To Byte

Category

Media

Instruction Format (Media instruction)



Assembler Syntax

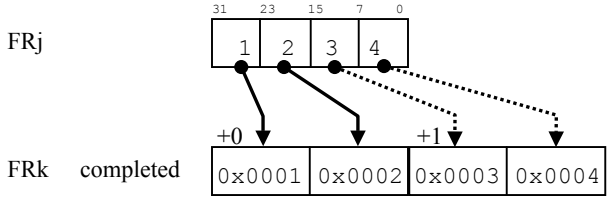
MBTOH FRj, FRk
MHTOB FRj, FRk

Description

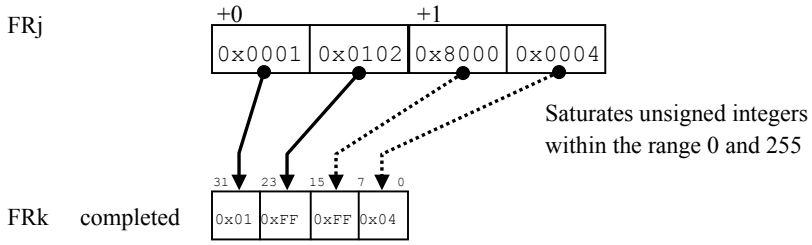
The MBTOH instruction converts the unsigned byte data in FRj3 to the unsigned halfword data in FRkhi, and converts the unsigned byte data in FRj2 to the unsigned halfword data in FRklo, and converts the unsigned byte data in FRj1 to the unsigned halfword data in FRk+1hi, and converts the unsigned byte data in FRj0 to the unsigned halfword data in FRk+1lo.
The MHTOB instruction converts the unsigned halfword data in FRjhi to the unsigned byte data in FRk3 after unsigned saturation processing with the maximum number of unsigned byte (0xff), and converts the unsigned halfword data in FRjlo to the unsigned byte data in FRk2 after unsigned saturation processing with the maximum number of unsigned byte (0xff), and converts the unsigned halfword data in FRj+1hi to the unsigned byte data in FRk1 after unsigned saturation processing with the maximum number of unsigned byte (0xff), and converts the unsigned halfword data in FRj+1lo to the unsigned byte data in FRk0 after unsigned saturation processing with the maximum number of unsigned byte (0xff).
It is necessary to set the register number of FRk of the MBTOH instruction and the register number of FRj of the MHTOB instruction in an even number with software. If the register number of FRk of the MBTOH instruction or the register number of FRj of the MHTOB instruction is an odd number, the mp_exception (register_not_aligned) occurs.
The processing exception detected by the instruction is initiated by executing MTRAP instruction.
The side small number of FRk* specify the position of byte in the register. The relation of the position of byte with the position of bit is below.
FRk3 = FRk (31 bits-24 bits), FRk2 = FRk (23 bits-16 bits)
FRk1 = FRk (15 bits-8 bits), FRk0 = FRk (7 bits-0 bit)

Operation example

MBTOH FRj, FRk



MHTOB FRj, FRk



Registers altered (except destination register)

none

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned)

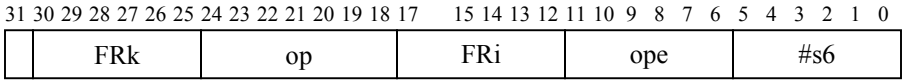
1.10.33. Media Bit Concatenate (Halfword Dual) Instruction

Ope code	op	ope	Operation
MCPLHI	1111000	001100	Dual Coupling Half Word Data (Immediate)

Category

Media

Instruction format (Media instruction)



Assembler description

MCPLHI FRi,#s6,FRk

Description

The media bit concatenate (Halfword Dual) instruction concatenates FRihi[15-n:0] and FRi+1hi[n-1:0] with allocating Fri+1[n-1:0] to the LSB side and then stores the result of this concatenation in Frkhi on condition that “n” represents the bit counts specified for lower 4 bits of #s6. Simultaneously, the instruction concatenates FRilo[15-n:0] and FRi+1lo[n-1:0] with allocating Fri+1lo[n-1:0] to the LSB side and then stores the result of this concatenation in FRklo.
The register number of the FRi register must be set to an even number previously by software; when it is set to an odd number, mp_exception (register_not_aligned) is detected.
For exceptions detected by this instruction, exception handling can be started by executing the MTRAP instruction.

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned)

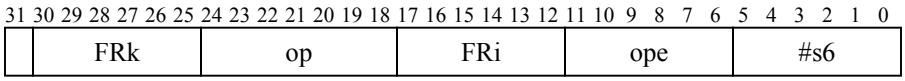
1.10.34. Media Bit Concatenate (Word Dual) Instruction

Ope code	op	ope	Operation
MCPLI	1111000	001101	Dual Coupling Word Data (Immediate)

Category

Media

Instruction format (Media instruction)



Assembler description

MCPLI FRi,#s6,FRk

Description

The media bit concatenate (Word Dual) instruction concatenates FRi[31-n:0] and FRi+1[n-1:0] with allocating Fri+1[n-1:0] to the LSB side and then stores the result of this concatenation in FRk on condition that “n” represents the bit counts specified for lower 5 bits of #s6.
The register number of the FRi register must be set to an even number previously by software; when it is an odd number, mp_exception (register_not_aligned) is detected.
For exceptions detected by this instruction, exception handling can be started by executing the MTRAP instruction.

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned)

Suspension exceptions to be started

None

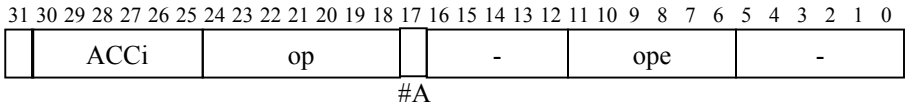
1.10.35. Clear Accumulator

Ope-code	op	ope	Operation
MCLRACC	1111011	111011	Clear Accumulator

Category

Media

Instruction Format (Media instruction)



Assembler Syntax

MCLRACC ACCi, #A

Description

The MCLRACC instruction clear the contents of ACCi and ACCGi, and set to zero.
If #A bit is “1” and ACCi is 0, all of ACC and all of ACCG are cleared.
If the indicated ACC doesn’t exist, this instruction behaves as NOP.

Registers altered (except destination register)

none

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception)

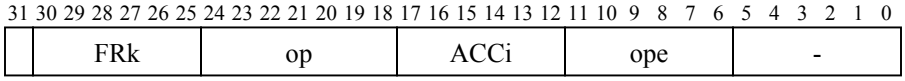
1.10.36. Read/Write Accumulator

Ope-code	op	ope	Operation
MRDACC	1111011	111100	Read Accumulator
MWTACC	1111011	111101	Write Accumulator
MRDACCG	1111011	111110	Read Accumulator Guard
MWTACCG	1111011	111111	Write Accumulator Guard

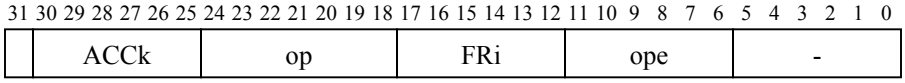
Category

Media

Instruction Format (Media instruction)



Instruction Format (Media instruction)



Assembler Syntax

MRDACC ACCi, FRk
MWTACC FRi, ACCk
MRDACCG ACCGi, FRk
MWTACCG FRi, ACCGk

Description

The MRDACC instruction copies the contents of ACCi to FRk.
The MWTACC instruction copies the contents of FRi to ACCk.
The MRDACCG instruction copies the contents of ACCGi to low-order 8-bit of FRk and zero to higher-order bits.
The MWTACCG instruction copies the contents of low-order 8-bit of FRi to ACCGk.
The processing exception detected by the instruction is initiated by executing MTRAP instruction.

Registers altered (except destination register)

none

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception)

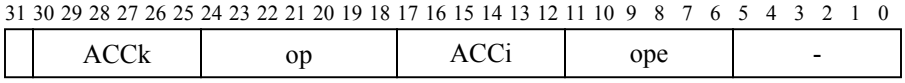
1.10.37. Media Accumulator Addition Instruction

Ope code	op	ope	Operation
MADDACCS	1111000	000100	Add Signed Accumulator with Saturation

Category

Media

Instruction format (Media accumulator instruction)



Assembler description

MADDACCS ACCi,ACCk

Description

The MADDACCS instruction adds signed 40-bit data in which ACCGi and ACCi are concatenated and signed 40-bit data in which “ACCGi+1” and “ACCi+1” are concatenated, and then stores the result obtained from this addition in the 40-bit register in which ACCGk and ACCk are concatenated. When the result overflows, the result obtained after signed saturating is stored in the register, and msr.ovf is set to 1.
The register number of the ACCi register must be set to a multiple of 2 previously by software; otherwise, mp_exception (acc_not_aligned) is detected.
For exceptions detected by this instruction, exception handling can be started by executing the MTRAP instruction.

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned,
acc_not_aligned , overflow)

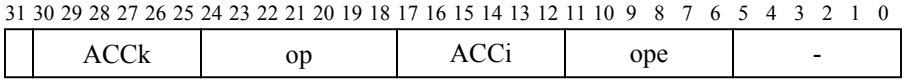
1.10.38. Media Accumulator Subtraction Instruction

Ope code	op	ope	Operation
MSUBACCS	1111000	000101	Subtract Signed Accumulator with Saturation

Category

Media

Instruction format (Media accumulator instruction)



Assembler description

MSUBACCS ACCi,ACCk

Description

The MSUBACCS instruction subtracts signed 40-bit data in which “ACCGi+1” and “ACCi+1” are concatenated, from signed 40-bit data in which ACCGi and ACCi are concatenated, and then stores the result obtained from this subtraction in the 40-bit register in which ACCGk and ACCk are concatenated. When the result overflows, the result obtained after signed saturating is stored in the register, and msr.ovf is set to 1.
The register number of the ACCi register must be set to a multiple of 2 previously by software; otherwise, mp_exception (acc_not_aligned) is detected.
For exceptions detected by this instruction, exception handling can be started by executing the MTRAP instruction.

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned,
acc_not_aligned , overflow)

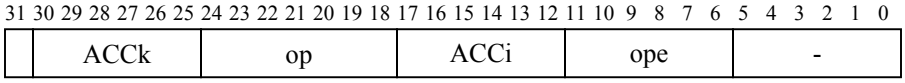
1.10.39. Media Dual Accumulator Addition Instruction

Ope code	op	ope	Operation
MDADDACCS	1111000	000110	Dual Add Signed Accumulator with Saturation

Category

Media

Instruction format (Media accumulator instruction)



Assembler description

MDADDACCS ACCi,ACCk

Description

The MDADDACCS instruction adds signed 40-bit data in which ACCGi and ACCi are concatenated and signed 40-bit data in which “ACCGi+1” and “ACCi+1” are concatenated, and then stores the result obtained from this addition in the 40-bit register in which ACCGk and ACCk are concatenated. Simultaneously, the instruction adds signed 40-bit data in which “ACCGi+2” and “ACCi+2” are concatenated and signed 40-bit data in which “ACCGi+3” and “ACCi+3” are concatenated, and then stores the result obtained from this addition in the 40-bit register in which “ACCGk+1” and “ACCk+1” are concatenated. When the result overflows, the result obtained after signed saturating is stored in the register, and msr.ovf is set to 1.
The register number of the ACCi register must be set to a multiple of 4 previously by software; otherwise, mp_exception (acc_not_aligned) is detected.
The register number of the ACCk register must be set to a multiple of 2 previously by software; otherwise, mp_exception (acc_not_aligned) is detected.
For exceptions detected by this instruction, exception handling can be started by executing the MTRAP instruction.

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned,
acc_not_aligned , overflow)

1.10.40. Media Dual Accumulator Subtraction Instruction

Ope code	op	ope	Operation
MDSUBACCS	1111000	000111	Dual Subtract Signed Accumulator with Saturation

Category

Media

Instruction format (Media accumulator instruction)

31	30 29 28 27	26 25 24 23	22 21 20 19	18 17 16 15	14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
	ACCk	op	ACCi	ope	-	

Assembler description

MDSUBACCS ACCi,ACCk

Description

The MDSUBACCS instruction subtracts signed 40-bit data in which “ACCGi+1” and “ACCi+1” are concatenated, from signed 40-bit data in which “ACCGi” and “ACCi” are concatenated, and then stores the result obtained from this subtraction in the 40-bit register in which ACCGk and ACCk are concatenated. Simultaneously, the instruction subtracts signed 40-bit data in which “ACCGi+3” and “ACCi+3” are concatenated, from signed 40-bit data in which “ACCGi+2” and “ACCi+2” are concatenated, and then stores the result obtained from this subtraction in the 40-bit register in which “ACCGk+1” and “ACCk+1” are concatenated. When the result overflows, the result obtained after signed saturating is stored in the register, and msr.ovf is set to 1.
The register number of the ACCi register must be set to a multiple of 4 previously by software; otherwise, mp_exception (acc_not_aligned) is detected.
The register number of the ACCk register must be set to a multiple of 2 previously by software; otherwise, mp_exception (acc_not_aligned) is detected.
For exceptions detected by this instruction, exception handling can be started by executing the MTRAP instruction.

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned,
acc_not_aligned , overflow)

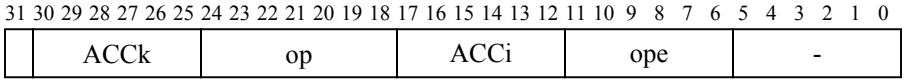
1.10.41. Media Accumulator Addition and Subtraction Instruction

Ope code	op	ope	Operation
MASACCS	1111000	001000	Add and Subtract Signed Accumulator with Saturation

Category

Media

Instruction format (Media accumulator instruction)



Assembler description

MASACCS ACCi,ACCk

Description

The MASACCS instruction adds signed 40-bit data in which ACCGi and ACCi are concatenated and signed 40-bit data in which “ACCGi+1” and “ACCi+1” are concatenated, and then stores the result obtained from this addition in the 40-bit register in which ACCGk and ACCk are concatenated. Simultaneously, the instruction subtracts signed 40-bit data in which “ACCGi+1” and “ACCi+1” are concatenated, from signed 40-bit data in which ACCGi and ACCi are concatenated, and then stores the result obtained from this subtraction in the 40-bit register in which “ACCGk+1” and “ACCk+1” are concatenated. When the result overflows, the result obtained after signed saturating is stored in the register, and msr.ovf is set to 1.
The register number of the ACCi register must be set to a multiple of 2 previously by software; otherwise, mp_exception (acc_not_aligned) is detected.
The register number of the ACCk register must be set to a multiple of 2 previously by software; otherwise, mp_exception (acc_not_aligned) is detected.
For exceptions detected by this instruction, exception handling can be started by executing the MTRAP instruction.

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned,
acc_not_aligned , overflow)

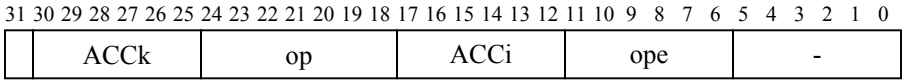
1.10.42. Media Dual Accumulator Addition and Subtraction Instruction

Ope code	op	ope	Operation
MDASACCS	1111000	001001	Dual Add and Subtract Signed Accumulator with Saturation

Category

Media

Instruction format (Media accumulator instruction)



Assembler description

MDASACCS ACCi,ACCk

Description

The MDASACCS instruction adds signed 40-bit data in which ACCGi and ACCi are concatenated and signed 40-bit data in which “ACCGi+1” and “ACCi+1” are concatenated, and then stores the result obtained from this addition in the 40-bit register in which ACCGk and ACCk are concatenated. Simultaneously, the instruction subtracts signed 40-bit data in which “ACCGi+1” and “ACCi+1” are concatenated, from signed 40-bit data in which ACCGi and ACCi are concatenated, and then stores the result obtained from this subtraction in the 40-bit register in which “ACCGk+1” and “ACCk+1” are concatenated. Simultaneously, the instruction adds signed 40-bit data in which “ACCGi+2” and “ACCi+2” are concatenated and signed 40-bit data in which “ACCGi+3” and “ACCi+3” are concatenated, and then stores the result obtained from this addition in the 40-bit register in which “ACCGk+2” and “ACCk+2” are concatenated. Simultaneously, the instruction subtracts signed 40-bit data in which “ACCGi+3” and “ACCi+3” are concatenated, from signed 40-bit data in which “ACCGi+2” and “ACCi+2” are concatenated, and then stores the result obtained from this subtraction in the 40-bit register in which “ACCGk+3” and “ACCk+3” are concatenated. When the result overflows, the result obtained after signed saturating is stored in the register, and msr.ovf is set to 1.

The register number of the ACCi register must be set to a multiple of 4 previously by software; otherwise, mp_exception (acc_not_aligned) is detected.

The register number of the ACCk register must be set to a multiple of 4 previously by software; otherwise, mp_exception (acc_not_aligned) is detected.

For exceptions detected by this instruction, exception handling can be started by executing the MTRAP instruction.

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned,
acc_not_aligned , overflow)

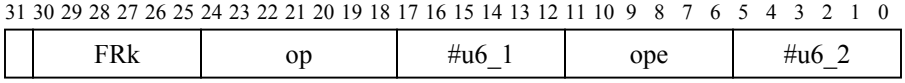
1.10.43. Media SETHI/SETLO (Halfword) Instruction

Ope code	op	ope	Operation
MHSETLOS	1111000	100000	Set Lower signed 12 bits
MHSETLOH	1111000	100001	Set Upper 5 bits
MHSETHIS	1111000	100010	Set Upper signed 12bits
MHSETHIH	1111000	100011	Set Upper 5 bits
MHDSETS	1111000	100100	Dual Set Half Word signed 12 bits
MHDSETH	1111000	100101	Dual Set Upper 5 bits of Half Word

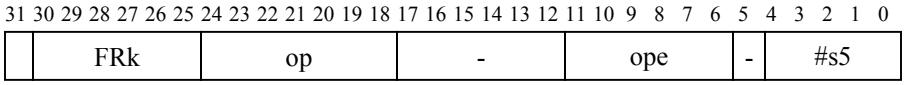
Category

Media

Instruction format (Media instruction (set immediate 12))



Instruction format (Media instruction (set immediate 5))



Assembler description

MHSETLOS	#u12,FRk (#u12[11:6]=#u6_1, #u12[5:0]=#u6_2)
MHSETHIS	#u12,FRk (#u12[11:6]=#u6_1, #u12[5:0]=#u6_2)
MHDSETS	#u12,FRk (#u12[11:6]=#u6_1, #u12[5:0]=#u6_2)
MHSETLOH	#s5,FRk
MHSETHIH	#s5,FRk
MHDSETH	#s5,FRk

Description

The MHSETLOS instruction replaces the lower 12 bits of FRklo with the value of #u12, and simultaneously, replaces each of the upper 4 bits of FRklo with MSB of #u12.
The MHSETHIS instruction replaces the lower 12 bits of FRkhi with the value of #u12, and simultaneously, replaces each of the upper 4 bits of FRkhi with MSB of #u12.
The MHDSETS instruction replaces the lower 12 bits of FRkhi with the value of #u12. simultaneously, replaces each of the upper 4 bits of FRkhi with MSB of #u12. simultaneously, replaces the lower 12 bits of FRklo with the value of #u12. simultaneously, replaces each of the upper 4 bits of FRklo with MSB of #u12.
The MHSETLOH instruction replaces the upper 5 bits of FRklo with the value of #s5.
The MHSETHIH instruction replaces the upper 5 bits of FRkhi with the value of #s5.
The MHDSETH instruction replaces the upper 5 bits of FRkhi with the value of #s5, and simultaneously, replaces the upper 5 bits of FRklo with the value of #s5.
For exceptions detected by this instruction, exception handling can be started by executing the MTRAP instruction.

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception)

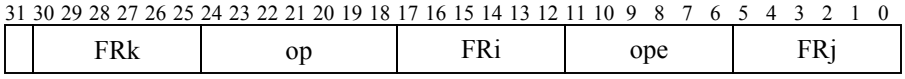
1.10.44. Media Quad Low Clear (Halfword Quad) Instruction (M-Type Instruction. This instruction is available for MB93451.)

Ope code	op	ope	Operation
MQLCLRHS	1111000	010000	Quad Clear Lower Value Signed Halfword

Category

Media

Instruction format (Extended Media instruction)



Assembler description

MQLCLRHS FRi,FRj,FRk

Description

The MQLCLRHS instruction stores A in C when B is positive and |A|>|B|, otherwise it stores 0 into C. This instruction also stores –A (if A=0x8000, stores 0x7FFF) into c when B is negative and |A|>|B|, otherwise it stores 0 into C.
However, A,B,C are signed halfword integers and the combinations are as follows:
{A, B, C}={FRihi, FRjhi, FRkhi}, {FRilo, FRjlo, FRklo}, {FRi+1hi, FRj+1hi, FRk+1hi}, {FRi+1lo, FRj+1lo, FRk+1lo }
The register numbers of FRi, FRj and FRk need to be set to even with software. When the register numbers of FRi, FRj, FRk are odd, mp_exception(register_not_aligned) are detected.
For exceptions detected by this instruction, exception handling can be started by executing the MTRAP instruction.

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception(unimplemented_exception, register_not_aligned)

Started Pending Exceptions

None

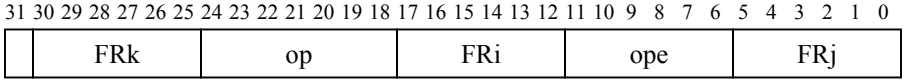
1.10.45. Media Quad Scope Limitation (Halfword Quad) Instruction (M-Type Instruction. This instruction is available for MB93451.)

Ope code	op	ope	Operation
MQLMTHS	1111000	010100	Quad Set Limits Signed Halfword

Category

Media

Instruction format (Extended Media instruction)



Assembler description

MQLMTHS FRi,FRj,FRk

Description

The MQLMTHS stores A into C when -|B|<A<|B|, stores –B (if B=0x8000, stores 0x7FFF) into C when A≦-|B| and stores B into C when |B|<=A.
However, A,B,C are signed halfword integers and the combinations are as follows:
{A, B, C}={FRihi, FRjhi, FRkhi}, {FRilo, FRjlo, FRklo}, {FRi+1hi, FRj+1hi, FRk+1hi}, {FRi+1lo, FRj+1lo, FRk+1lo }
The register numbers of FRi, FRj and FRk need to be set to even with software. When the register numbers of FRi, FRj and FRk are odd, mp_exception(register_not_aligned) are detected.
For exceptions detected by this instruction, exception handling can be started by executing the MTRAP instruction.

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception(unimplemented_exception, register_not_aligned)

Started Pending Exceptions

None

1.10.46. Media Quad Shift (Halfword Quad) instruction (M-Type instruction. These instructions are available for MB93451.)

Ope code	op	ope	Operation
MQSLLHI	1111000	010001	Quad Shift Lest Logical Halfword (Immediate)
MQSRAHI	1111000	010011	Quad Shift Right Arithmetic Logical Halfword (Immediate)

Category

Media

Instruction format (Extended Media instruction)

31	30 29 28 27	26 25 24 23	22 21 20 19	18 17 16 15	14 13 12 11	10 9 8 7 6 5 4 3 2 1 0
	FRk	op	FRi	ope	#s6	

Assembler description

MQSLLHI FRi,#s6,FRk
MQSRAHI FRi,#s6,FRk

Description

The Media Quad Shift instruction shifts FRihi by the number of bits specified by the shift counter and stores the result into FRkhi. It also shifts FRilo and stores the result into FRklo, shifts FRi+1hi and stores the result into FRk+1hi and shifts Ri+1lo and stores the result into FRk+1lo. The shift count is specified by the lower 4 bits of #s6. The MQSRAHI instruction takes the value rounded by the rounding method specified in MSR0 as the result.
The MQSLLHI instruction shifts FRihi, FRilo, FRi+1hi and FRi+1lo to the left and makes the bits that became empty by the shift 0.
The MQSRAHI instruction shifts FRihi, FRilo, FRi+1hi and FRi+1lo to the right and fills the bits that became empty the shift with the highest bits of FRihi, FRilo, FRi+1hi and FRi+1lo respectively.
The register numbers of Fri and FRk need to be set even with software. When the register numbers of FRi and FRk are odd, mp_exception(register_not_aligned) are detected.
For exceptions detected by this instruction, exception handling can be started by executing the MTRAP instruction.
(FRihi is the higher 16 bits of FRi. FRilo is the lower 16 bits of FRi. FRi+1hi is the higher 16 bits of FRi+1. FRi+1lo is the lower 16 bits of FRi+1. FRk is the same as this.)

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception(unimplemented_exception, register_not_aligned)

Started Pending Exceptions

None

1.11. Conditional Integer Instructions

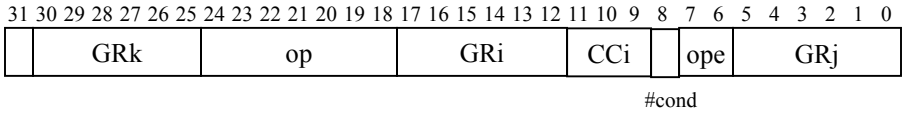
1.11.1. Add / Subtract / Multiply / Divide

Ope-code	op	ope	Operation
CADD	1011000	00	Conditional Add
CSUB	1011000	01	Conditional Subtract
CSMUL	1011000	10	Conditional Signed Integer Multiply
CSDIV	1011000	11	Conditional Signed Integer Divide

Category

Integer, Conditional

Instruction Format (Conditional instruction)



Assembler Syntax

CADD	GRi, GRj, GRk, CCi, #cond
CSUB	GRi, GRj, GRk, CCi, #cond
CSMUL	GRi, GRj, GRk, CCi, #cond
CSDIV	GRi, GRj, GRk, CCi, #cond

Description

The Conditional Integer Arithmetic Instruction operates GRi and GRj in arithmetic. When the condition specified by the #cond field is equal to the condition shown with CCi, the instruction writes the result in GRk. Otherwise, it doesn't change GRk.
The CADD instruction operates "GRi+GRj", when the condition specified by the #cond field is equal to the condition shown with CCi.
The CSUB instruction operates "GRi-GRj", when the condition specified by the #cond field is equal to the condition shown with CCi.
The CSMUL instruction operates "GRi*GRj", when the condition specified by the #cond field is equal to the condition shown with CCi.
The CSDIV instruction operates "GRi/GRj", when the condition specified by the #cond field is equal to the condition shown with CCi.
Table 25 shows the values of #cond field.

Registers altered (except destination register)

none

Occurrence Exceptions

division_exception
register_exception (unimplement_exception)

Detected Exceptions

none

Table 25 Values of #cond field

value	meaning
0	False
1	True

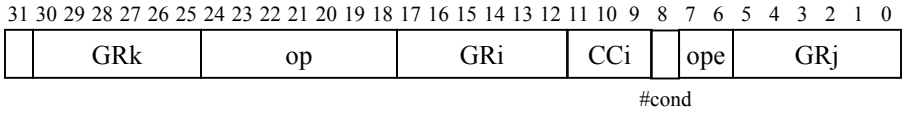
1.11.2. Add, Subtract and Multiply with setting ICC / Divide unsigned integer

Ope-code	op	ope	Operation
CADDcc	1011001	00	Conditional Add and ICC setting
CSUBcc	1011001	01	Conditional Subtract ICC setting
CSMULcc	1011001	10	Conditional Signed Integer Multiply ICC setting
CUDIV	1011001	11	Conditional Unsigned Integer Divide

Category

Integer, Conditional

Instruction Format (Conditional instruction)



Assembler Syntax

CADDcc GRi, GRj, GRk, CCi, #cond
CSUBcc GRi, GRj, GRk, CCi, #cond
CSMULcc GRi, GRj, GRk, CCi, #cond
CUDIV GRi, GRj, GRk, CCi, #cond

Description

The Conditional Integer Arithmetic Instruction operates GRi and GRj in arithmetic. When the condition specified by the #cond field is equal to the condition shown with CCi, the instruction writes the result in GRk. Otherwise, it doesn't change GRk.
The CADDcc instruction operates "GRi+GRj", when the condition specified by the #cond field is equal to the condition shown with CCi.
The CSUBcc instruction operates "GRi-GRj", when the condition specified by the #cond field is equal to the condition shown with CCi.
The CSMULcc instruction operates "GRi*GRj", when the condition specified by the #cond field is equal to the condition shown with CCi.
The CUDIV instruction operates "GRi/GRj", when the condition specified by the #cond field is equal to the condition shown with CCi.
The CADDcc, CSUBcc, and CSMULcc instructions change the integer condition code (ICC) specified by the low-order 2-bit of CCi field.
Table 26 shows the values of #cond field.

Registers altered (except destination register)

ICC … instructions with “cc” only

N	Z	V	C
O	O	O	O

Occurrence Exceptions

division_exception
register_exception (unimplement_exception)

Detected Exceptions

none

Table 26 Values of #cond field

Value	meaning
0	False
1	True

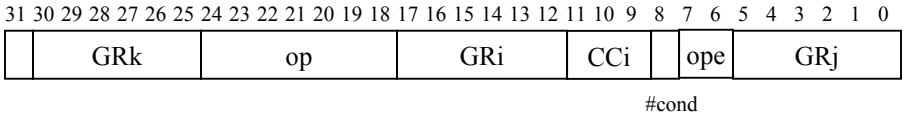
1.11.3. Logical Operations

Ope-code	op	ope	Operation
CAND	1011010	00	Conditional AND
COR	1011010	01	Conditional OR
CXOR	1011010	10	Conditional XOR
CNOT	1011010	11	Conditional NOT

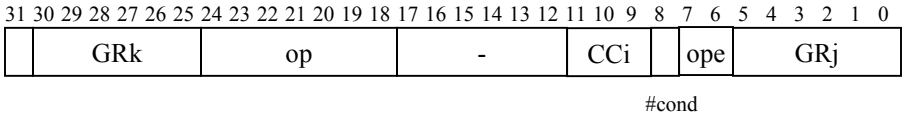
Category

Integer, Conditional

Instruction Format (Conditional instruction)



Instruction Format (Conditional Not-Operation instruction)



Assembler Syntax

CAND	GRi, GRj, GRk, CCi, #cond
COR	GRi, GRj, GRk, CCi, #cond
CXOR	GRi, GRj, GRk, CCi, #cond
CNOT	GRj, GRk, CCi, #cond

Description

The Conditional Integer Logical Instruction operates GRi and GRj in logical. When the condition specified by the #cond field is equal to the condition shown with CCi, the instruction writes the result in GRk. Otherwise, it doesn't change GRk.
The CAND instruction operates "GRi and GRj", when the condition specified by the #cond field is equal to the condition shown with CCi.
The COR instruction operates "GRi or GRj", when the condition specified by the #cond field is equal to the condition shown with CCi.
The CXOR instruction operates "GRi xor GRj", when the condition specified by the #cond field is equal to the condition shown with CCi.
The CNOT instruction operates "not GRj", when the condition specified by the #cond field is equal to the condition shown with CCi.
Table 27 shows the values of #cond field.

Registers altered (except destination register)

none

Occurrence Exceptions

register_exception (unimplement_exception)

Detected Exceptions

none

Table 27 Values of #cond field

value	meaning
0	False
1	True

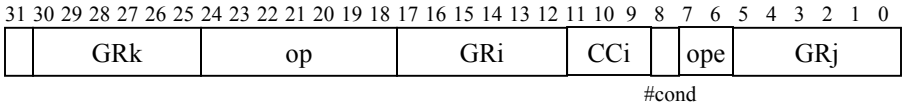
1.11.4. Logical Operations with setting ICC

Ope-code	op	ope	Operation
CANDcc	1011011	00	Conditional AND and ICC setting
CORcc	1011011	01	Conditional OR and ICC setting
CXORcc	1011011	10	Conditional XOR and ICC setting

Category

Integer, Conditional

Instruction Format (Conditional instruction)



Assembler Syntax

CANDcc GRi, GRj, GRk, CCi, #cond
CORcc GRi, GRj, GRk, CCi, #cond
CXORcc GRi, GRj, GRk, CCi, #cond

Description

The Conditional Integer Logical Instruction operates GRi and GRj in logical. When the condition specified by the #cond field is equal to the condition shown with CCi, the instruction writes the result in GRk. Otherwise, it doesn't change GRk.
The CANDcc instruction operates "GRi and GRj", when the condition specified by the #cond field is equal to the condition shown with CCi.
The CORcc instruction operates "GRi or GRj", when the condition specified by the #cond field is equal to the condition shown with CCi.
The CXORcc instruction operates "GRi xor GRj", when the condition specified by the #cond field is equal to the condition shown with CCi.
The CANDcc, CORcc, and CXORcc instructions change the integer condition code (ICC) specified by the low-order 2-bit of CCi field.
Table 28 shows the values of #cond field.

Registers altered (except destination register)

ICC … instructions with "cc" only

N	Z	V	C
O	O	X	X

Occurrence Exceptions

register_exception (unimplement_exception)

Detected Exceptions

none

Table 28 Values of #cond field

value	meaning
0	False
1	True

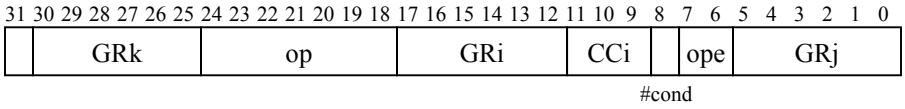
1.11.5. Shift

Ope-code	op	ope	Operation
CSLL	1011100	00	Conditional Shift Left Logical
CSRL	1011100	01	Conditional Shift Right Logical
CSRA	1011100	10	Conditional Shift Right Arithmetic

Category

Integer, Conditional

Instruction Format (Conditional instruction)



Assembler Syntax

CSLL GRi, GRj, GRk, CCi, #cond
CSRL GRi, GRj, GRk, CCi, #cond
CSRA GRi, GRj, GRk, CCi, #cond

Description

The Conditional Integer Shift Instructions shift GRi by the number of bits implied by the shift-count and write the result in GRk. The shift-count is specified by the low-order 5 bits of GRj. When the condition specified by the #cond field is equal to the condition shown with CCi, the instruction writes the result in GRk. Otherwise, it doesn't change GRk.
The CSLL instruction shifts GRi to the left, replacing the vacated positions with zero, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CSRL instruction shifts GRi to the right, replacing the vacated positions with zero, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CSRA instruction shifts GRi to the right, replacing the vacated positions with the highest bit of GRi, when the condition specified by the #cond field is equal to the condition shown with CCi.
Table 29 shows the values of #cond field.

Registers altered (except destination register)

none

Occurrence Exceptions

register_exception (unimplement_exception)

Detected Exceptions

none

Table 29 Values of #cond field

value	meaning
0	False
1	True

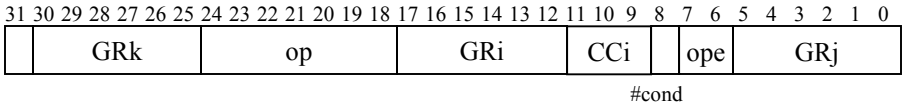
1.11.6. Shift with setting ICC

Ope-code	op	ope	Operation
CSLLcc	1011101	00	Conditional Shift Left Logical and ICC setting
CSRLcc	1011101	01	Conditional Shift Right Logical and ICC setting
CSRAcc	1011101	10	Conditional Shift Right Arithmetic and ICC setting

Category

Integer, Conditional

Instruction Format (Conditional instruction)



Assembler Syntax

CSLLcc GRi, GRj, GRk, CCi, #cond
CSRLcc GRi, GRj, GRk, CCi, #cond
CSRAcc GRi, GRj, GRk, CCi, #cond

Description

The Conditional Integer Shift Instructions shift GRi by the number of bits implied by the shift-count and write the result in GRk. The shift-count is specified by the low-order 5 bits of GRj. When the condition specified by the #cond field is equal to the condition shown with CCi, the instruction writes the result in GRk. Otherwise, it doesn't change GRk.
The CSLLcc instruction shifts GRi to the left, replacing the vacated positions with zero, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CSRLcc instruction shifts GRi to the right, replacing the vacated positions with zero, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CSRAcc instruction shifts GRi to the right, replacing the vacated positions with the highest bit of GRi, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CSLLcc, CSRLcc, and CSRAcc instructions change the integer condition code (ICC) specified by the low-order 2-bit of the CCi field.
Table 30 shows the values of #cond field.

Registers altered (except destination register)

ICC … instructions with "cc" only

N	Z	V	C
O	O	X	O

Occurrence Exceptions

register_exception (unimplement_exception)

Detected Exceptions

none

Table 30 Values of #cond field

value	meaning
0	False
1	True

1.12. Conditional Load/Store Instructions

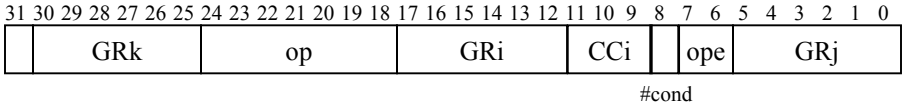
1.12.1. Load GR

Ope-code	op	ope	Operation
CLDSB	1011110	00	Conditional Load Signed Byte
CLDUB	1011110	01	Conditional Load Unsigned Byte
CLDSH	1011110	10	Conditional Load Signed Halfword
CLDUH	1011110	11	Conditional Load Unsigned Halfword
CLD	1011111	00	Conditional Load Word
CLDD	1011111	01	Conditional Load Double
CLDSBU	1100001	00	Conditional Load Signed Byte with Update Index
CLDUBU	1100001	01	Conditional Load Unsigned Byte Update Index
CLDSHU	1100001	10	Conditional Load Signed Halfword Update Index
CLDUHU	1100001	11	Conditional Load Unsigned Halfword Update Index
CLDU	1100010	00	Conditional Load Word Update Index
CLDDU	1100010	01	Conditional Load Double Update Index

Category

Integer, Conditional

Instruction Format (Conditional instruction)



Assembler Syntax

CLDSB	@(GRi, GRj), GRk, CCi, #cond
CLDUB	@(GRi, GRj), GRk, CCi, #cond
CLDSH	@(GRi, GRj), GRk, CCi, #cond
CLDUH	@(GRi, GRj), GRk, CCi, #cond
CLD	@(GRi, GRj), GRk, CCi, #cond
CLDD	@(GRi, GRj), GRk, CCi, #cond
CLDSBU	@(GRi, GRj), GRk, CCi, #cond
CLDUBU	@(GRi, GRj), GRk, CCi, #cond
CLDSHU	@(GRi, GRj), GRk, CCi, #cond
CLDUHU	@(GRi, GRj), GRk, CCi, #cond
CLDU	@(GRi, GRj), GRk, CCi, #cond
CLDDU	@(GRi, GRj), GRk, CCi, #cond

Description

The conditional integer Load instructions calculate “GRi + GRj” as an effective address. When the condition specified by the #cond field is equal to the condition shown with CCi, the instructions copy data from memory to GRk. Otherwise, they don’t change GRk.
As the description of each instruction, refer to 1.3.1 Load GR (page 19).
The CLDSB instruction operates as LDSB instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.

The CLDUB instruction operates as LDUB instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CLDSH instruction operates as LDSH instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CLDUH instruction operates as LDUH instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CLD instruction operates as LD instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CLDD instruction operates as LDD instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CLDSBU instruction operates as LDSBU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CLDUBU instruction operates as LDUBU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CLDSHU instruction operates as LDSHU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CLDUHU instruction operates as LDUHU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CLDU instruction operates as LDU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CLDDU instruction operates as LDDU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
Table 31 shows the values of #cond field.

Registers altered (except destination register)

GRi … Instruction with "update" form only

Occurrence Exceptions

mem_address_not_aligned (except byte Load instruction)
data_access_exception
data_access_MMU_miss
data_access_error
register_exception (unimplement_exception, register_not_aligned)

Detected Exceptions

none

Table 31 Values of #cond field

value	meaning
0	False
1	True

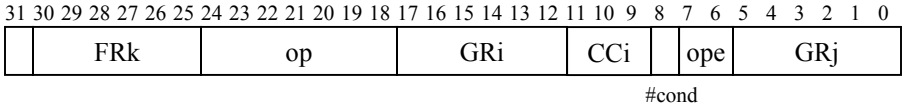
1.12.2. Load FR

Ope-code	op	ope	Operation
CLDBF	1100000	00	Conditional Load Byte FR register
CLDHF	1100000	01	Conditional Load Halfword FR register
CLDF	1100000	10	Conditional Load Word FR register
CLDDF	1100000	11	Conditional Load Double FR register
CLDBFU	1100011	00	Conditional Load Byte FR register with Update index
CLDHFU	1100011	01	Conditional Load Halfword FR register with Update index
CLDFU	1100011	10	Conditional Load Word FR register with Update index
CLDDFU	1100011	11	Conditional Load Double FR register with Update index

Category

Integer, Conditional

Instruction Format (Conditional instruction)



Assembler Syntax

CLDBF	@(GRi, GRj), FRk, CCi, #cond
CLDHF	@(GRi, GRj), FRk, CCi, #cond
CLDF	@(GRi, GRj), FRk, CCi, #cond
CLDDF	@(GRi, GRj), FRk, CCi, #cond
CLDBFU	@(GRi, GRj), FRk, CCi, #cond
CLDHFU	@(GRi, GRj), FRk, CCi, #cond
CLDFU	@(GRi, GRj), FRk, CCi, #cond
CLDDFU	@(GRi, GRj), FRk, CCi, #cond

Description

The conditional floating-point Load instructions calculate “GRi + GRj” as an effective address. When the condition specified by the #cond field is equal to the condition shown with CCi, the instructions copy data from memory to FRk. Otherwise, they don’t change FRk.
As the description of each instruction, refer to 1.3.2 Load FR (page 22).
The CLDBF instruction operates as LDBF instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CLDHF instruction operates as LDHF instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CLDF instruction operates as LDF instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CLDDF instruction operates as LDDF instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CLDBFU instruction operates as LDBFU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CLDHFU instruction operates as LDHFU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.

The CLDFU instruction operates as LDFU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CLDDFU instruction operates as LDDFU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
Table 32 shows the values of #cond field.

Registers altered (except destination register)

GRi … Instruction with "update" form only

Occurrence Exceptions

fp_disabled
mem_address_not_aligned (except byte Load instruction)
data_access_exception
data_access_MMU_miss
data_access_error
register_exception (unimplement_exception, register_not_aligned)

Detected Exceptions

none

Table 32 Values of #cond field

value	meaning
0	False
1	True

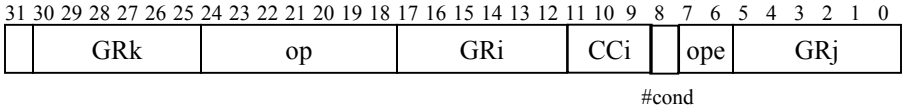
1.12.3. Store GR

Ope-code	op	ope	Operation
CSTB	1100100	00	Conditional Store Byte
CSTH	1100100	01	Conditional Store Halfword
CST	1100100	10	Conditional Store Word
CSTD	1100100	11	Conditional Store Double
CSTBU	1100111	00	Conditional Store Byte with Update Index
CSTHU	1100111	01	Conditional Store Halfword with Update Index
CSTU	1100111	10	Conditional Store Word with Update Index
CSTDU	1100111	11	Conditional Store Double with Update Index

Category

Integer, Conditional

Instruction Format (Conditional instruction)



Assembler Syntax

CSTB	GRk, @(GRi, GRj), CCi, #cond
CSTH	GRk, @(GRi, GRj), CCi, #cond
CST	GRk, @(GRi, GRj), CCi, #cond
CSTD	GRk, @(GRi, GRj), CCi, #cond
CSTBU	GRk, @(GRi, GRj), CCi, #cond
CSTHU	GRk, @(GRi, GRj), CCi, #cond
CSTU	GRk, @(GRi, GRj), CCi, #cond
CSTDU	GRk, @(GRi, GRj), CCi, #cond

Description

The conditional integer Store instructions calculate “GRi + GRj” as an effective address. When the condition specified by the #cond field is equal to the condition shown with CCi, the instructions copy data from GRk into memory. Otherwise, they don’t change memory.
As the description of each instruction, refer to 1.3.3 Store GR (page 24).
The CSTB instruction operates as STB instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CSTH instruction operates as STH instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CST instruction operates as ST instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CSTD instruction operates as STD instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CSTBU instruction operates as STBU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CSTHU instruction operates as STHU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.

The CSTU instruction operates as STU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CSTDU instruction operates as STDU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
Table 33 shows the values of #cond field.

Registers altered (except destination register)

GRi … Instruction with "update" form only

Occurrence Exceptions

mem_address_not_aligned (except byte Store)
data_access_exception
data_access_MMU_miss
data_access_error
data_store_error
register_exception (unimplement_exception, register_not_aligned)

Detected Exceptions

none

Table 33 Values of #cond field

value	meaning
0	False
1	True

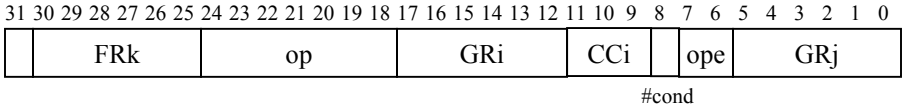
1.12.4. Store FR

Ope-code	op	ope	Operation
CSTBF	1100110	00	Conditional Store Byte FR register
CSTHF	1100110	01	Conditional Store Halfword FR register
CSTF	1100110	10	Conditional Store Word FR register
CSTDF	1100110	11	Conditional Store Double FR register
CSTBFU	1101000	00	Conditional Store Byte FR register with Update Index
CSTHFU	1101000	01	Conditional Store Halfword FR register with Update Index
CSTFU	1101000	10	Conditional Store Word FR register with Update Index
CSTDFU	1101000	11	Conditional Store Double FR register with Update Index

Category

Integer, Conditional

Instruction Format (Conditional instruction)



Assembler Syntax

CSTBF	FRk, @(GRi, GRj), CCi, #cond
CSTHF	FRk, @(GRi, GRj), CCi, #cond
CSTF	FRk, @(GRi, GRj), CCi, #cond
CSTDF	FRk, @(GRi, GRj), CCi, #cond
CSTBFU	FRk, @(GRi, GRj), CCi, #cond
CSTHFU	FRk, @(GRi, GRj), CCi, #cond
CSTFU	FRk, @(GRi, GRj), CCi, #cond
CSTDFU	FRk, @(GRi, GRj), CCi, #cond

Description

The conditional floating-point Store instructions calculate “GRi + GRj” as an effective address. When the condition specified by the #cond field is equal to the condition shown with CCi, the instructions copy data from FRk into memory. Otherwise, they don’t change memory.
As the description of each instruction, refer to 1.3.4 Store FR (page 26).
The CSTBF instruction operates as STBF instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CSTHF instruction operates as STHF instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CSTF instruction operates as STF instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CSTDF instruction operates as STDF instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CSTBFU instruction operates as STBFU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CSTHFU instruction operates as STHFU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.

The CSTFU instruction operates as STFU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CSTDFU instruction operates as STDFU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
Table 34 shows the values of #cond field.

Registers altered (except destination register)

GRi … Instruction with "update" form only

Occurrence Exceptions

fp_disabled
mem_address_not_aligned (except byte Store)
data_access_exception
data_access_MMU_miss
data_access_error
data_store_error
register_exception (unimplement_exception, register_not_aligned)

Detected Exceptions

none

Table 34 Values of #cond field

value	meaning
0	False
1	True

1.13. Conditional Data transfer Instructions

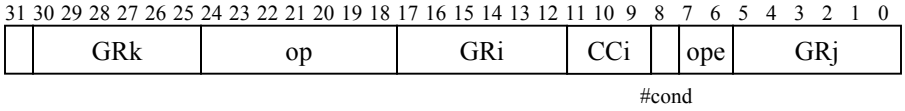
1.13.1. Swap

Ope-code	op	ope	Operation
CSWAP	1100101	10	Conditional Swap register with memory

Category

Control, Conditional

Instruction Format (Conditional instruction)



Assembler Syntax

CSWAP @(GRi, GRj), GRk, CCi, #cond

Description

The Conditional Swap Load-Store Instructions calculate “GRi + GRj” as an effective address. When the condition specified by the #cond field is equal to the condition shown with CCi, the instruction exchange GRk with the contents of the word addressed memory location. Otherwise, the instruction changes neither GRk nor memory.
The CSWAP instruction operates as SWAP instruction(page 28), when the condition specified by the #cond field is equal to the condition shown with CCi.
Table 35 shows the values of #cond field.

Registers altered (except destination register)

none

Occurrence Exceptions

mem_address_not_aligned
data_access_exception
data_access_MMU_miss
data_access_error
data_store_error
register_exception (unimplement_exception)

Detected Exceptions

none

Table 35 Values of #cond field

value	meaning
0	False
1	True

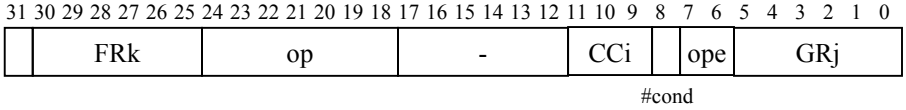
1.13.2. Move

Ope-code	op	ope	Operation
CMOVGF	1101001	00	Conditional Move GR to FR
CMOVGFD	1101001	01	Conditional Move GR to FR Double
CMOVFG	1101001	10	Conditional Move FR to GR
CMOVFGD	1101001	11	Conditional Move FR to GR Double

Category

Integer, Conditional

Instruction Format (Conditional instruction)



Assembler Syntax

CMOVGF GRj, FRk, CCi, #cond
CMOVGFD GRj, FRk, CCi, #cond
CMOVFG FRk, GRj, CCi, #cond
CMOVFGD FRk, GRj, CCi, #cond

Description

When the condition specified by the #cond field is equal to the condition shown with CCi, the conditional register transfer instructions copy data between GR and FR. Otherwise, the instructions do not copy data.
As the description of each instruction, refer to 1.4.2 Move (page 30).
The CMOVGF instruction operates as MOVGF instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CMOVGFD instruction operates as MOVGFD instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CMOVFG instruction operates as MOVFG instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CMOVFGD instruction operates as MOVFGD instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
Table 36 shows the values of #cond field.

Registers altered (except destination register)

none

Occurrence Exceptions

fp_disabled
register_exception (unimplement_exception, register_not_aligned)

Detected Exceptions

none

Table 36 Values of #cond field

value	meaning
0	False
1	True

1.14. Conditional Control transfer Instructions

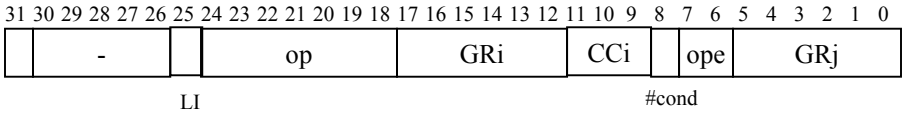
1.14.1. Jump and Link

Ope-code	op	ope	Operation
CJMPL	1101010	10	Conditional JUMP and Link

Category

Integer, Conditional

Instruction Format (Conditional instruction)



Assembler Syntax

CJMPL	@(GRi, GRj), CCi, #cond	(In case LI=0)
CCALLL	@(GRi, GRj), CCi, #cond	(In case LI=1)

Description

The CJMPL Instructions operates as JMPL instruction(page 43), when the condition specified by the #cond field is equal to the condition shown with CCi.
Table 37 shows the values of #cond field.
Even if the branch target address is not aligned on word boundary, the mem_address_not_aligned exception doesn't occur. In this case, the low-order bits below the word boundary in the branch target address is "0".

Table 37 Values of #cond field

value	meaning
0	False
1	True

Registers altered (except destination register)

LR

Occurrence Exceptions

register_exception (unimplement_exception)

Detected Exceptions

none

1.15. Conditional Scan instruction

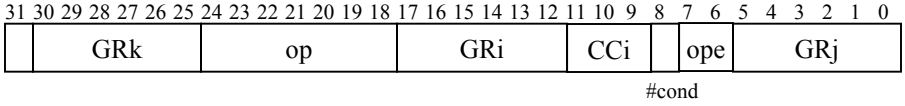
1.15.1. Scan

Ope-code	op	Ope	Operation
CSCAN	1100101	11	Conditional SCAN

Category

Integer, Conditional

Instruction Format (Conditional instruction)



Assembler Syntax

CSCAN GRi, GRj, GRk, CCi, #cond

Description

The Conditional SCAN Instruction operates as SCAN instruction(page 56), when the condition specified by the #cond field is equal to the condition shown with CCi.
Table 38 shows the values of #cond field.

Registers altered (except destination register)

none

Occurrence Exceptions

register_exception (unimplement_exception)

Detected Exceptions

none

Table 38 Values of #cond field

value	meaning
0	False
1	True

1.16. Conditional Condition code operating Instructions

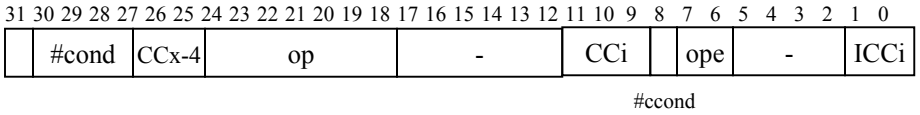
1.16.1. Check for Integer Condition code

Ope-code	op	ope	Operation
CCKicc	1101010	00	Conditional Check for Integer Condition code

Category

Branch, Conditional

Instruction Format (Conditional Branch Instruction)



Assembler Syntax

CCKicc ICCi, CCx, CCi, #ccond (x =4-7)

CCKEQ	: Z==1
CCKNE	: Z==0
CCKLE	: (Z or(N xor V))==1
CCKGT	: (Z or(N xor V))==0
CCKLT	: (N xor V)==1
CCKGE	: (N xor V)==0
CCKLS	: (C or Z)==1
CCKHI	: (C or Z)==0
CCKC	: C ==1
CCKNC	: C ==0
CCKN	: N==1
CCKP	: N==0
CCKV	: V==1
CCKNV	: V==0

CCKNO CCx, CCi, #ccond (x=4–7)
CCKRA CCx, CCi, #ccond (x=4–7)

Description

When the condition specified by the #ccond field is equal to the condition shown with CCi, the CCKicc instruction tests the #cond field of ICCi shown in Table 39 and copies the result in CCx. Otherwise, the instruction copies Undefined to CCx. Table 40 shows the value of #ccond field.

Table 39 #cond field and pseudo opecode

Pseudo Ope-code	cond	Operation	icc test
CCKEQ	0100	Conditional Check Equal	Z
CCKNE	1100	Conditional Check Not Equal	Not Z
CCKLE	0111	Conditional Check Less or Equal	Z or (N xor V)
CCKGT	1111	Conditional Check Greater	Not (Z or (N xor V))
CCKLT	0011	Conditional Check Less	N xor V
CCKGE	1011	Conditional Check Greater or Equal	Not (N xor V)
CCKLS	0101	Conditional Check Less or Equal Unsigned	C or Z
CCKHI	1101	Conditional Check Greater Unsigned	Not (C or Z)
CCKC	0001	Conditional Check Carry Set	C
CCKNC	1001	Conditional Check Carry Clear	Not C
CCKN	0110	Conditional Check Negative	N
CCKP	1110	Conditional Check Positive	Not N
CCKV	0010	Conditional Check Overflow Set	V
CCKNV	1010	Conditional Check Overflow Clear	Not V
CCKNO	0000	Conditional Check Never	0
CCKRA	1000	Conditional Check Always	1

Table 40 Values of #ccond field

value	meaning
0	False
1	True

Table 41 Values of CCCR

value	meaning
00	Undefined
01	Undefined
10	False
11	True

Table 42 test results and values of CCCR

icc result	meaning
True	11
False	10

Registers altered (except destination register)

none

Occurrence Exceptions

none

Detected Exceptions

none

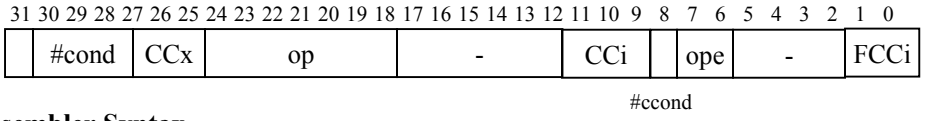
1.16.2. Check for Floating-point/Media Conditional code

Ope-code	op	ope	Operation
CFCKfcc	1101010	01	Conditional Check for Floating-point/Media Conditional code

Category

Branch, Conditional

Instruction Format (Conditional Branch Instruction)



Assembler Syntax

CFCKfcc FCCi, CCx, CCi, #ccond (x = 0-3)
CFCKNE : (L or G or U)==1
CFCKEQ : E==1
CFCKLG : (L or G)==1
CFCKUE : (E or U)==1
CFCKUL : (L or U)==1
CFCKGE : (E or G)==1
CFCKLT : L==1
CFCKUGE : (E or G or U)==1
CFCKUG : (G or U)==1
CFCKLE : (E or L)==1
CFCKGT : G==1
CFCKULE : (E or L or U)==1
CFCKU : U==1
CFCKO : (E or L or G)==1
CFCKNO CCx, CCi, #ccond (x = 0–3)
CFCKRA CCx, CCi, #ccond (x = 0–3)

Description

When the condition specified by the #ccond field is equal to the condition shown with CCi, the CFCKfcc instruction tests the #cond field of FCCi shown in Table43, and copies the result in CCx. Otherwise the instruction copies Undefined to CCx. Table 44 shows the value of #ccond field.

Table 43 #cond field and pseudo ope-code

Pseudo Ope-code	Cond	Operation	fcc test
CFCKNE	0111	Conditional Check Not Equal	L or G or U
CFCKEQ	1000	Conditional Check Equal	E
CFCKLG	0110	Conditional Check Less or Greater	L or G
CFCKUE	1001	Conditional Check Unordered or Equal	E or U
CFCKUL	0101	Conditional Check Unordered or Less	L or U
CFCKGE	1010	Conditional Check Greater or Equal	E or G
CFCKLT	0100	Conditional Check Less	L
CFCKUGE	1011	Conditional Check Unordered or Greater or Equal	E or G or U
CFCKUG	0011	Conditional Check Unordered or Greater	G or U
CFCKLE	1100	Conditional Check Less or Equal	E or L
CFCKGT	0010	Conditional Check Greater	G
CFCKULE	1101	Conditional Check Unordered or Less or Equal	E or L or U
CFCKU	0001	Conditional Check Unordered	U
CFCKO	1110	Conditional Check Ordered	E or L or G
CFCKNO	0000	Conditional Check Never	0
CFCKRA	1111	Conditional Check Always	1

Table 44 Values of #ccond field

value	meaning
0	False
1	True

Table 45 Values of CCCR

value	meaning
00	Undefined
01	Undefined
10	False
11	True

Registers altered (except destination register)

none

Occurrence Exceptions

fp_disabled

Detected Exceptions

none

1.17. Conditional Media Instructions

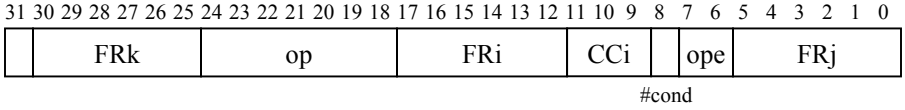
1.17.1. Logical Operations

Ope-code	op	ope	Operation
CMAND	1110000	00	Conditional Media And
CMOR	1110000	01	Conditional Media OR
CMXOR	1110000	10	Conditional Media XOR
CMNOT	1110000	11	Conditional Media NOT

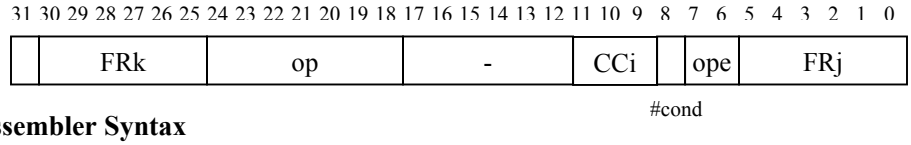
Category

Media, Conditional
Single word

Instruction Format (Conditional instruction)



Instruction Format (Conditional Not-Operation instruction)



Assembler Syntax

CMAND FRi, FRj, FRk, CCi, #cond
CMOR FRi, FRj, FRk, CCi, #cond
CMXOR FRi, FRj, FRk, CCi, #cond
CMNOT FRj, FRk, CCi, #cond

Description

The Conditional Media Logical Instruction operates FRi and FRj in logical. When the condition specified by the #cond field is equal to the condition shown with CCi, the instruction writes the result in FRk. Otherwise, they don't change FRk.
As the description of each instruction, refer to 1.10.2 Logical Operations (page 86).
The CMAND instruction operates as MAND instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CMOR instruction operates as MOR instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CMXOR instruction operates as MXOR instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CMNOT instruction operates as MNOT instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
Table 46 shows the values of #cond field.

Registers altered (except destination register)

none

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception)

Table 46 Values of #cond field

value	meaning
0	False
1	True

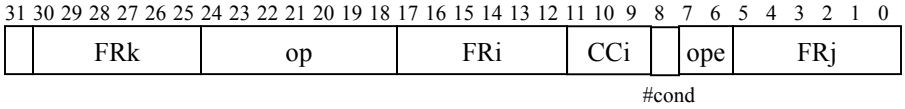
1.17.2. Add / Subtract with Saturation (Halfword Dual)

Ope-code	op	ope	Operation
CMADDHSS	1110001	00	Conditional Dual Add Signed Halfword with Saturation
CMADDHUS	1110001	01	Conditional Dual Add Unsigned Halfword with Saturation
CMSUBHSS	1110001	10	Conditional Dual Subtract Signed Halfword with Saturation
CMSUBHUS	1110001	11	Conditional Dual Subtract Unsigned Halfword with Saturation

Category

Media, Conditional

Instruction Format (Conditional instruction)



Assembler Syntax

CMADDHSS	FRi, FRj, FRk, CCi, #cond
CMADDHUS	FRi, FRj, FRk, CCi, #cond
CMSUBHSS	FRi, FRj, FRk, CCi, #cond
CMSUBHUS	FRi, FRj, FRk, CCi, #cond

Description

The Conditional Media Addition and Subtraction with Saturation Instructions operate FRi and FRj in arithmetic. When the condition specified by the #cond field is equal to the condition shown with CCi, the instruction writes the result in FRk. Otherwise, they don't change FRk.
As the description of each instruction, refer to 1.10.12 Add / Subtract with Saturation (Halfword Dual) (page 98).
The CMADDHSS instruction operates as MADDHSS instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CMADDHUS instruction operates as MADDHUS instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CMSUBHSS instruction operates as MSUBHSS instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CMSUBHUS instruction operates as MSUBHUS instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
Table 47 shows the values of #cond field.

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, overflow)

Table 47 Values of #cond field

value	meaning
0	False
1	True

1.17.3. Multiply and Accumulate (Halfword Dual)

Ope-code	op	ope	Operation
CMMULHS	1110010	00	Conditional Dual Multiply Signed Halfword
CMMULHU	1110010	01	Conditional Dual Multiply Unsigned Halfword
CMMACHS	1110010	10	Conditional Dual Multiply and Accumulate Signed Halfword
CMMACHU	1110010	11	Conditional Dual Multiply and Accumulate Unsigned Halfword

Category

Media, Conditional

Instruction Format (Conditional media ACC instruction)

Assembler Syntax

CMMULHS FRi, FRj, ACCk, CCi, #cond
CMMULHU FRi, FRj, ACCk, CCi, #cond
CMMACHS FRi, FRj, ACCk, CCi, #cond
CMMACHU FRi, FRj, ACCk, CCi, #cond

Description

The Conditional Media Dual Multiply and Accumulation Instructions operate FRi and FRj in arithmetic. When the condition specified by the #cond field is equal to the condition shown with CCi, the instruction writes the result in ACCk. Otherwise, they don't change ACCk.
The Conditional Media Dual Multiply and Accumulation Instructions operate FRi, FRj, ACCGk and ACCk in arithmetic. When the condition specified by the #cond field is equal to the condition shown with CCi, the instruction writes the result in the accumulator which concatenates ACCGk and ACCk as 40-bit integer. Otherwise, they change neither ACCGk nor ACCk.
The CMMULHS instruction operates as MMULHS instruction(page 100), when the condition specified by the #cond field is equal to the condition shown with CCi.
The CMMULHU instruction operates as MMULHU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CMMACHS instruction operates as MMACHS instruction(page 104), when the condition specified by the #cond field is equal to the condition shown with CCi.
The CMMACHU instruction operates as MMACHU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
Table 48 shows the values of #cond field.

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, acc_not_aligned, overflow)

Table 48 Values of #cond field

value	meaning
0	False
1	True

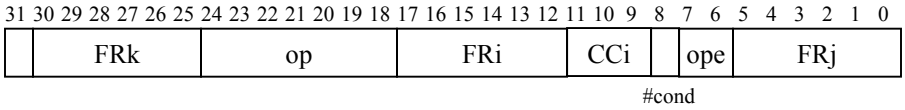
1.17.4. Add / Subtract with Saturation (Halfword Quad)

Ope-code	op	ope	Operation
CMQADDHSS	1110011	00	Conditional Quad Add Signed Halfword with Saturation
CMQADDHUS	1110011	01	Conditional Quad Add Unsigned Halfword with Saturation
CMQSUBHSS	1110011	10	Conditional Quad Subtract Signed Halfword with Saturation
CMQSUBHUS	1110011	11	Conditional Quad Subtract Unsigned Halfword with Saturation

Category

Media, Conditional

Instruction Format (Conditional instruction)



Assembler Syntax

CMQADDHSS	FRi, FRj, FRk, CCi, #cond
CMQADDHUS	FRi, FRj, FRk, CCi, #cond
CMQSUBHSS	FRi, FRj, FRk, CCi, #cond
CMQSUBHUS	FRi, FRj, FRk, CCi, #cond

Description

The Conditional Media Quad Addition and Subtraction with Saturation Instructions operate FRi and FRj in arithmetic. When the condition specified by the #cond field is equal to the condition shown with CCi, the instruction writes the result in FRk. Otherwise, they don't change FRk.
As the description of each instruction, refer to 1.10.17 Add / Subtract with Saturation (Halfword Quad) (page 108).
The CMQADDHSS instruction operates as MQADDHSS instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CMQADDHUS instruction operates as MQADDHUS instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CMQSUBHSS instruction operates as MQSUBHSS instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CMQSUBHUS instruction operates as MQSUBHUS instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
Table 49 shows the values of #cond field.

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned, overflow)

Table 49 Values of #cond field

values	meaning
0	False
1	True

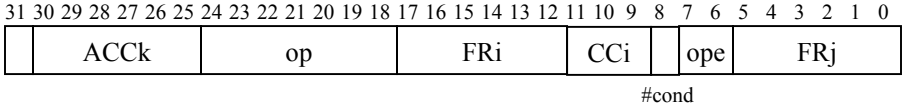
1.17.5. Multiply / Multiply and Accumulate (Halfword Quad)

Ope-code	op	ope	Operation
CMQMULHS	1110100	00	Conditional Quad Multiply Signed Halfword
CMQMULHU	1110100	01	Conditional Quad Multiply Unsigned Halfword
CMQMACHS	1110100	10	Conditional Quad Multiply and Accumulate Signed Halfword
CMQMACHU	1110100	11	Conditional Quad Multiply and Accumulate Unsigned Halfword

Category

Media, Conditional

Instruction Format (Conditional media ACC instruction)



Assembler Syntax

CMQMULHS	FRi, FRj, ACCk, CCi, #cond
CMQMULHU	FRi, FRj, ACCk, CCi, #cond
CMQMACHS	FRi, FRj, ACCk, CCi, #cond
CMQMACHU	FRi, FRj, ACCk, CCi, #cond

Description

The Conditional Media Quad Multiply/Multiply and Accumulation Instructions operate FRi and FRj in arithmetic. When the condition specified by the #cond field is equal to the condition shown with CCi, the instruction writes the result in accumulator which concatenates ACCGk, ACCk as 40-bit integer. Otherwise, they don't change accumulator.

The Conditional Media Quad Multiply/Multiply and Accumulation Instructions operate FRi, FRj, ACCGk and ACCk in arithmetic. When the condition specified by the #cond field is equal to the condition shown with CCi, the instruction writes the result in the accumulator which concatenates ACCGk, ACCk as 40-bit integer. Otherwise, they don't change accumulator.

The CMQMULHS instruction operates as MQMULHS instruction(page 111), when the condition specified by the #cond field is equal to the condition shown with CCi.

The CMQMULHU instruction operates as MQMULHU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.

The CMQMACHS instruction operates as MQMACHS instruction(page 115), when the condition specified by the #cond field is equal to the condition shown with CCi.

The CMQMACHU instruction operates as MQMACHU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.

Table 50 shows the values of #cond field.

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned, acc_not_aligned, overflow)

Table 50 Values of #cond field

value	meaning
0	False
1	True

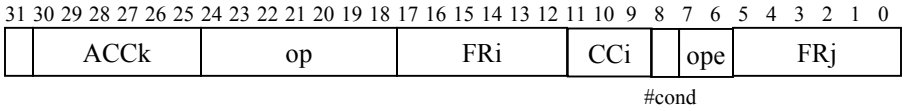
1.17.6. Complex Multiply (Halfword Dual)

Ope-code	op	ope	Operation
CMCPXRS	1110101	00	Conditional Dual Complex Real Signed Halfword
CMCPXRU	1110101	01	Conditional Dual Complex Real Unsigned Halfword
CMCPXIS	1110101	10	Conditional Dual Complex Imaginary Signed Halfword
CMCPXIU	1110101	11	Conditional Dual Complex Imaginary Unsigned Halfword

Category

Media, Conditional

Instruction Format (Conditional instruction)



Assembler Syntax

CMCPXRS FRi, FRj, ACCk, CCi, #cond
CMCPXRU FRi, FRj, ACCk, CCi, #cond
CMCPXIS FRi, FRj, ACCk, CCi, #cond
CMCPXIU FRi, FRj, ACCk, CCi, #cond

Description

The Conditional Media Dual Multiply with Additional for Complex number Instruction operates FRi and FRj in arithmetic. When the condition specified by the #cond field is equal to the condition shown with CCi, the instruction writes the result in accumulator which concatenates ACCGk, ACCk as 40-bit integer. Otherwise, they don't change accumulator.
As the description of each instruction, refer to 1.10.24 Complex Multiply (Halfword Dual) (page 123).
The CMCPXRS instruction operates as MCPXRS instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CMCPXRU instruction operates as MCPXRU instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CMCPXIS instruction operates as MCPXIS instruction, when the condition specified by the #cond field equal to the condition shown with CCi.
The CMCPXIU instruction operates as MCPXIU instruction, when the condition specified by the #cond field equal to the condition shown with CCi.
Table 51 shows the values of #cond field.

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception)

Table 51 Values of #cond field

values	meaning
0	False
1	True

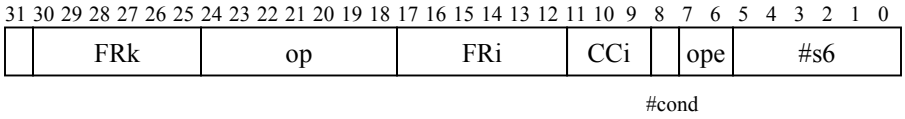
1.17.7. Expand (Halfword)

Ope-code	op	ope	Operation
CMEXPDHW	1110110	10	Conditional Expand Halfword to Word
CMEXPDHD	1110110	11	Conditional Expand Halfword to Double-Word

Category

Media, Conditional

Instruction Format (Conditional instruction)



Assembler Syntax

CMEXPDHW FRi, #s6, FRk, CCi, #cond
CMEXPDHD FRi, #s6, FRk, CCi, #cond

Description

When the condition specified by the #cond field is equal to the condition shown with CCi, the Conditional Media Expand Instructions copy data of FRi indicated by #s6 to FRk. Otherwise, they don't change FRk.
The CMEXPDHW instruction operates as MEXPDHW instruction(page 133), when the condition specified by the #cond field is equal to the condition shown with CCi.
The CMEXPDHD instruction operates as MEXPDHD instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
Table 52 shows the values of #cond field.

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned)

Table 52 Values of #cond field

value	meaning
0	False
1	True

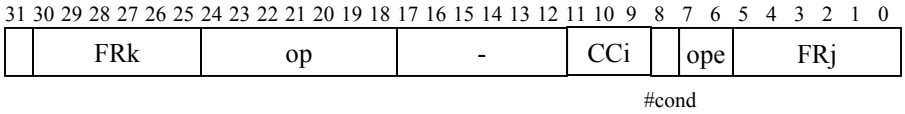
1.17.8. Convert Byte to/from Halfword

Ope-code	op	ope	Operation
CMBTOH	1110111	00	Conditional Byte To Halfword
CMHTOB	1110111	01	Conditional Halfword To Byte

Category

Media, Conditional

Instruction Format (Conditional instruction)



Assembler Syntax

CMBTOH FRj, FRk, CCi, #cond
CMHTOB FRj, FRk, CCi, #cond

Description

The Conditional Media Byte-Halfword Conversion Instructions convert data in FRj. When the condition specified by the #cond field is equal to the condition shown with CCi, the instructions write the result in FRk. Otherwise, they don't change FRk.
As the description of each instruction, refer to 1.10.32 Convert Byte to/from Halfword (page 139).
The CMBTOH instruction operates as MBTOH instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
The CMHTOB instruction operates as MHTOB instruction, when the condition specified by the #cond field is equal to the condition shown with CCi.
Table 53 shows the values of #cond field.

Registers altered (except destination register)

MSR

Occurrence Exceptions

mp_disabled

Detected Exceptions

mp_exception (unimplement_exception, register_not_aligned)

Table 53 Values of #cond field

value	meaning
0	False
1	True

2. VLIW instruction

FR400 series processor is implemented based on embedded Very Long Instruction Word (VLIW) processor architecture (FR-V architecture). This chapter provide the construction method of VLIW instruction, and VLIW instruction behavior.

2.1. Construct ion of VLIW instruction

FR-V architecture is based on Very Long Instruction Word (VLIW) architecture. Each instruction in a VLIW instruction is executed in parallel. VLIW instruction boundaries are specified by packing flag in each instruction. If a packing flag indicates 1, the instruction and the preceding instructions with the packing flag equal to 0 is packed in same VLIW instruction.

FR-V assembler language define instruction option (.p) to specify the instructions in a VLIW instruction. Specifying the instruction option (.p) in the operation field indicates that the succeeding instruction is packed into the same VLIW instruction.

[Example]

<Assembler Language>

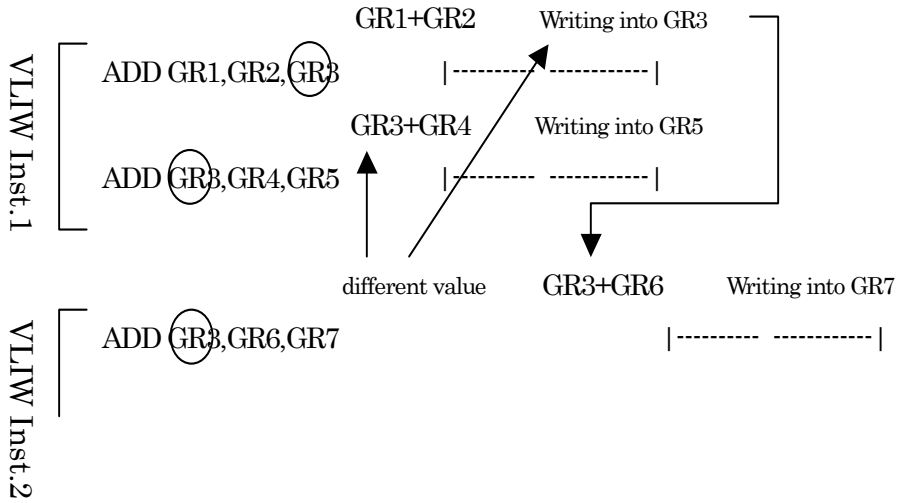
```
I1 : SUBcc.p      GR26,GR22,GR0,icc0
I2 : MSUBHSS.p    FR2,FR3,FR7
I3 : SUBI.p       GR26,#16,GR24
I4 : MADDHSS      FR2,FR3,FR6
I5 : CKLT         ICC0,CC4
I6 : CLDDF.p      @(GR26,GR0), FR0,CC4,#1
I7 : CLDDF        @(GR26,GR23),FR2,CC4,#1
```

<VLIW Instruction>

VLIW1	0	I1	0	I2	0	I3	1	I4
VLIW2	1	I5						
VLIW3	0	I6	1	I7				

2.2. Execution of VLIW instruction

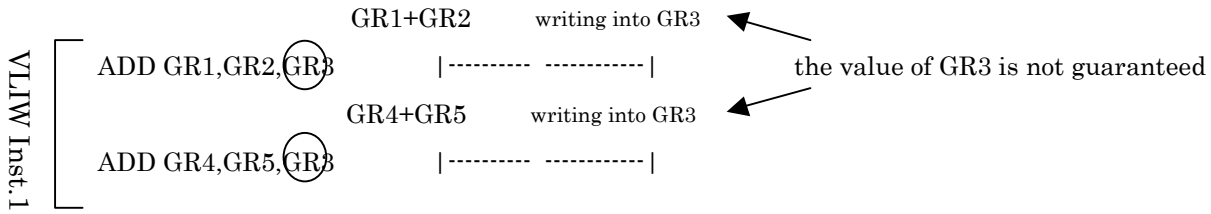
2.2.1. Read/Write operation in same VLIW instruction



The instructions in one VLIW instruction are executed in same time. When the same resource is read and written in the same VLIW instruction, in this case, the reading value is the value before updating.

If the same resource is read and written in same VLIW instruction and the instruction which is read from the same resource detects the non-deferred precise interrupts, the instruction which is written into the same resource completes, and to read and write the same resource in the same VLIW instruction when it is necessary to ensure the recovery from the interrupts.

If storing a result into same resource with different instructions in the same VLIW instruction, the value is not guaranteed.



2.2.2. Execution of Control Transfer Instruction

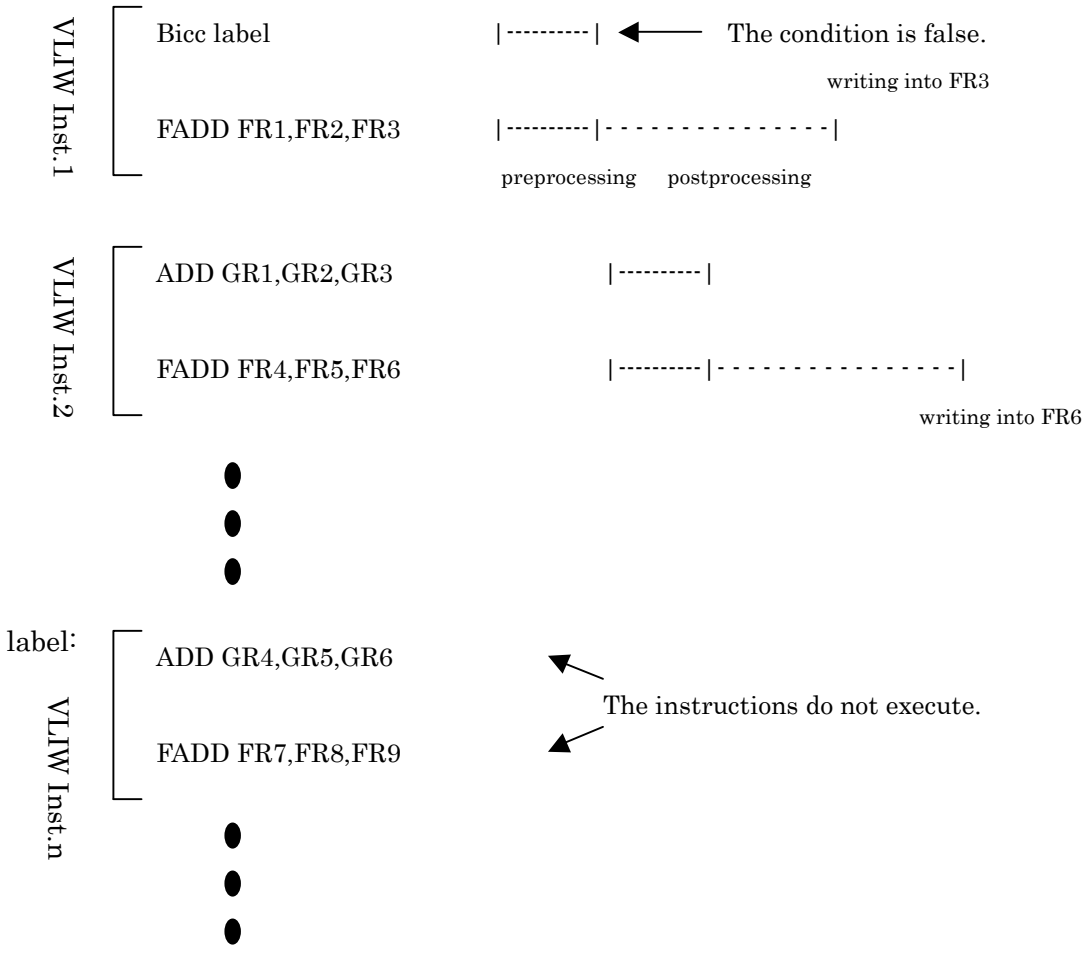
This section explains the operation of the control transfer instruction. The control transfer instruction is composed by the following six instructions.

- conditional branch instruction(Bicc,FBfcc)
- conditional branch instruction to LR(BiccLR,FBfccLR)
- JMP and LINK instruction
- CALL and LINK instruction
- RETT instruction
- trap instruction(Ticc,FTfcc)

2.2.2.1 Execution of Control Transfer Instruction

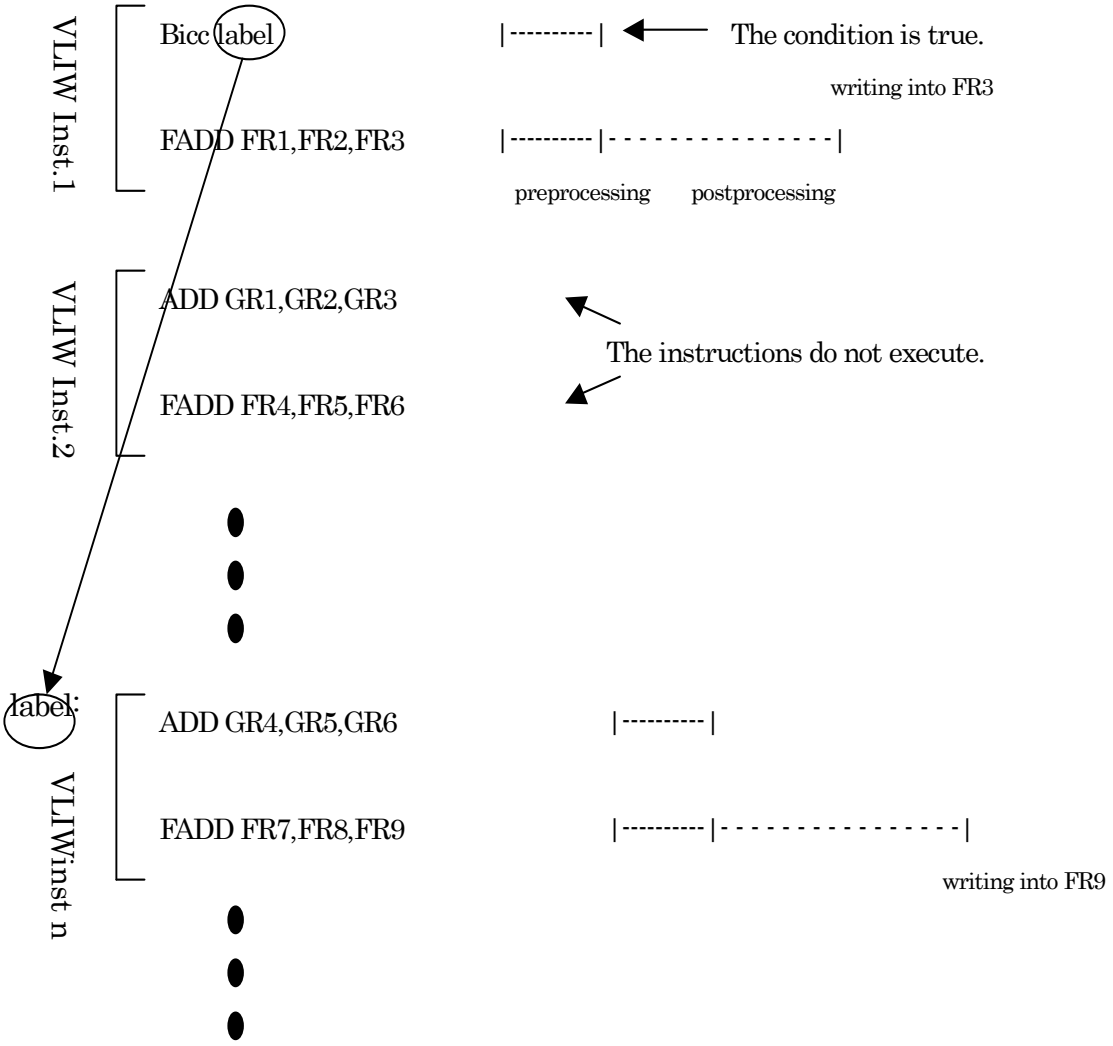
The pipeline behavior when the condition is false and true is explained by using the conditional branch instruction as a control transfer instruction. To make easy to understand, the execution of a control transfer instruction in the VLIW instruction is explained in this section.

◆ When the condition of conditional branch is false.



When the condition of conditional branch is false, the sequential VLIW instruction in the VLIW instruction which contains the conditional branch instruction is executed after the VLIW instruction is executed. The instruction included in the same VLIW instruction as the branch instruction completed execution.

◆ When the condition of conditional branch is true.



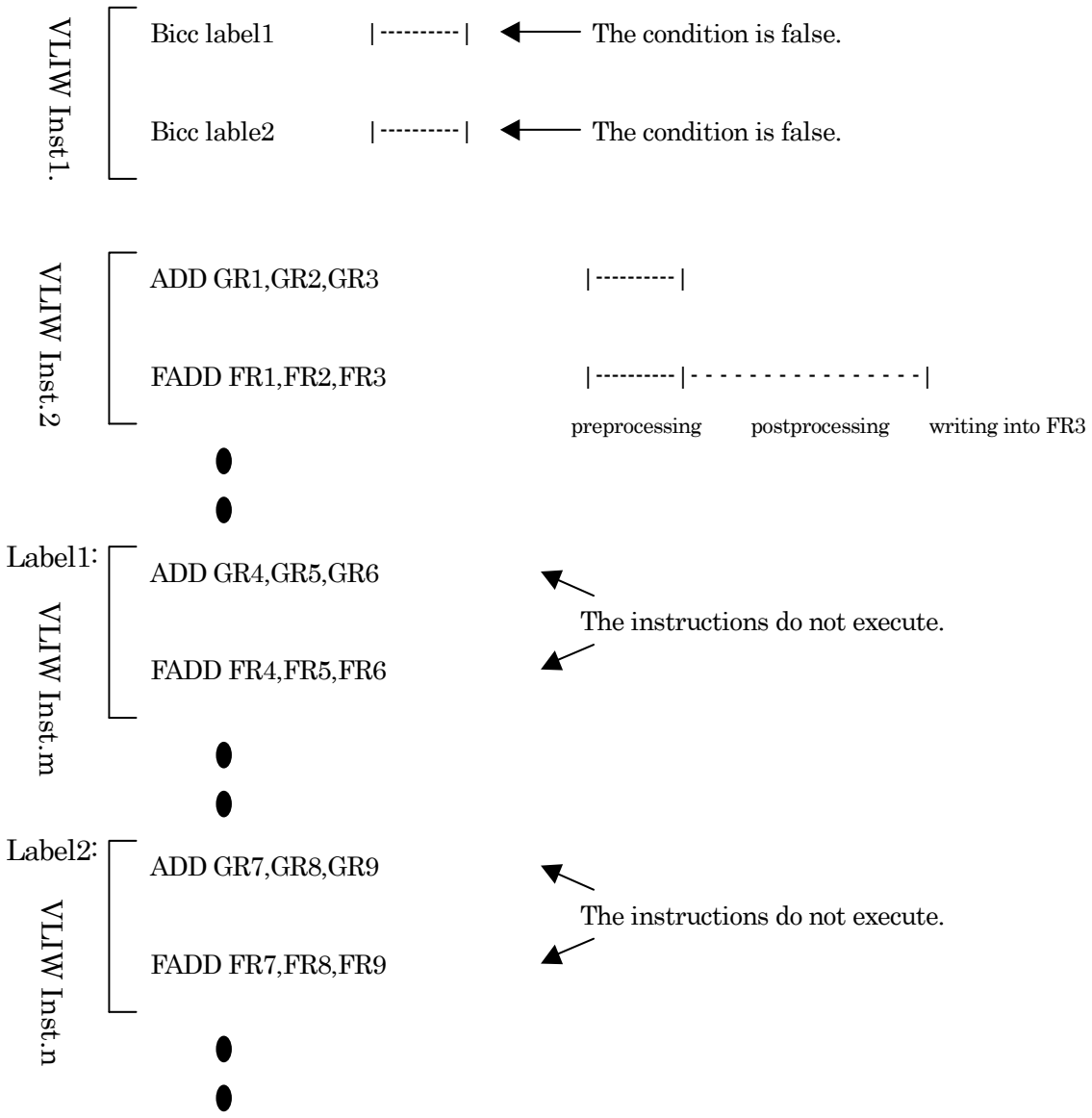
When the condition of conditional branch is true, a target VLIW instruction is executed after the VLIW instruction which contains the conditional branch instruction, and the sequential VLIW instruction in the VLIW instruction is not executed. The instruction included in the same VLIW instruction as the branch instruction completed execution.

The target address of control transfer instruction should be a head of VLIW instruction. The operation when the target address is not at the head of VLIW instruction is not ensure.

2.2.2.2 Execution of Multiple Control Transfer Instruction is One VLIW

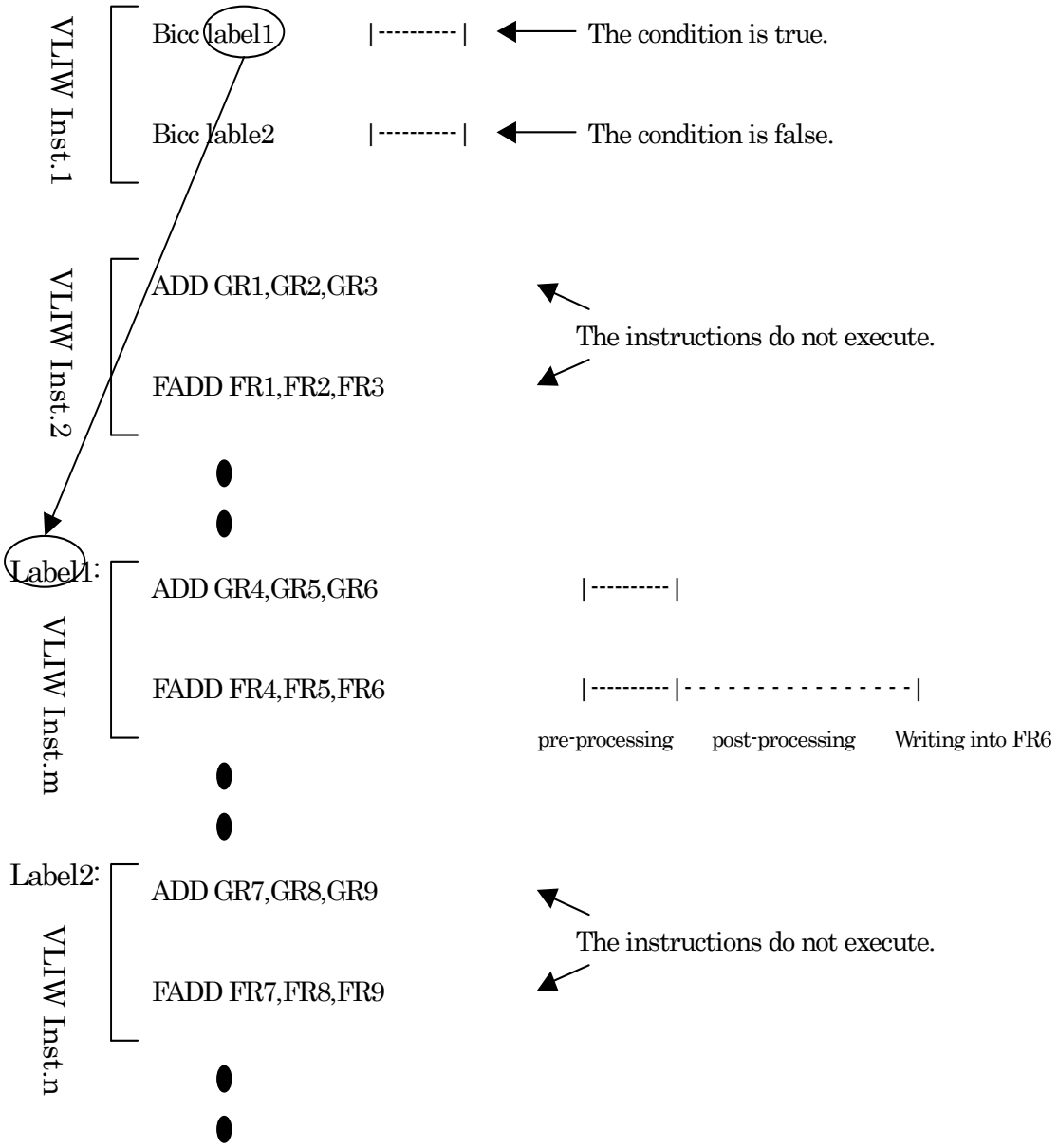
In this section, the execution of multiple control transfer instruction in the VLIW instruction is explained. To make easy to understand the execution of two control transfer instructions in the VLIW instruction are used for the explanation.

◆ When both condition of conditional branch are false



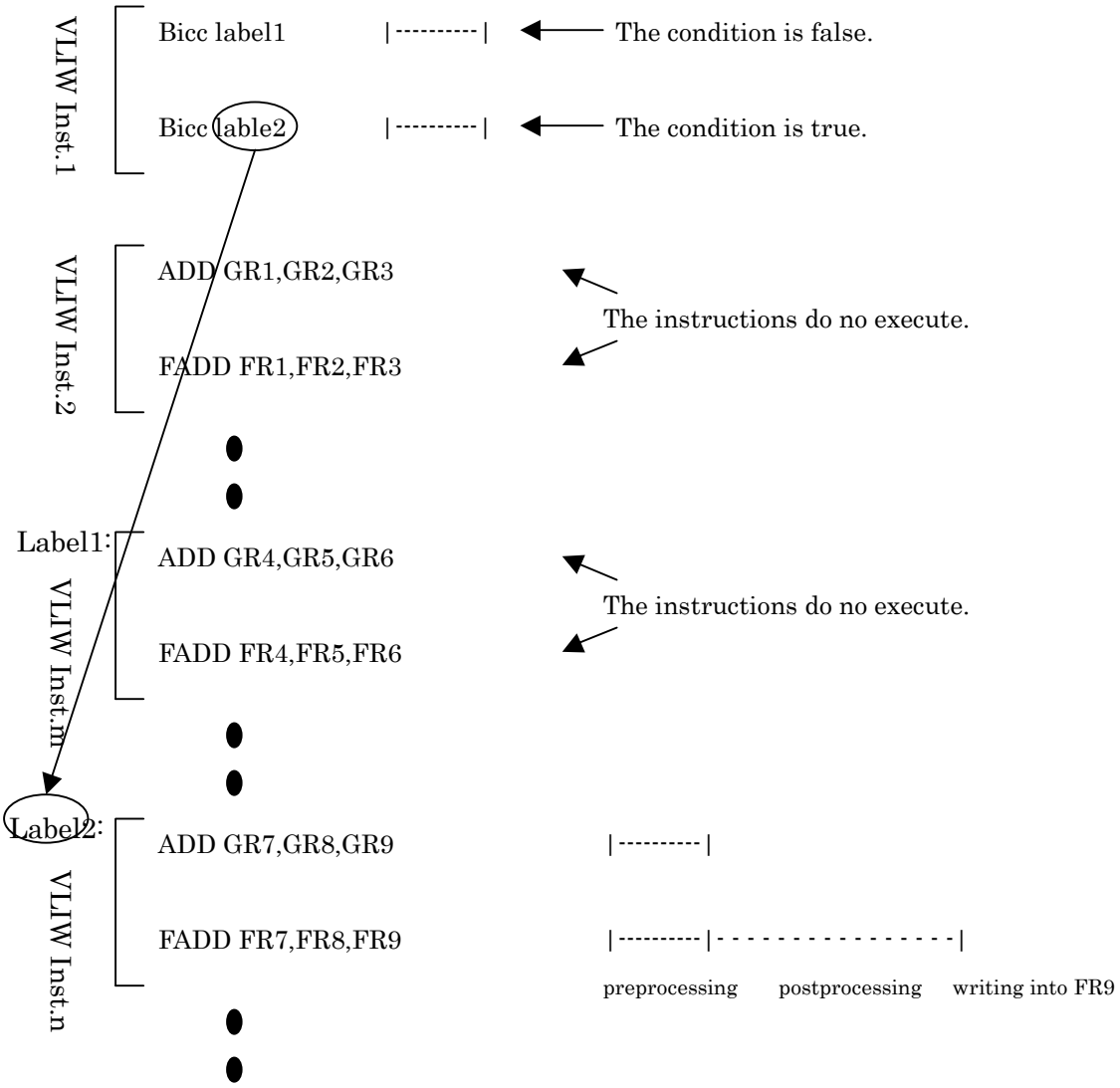
When both of the condition of conditional branch is false, the sequential VLIW instruction in the VLIW instruction which contains the conditional instruction is executed after the VLIW instruction is executed, and the target VLIW instruction do not execute.

◆ When the first condition of conditional branch is true and the second is false.



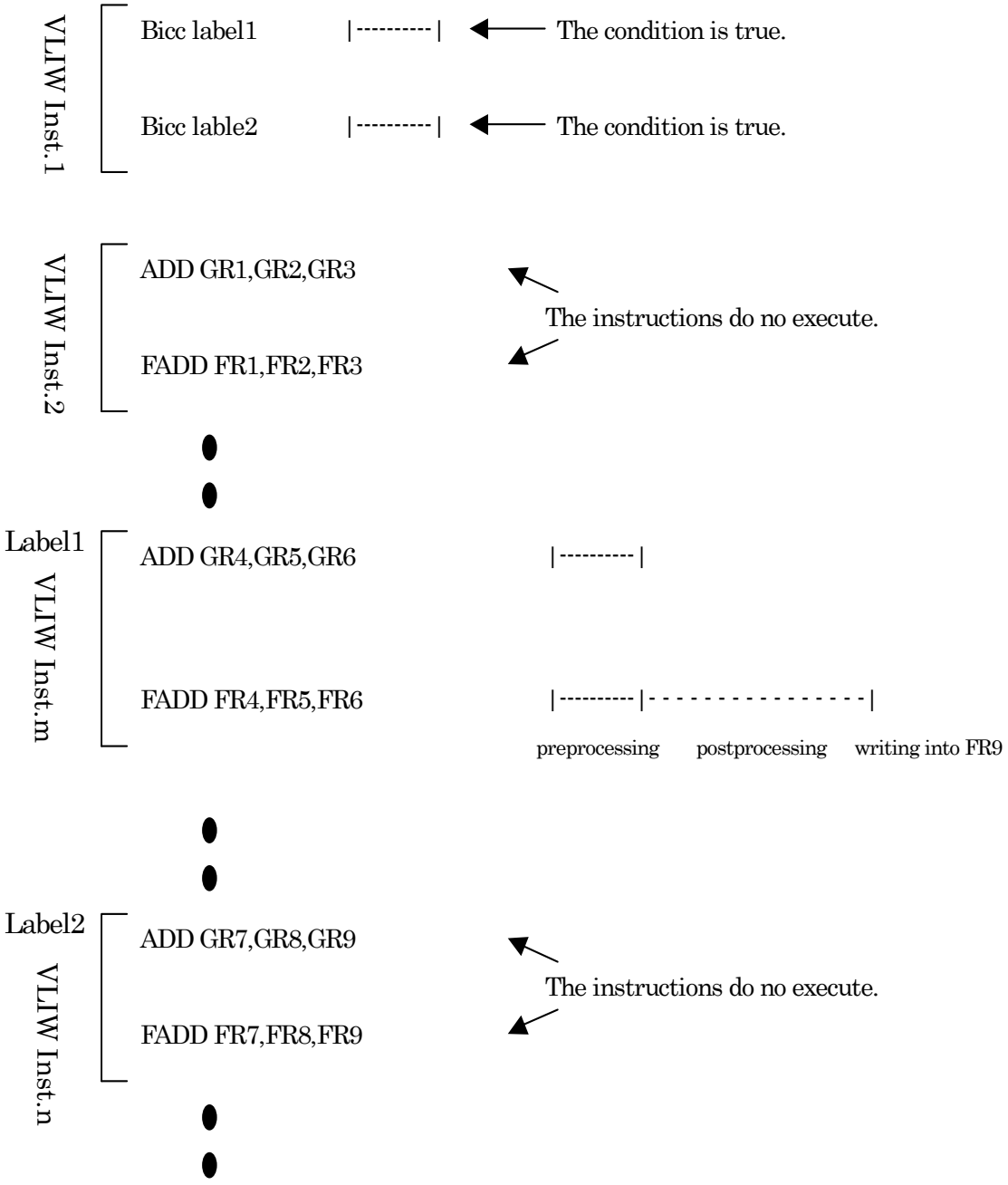
When the first condition of conditional branch is true and the second one is false, the target VLIW instruction which is target of the first conditional branch is executed after the VLIW instruction which contains the conditional branch instruction.

◆ When the first condition of conditional branch is false and the second is true.



When the first condition of conditional branch is false and the second one is true, the target VLIW instruction which is target of the second conditional branch is executed after the VLIW instruction which contains the conditional branch instruction.

◆ When both of the condition of conditional branch are true.



When multiple condition of control transfer instruction are taken, the target VLIW instruction, which is target of the branch instruction with young PC value, is executed.

Appendix

1. Instruction Code Table

	31	30 29 28 27 26 25	24 23 22 21 20 19 18 17 16 15 14 13 12	11 10	9 8 7 6	5 4 3 2 1 0
ADD		GRk	**0000000**	GRi	-	**0000**	GRj
ADDcc		GRk	**0000000**	GRi	ICCi	**0001**	GRj
ADDX		GRk	**0000000**	GRi	ICCi	**0010**	GRj
ADDXcc		GRk	**0000000**	GRi	ICCi	**0011**	GRj
SUB		GRk	**0000000**	GRi	-	**0100**	GRj
SUBcc		GRk	**0000000**	GRi	ICCi	**0101**	GRj
SUBX		GRk	**0000000**	GRi	ICCi	**0110**	GRj
SUBXcc		GRk	**0000000**	GRi	ICCi	**0111**	GRj
SMUL		GRk	**0000000**	GRi	-	**1000**	GRj
SMULcc		GRk	**0000000**	GRi	ICCi	**1001**	GRj
UMUL		GRk	**0000000**	GRi	-	**1010**	GRj
UMULcc		GRk	**0000000**	GRi	ICCi	**1011**	GRj
CMPB		-	**0000000**	GRi	ICCi	**1100**	GRj
CMPBA		-	**0000000**	GRi	ICCi	**1101**	GRj
SDIV		GRk	**0000000**	GRi	-	**1110**	GRj
UDIV		GRk	**0000000**	GRi	-	**1111**	GRj
AND		GRk	**0000001**	GRi	-	**0000**	GRj
ANDcc		GRk	**0000001**	GRi	ICCi	**0001**	GRj
OR		GRk	**0000001**	GRi	-	**0010**	GRj
ORcc		GRk	**0000001**	GRi	ICCi	**0011**	GRj
XOR		GRk	**0000001**	GRi	-	**0100**	GRj
XORcc		GRk	**0000001**	GRi	ICCi	**0101**	GRj
NOT		GRk	**0000001**	-	-	**0110**	GRj
SLL		GRk	**0000001**	GRi	-	**1000**	GRj
SLLcc		GRk	**0000001**	GRi	ICCi	**1001**	GRj
SRL		GRk	**0000001**	GRi	-	**1010**	GRj
SRLcc		GRk	**0000001**	GRi	ICCi	**1011**	GRj
SRA		GRk	**0000001**	GRi	-	**1100**	GRj
SRAcc		GRk	**0000001**	GRi	ICCi	**1101**	GRj

	31	30 29 28 27 26 25	24 23 22 21 20 19 18	17 16 15 14 13 12	11 10 9 8 7 6	5 4 3 2 1 0
LDSB		GRk	**0000010**	GRi	**000000**	GRj
LDUB		GRk	**0000010**	GRi	**000001**	GRj
LDSH		GRk	**0000010**	GRi	**000010**	GRj
LDUH		GRk	**0000010**	GRi	**000011**	GRj
LD		GRk	**0000010**	GRi	**000100**	GRj
LDD		GRk	**0000010**	GRi	**000101**	GRj
LDBF		FRk	**0000010**	GRi	**001000**	GRj
LDHF		FRk	**0000010**	GRi	**001001**	GRj
LDF		FRk	**0000010**	GRi	**001010**	GRj
LDDF		FRk	**0000010**	GRi	**001011**	GRj
LDSBU		GRk	**0000010**	GRi	**010000**	GRj
LDUBU		GRk	**0000010**	GRi	**010001**	GRj
LDSHU		GRk	**0000010**	GRi	**010010**	GRj
LDUHU		GRk	**0000010**	GRi	**010011**	GRj
LDU		GRk	**0000010**	GRi	**010100**	GRj
LDDU		GRk	**0000010**	GRi	**010101**	GRj
LDBFU		FRk	**0000010**	GRi	**011000**	GRj
LDHFU		FRk	**0000010**	GRi	**011001**	GRj
LDFU		FRk	**0000010**	GRi	**011010**	GRj
LDDFU		FRk	**0000010**	GRi	**011011**	GRj

	31	30 29 28 28 27 26	25	24 23 22 21 20 19 18	17 16 15 14 13 12	11 10 9 8 7 6	5 4 3 2 1 0
STB		GRk		**0000011**	GRi	**000000**	GRj
STH		GRk		**0000011**	GRi	**000001**	GRj
ST		GRk		**0000011**	GRi	**000010**	GRj
STD		GRk		**0000011**	GRi	**000011**	GRj
SWAP		GRk		**0000011**	GRi	**000101**	GRj
MOVGS		FRk		**0000011**	sr	**000110**	GRj
MOVSG		FRk		**0000011**	sr	**000111**	GRj
STBF		FRk		**0000011**	GRi	**001000**	GRj
STHF		FRk		**0000011**	GRi	**001001**	GRj
STF		FRk		**0000011**	GRi	**001010**	GRj
STDF		FRk		**0000011**	GRi	**001011**	GRj
MOVFG		FRk		**0000011**	-	**001101**	GRj
MOVFGD		FRk		**0000011**	-	**001110**	GRj
STBU		GRk		**0000011**	GRi	**010000**	GRj
STHU		GRk		**0000011**	GRi	**010001**	GRj
STU		GRk		**0000011**	GRi	**010010**	GRj
STDU		GRk		**0000011**	GRi	**010011**	GRj
MOVGF		FRk		**0000011**	-	**010101**	GRj
MOVGFD		FRk		**0000011**	-	**010110**	GRj
STBFU		FRk		**0000011**	GRi	**011000**	GRj
STHFU		FRk		**0000011**	GRi	**011001**	GRj
STFU		FRk		**0000011**	GRi	**011010**	GRj
STDFU		FRk		**0000011**	GRi	**011011**	GRj
LRAI		GRk		**0000011**	GRi	**100000**	E D S 000
LRAD		GRk		**0000011**	GRi	**100001**	E D S 000
TLBPR		- opx	L	**0000011**	GRi	**100100**	GRj

	31	30 29 28 27 26	25	24 23 22 21 20 19 18 17 16	15 14 13 12 11 10	9 8 7 6 5 4 3 2 1 0
ICPL		-	lock	**0000011**	GRi	**110000**	GRj
ICUL		-		**0000011**	GRi	**110001**	-
DCPL		-	lock	**0000011**	GRi	**110100**	GRj
DCUL		-		**0000011**	GRi	**110101**	-
ICI		-		**0000011**	GRi	**111000**	GRj
ICEI		-	a	**0000011**	GRi	**111001**	GRj
DCEI		-	a	**0000011**	GRi	**111010**	GRj
DCEF		-	a	**0000011**	GRi	**111010**	GRj
DCI		-		**0000011**	GRi	**111100**	GRj
DCF		-		**0000011**	GRi	**111101**	GRj
BAR		-		**0000011**	-	**111110**	-
MEMBAR		-		**0000011**	-	**111111**	-

	31	30 29 28 27	26 25	24 23 22 21 20 19 18	17 16 15 14 13 12	11 10 9 8	7 6	5 4 3 2 1 0
Ticc		#cond	ICCi	**0000100**	GRi	**-**	**00**	GRj
FTfcc		#cond	FCCi	**0000100**	GRi	**-**	**01**	GRj
MTRAP		-		**0000100**	-	**-**	**10**	-
BREAK		-		**0000100**	-	**-**	**11**	-
RETT		-	d	**0000101**	-	**-**		-
Bicc		#cond	ICCi	**0000110**	#hint	label 16		
FBfcc		#cond	FCCi	**0000111**	#hint	label 16		
CKicc		#cond	CCx-4	**0001000**	-			ICCi
FCKfcc		#cond	CCx	**0001001**	-			FCCi
CLRGR		GRk		**0001010**	-	**000000**		
CLRGA		-		**0001010**	-	**000001**		-
CLRFR		FRk		**0001010**	-	**000010**		
CLRFA		-		**0001010**	-	**000011**		-
ANDCR		-	CCz	**0001010**	- CCx	**001000**	-	CCy
ORCR		-	CCz	**0001010**	- CCx	**001001**	-	CCy
XORCR		-	CCz	**0001010**	- CCx	**001010**	-	CCy
NOTCR		-	CCz	**0001010**	-	**001011**	-	CCy
NANDCR		-	CCz	**0001010**	- CCx	**001100**	-	CCy
NORCR		-	CCz	**0001010**	- CCx	**001101**	-	CCy
ANDNCR		-	CCz	**0001010**	- CCx	**010000**	-	CCy
ORNCR		-	CCz	**0001010**	- CCx	**010001**	-	CCy
NANDNCR		-	CCz	**0001010**	- CCx	**010100**	-	CCy
NORNCR		-	CCz	**0001010**	- CCx	**010101**	-	CCy
SCAN		GRk		**0001011**	GRi	**-**		GRj
JMPL		-	0	**0001100**	GRi	**-**		GRj
CALLL		-	1	**0001100**	GRi	**-**		GRj
JMPIL		-	0	**0001101**	GRi	d12		
CALLIL		-	1	**0001101**	GRi	d12		
BctrLR		-		**0001110**	#hint **001**	<-#ccond –		
BiccLR		#cond	ICCi	**0001110**	#hint **010** –	–		
BCiccLR		#cond	ICCi	**0001110**	#hint **011**	<-#ccond –		
FBfccLR		#cond	FCCi	**0001110**	#hint **110** –	–		
FCBfccLR		#cond	FCCi	**0001110**	#hint **111**	<-#ccond –		
CALL		labelh 6		**0001111**	labell 18			

	31	30 29 28 27	26 25	24 23 22 21 20 19 18 17 16 15 14 13 12	11 10	9 8 7 6 5 4 3 2 1 0
ADDI		GRk		**0010000** GRi		#s12
ADDIcc		GRk		**0010001** GRi	ICCi	#s10
ADDXI		GRk		**0010010** GRi	ICCi	#s10
ADDXIcc		GRk		**0010011** GRi	ICCi	#s10
SUBI		GRk		**0010100** GRi		#s12
SUBIcc		GRk		**0010101** GRi	ICCi	#s10
SUBXI		GRk		**0010110** GRi	ICCi	#s10
SUBXIcc		GRk		**0010111** GRi	ICCi	#s10
SMULI		GRk		**0011000** GRi		#s12
SMULIcc		GRk		**0011001** GRi	ICCi	#s10
UMULI		GRk		**0011010** GRi		#s12
UMULIcc		GRk		**0011011** GRi	ICCi	#s10
TIicc		#cond	ICCi	**0011100** GRi		#s12
FTIfcc		#cond	FCCi	**0011101** GRi		#s12
SDIVI		GRk		**0011110** GRi		#s12
UDIVI		GRk		**0011111** GRi		#s12
ANDI		GRk		**0100000** GRi		#s12
ANDIcc		GRk		**0100001** GRi	ICCi	#s10
ORI		GRk		**0100010** GRi		#s12
ORIcc		GRk		**0100011** GRi	ICCi	#s10
XORI		GRk		**0100100** GRi		#s12
XORIcc		GRk		**0100101** GRi	ICCi	#s10
SLLI		GRk		**0101000** GRi		#s12
SLLIcc		GRk		**0101001** GRi	ICCi	#s10
SRLI		GRk		**0101010** GRi		#s12
SRLIcc		GRk		**0101011** GRi	ICCi	#s10
SRAI		GRk		**0101100** GRi		#s12
SRAIcc		GRk		**0101101** GRi	ICCi	#s10

	31	30 29 28 27 26 25	24 23 22 21 20 19	18 17 16	15 14 13 12 11 10	9 8 7 6 5 4 3 2 1 0
LDSBI		GRk	**0110000**	GRi	d12	
LDSHI		GRk	**0110001**	GRi	d12	
LDI		GRk	**0110010**	GRi	d12	
LDDI		GRk	**0110011**	GRi	d12	
LDUBI		GRk	**0110101**	GRi	d12	
LDUHI		GRk	**0110110**	GRi	d12	
LDBFI		FRk	**0111000**	GRi	d12	
LDHFI		FRk	**0111001**	GRi	d12	
LDFI		FRk	**0111010**	GRi	d12	
LDDFI		FRk	**0111011**	GRi	d12	
SETLO		GRk	**0111101**	-	#u16	
SETHI		GRk	**0111110**	-	#i16	
SETLOS		GRk	**0111111**	-	#s16	
ADDSS		GRk	**1000110**	GRi	**000000**	GRj
SUBSS		GRk	**1000110**	GRi	**000001**	GRj
SLASS		GRk	**1000110**	GRi	**000010**	GRj
SCUTSS		GRk	**1000110**	-	**000100**	GRj
SMU		-	**1000110**	GRi	**000101**	GRj
SMASS		-	**1000110**	GRi	**000110**	GRj
SMSSS		-	**1000110**	GRi	**000111**	GRj
SCANI		GRk	**1000111**	GRi	#s12	
SWAPI		GRk	**1001101**	GRi	d12	
STBFI		FRk	**1001110**	GRi	d12	
STHFI		FRk	**1001111**	GRi	d12	
STBI		GRk	**1010000**	GRi	d12	
STHI		GRk	**1010001**	GRi	d12	
STI		GRk	**1010010**	GRi	d12	
STDI		GRk	**1010011**	GRi	d12	
STFI		FRk	**1010101**	GRi	d12	
STDFI		FRk	**1010110**	GRi	d12	

	31	30 29 28 27 26 25	24 23 22 21 20 19 18	17 16 15 14 13 12	11 10	9	8	7 6	5 4 3 2 1 0
CADD		GRk	**1011000**	GRi	CCi	#c	**00**	GRj
CSUB		GRk	**1011000**	GRi	CCi	#c	**01**	GRj
CSMUL		GRk	**1011000**	GRi	CCi	#c	**10**	GRj
CSDIV		GRk	**1011000**	GRi	CCi	#c	**11**	GRj
CADDcc		GRk	**1011001**	GRi	CCi	#c	**00**	GRj
CSUBcc		GRk	**1011001**	GRi	CCi	#c	**01**	GRj
CSMULcc		GRk	**1011001**	GRi	CCi	#c	**10**	GRj
CUDIV		GRk	**1011001**	GRi	CCi	#c	**11**	GRj
CAND		GRk	**1011010**	GRi	CCi	#c	**00**	GRj
COR		GRk	**1011010**	GRi	CCi	#c	**01**	GRj
CXOR		GRk	**1011010**	GRi	CCi	#c	**10**	GRj
CNOT		GRk	**1011010**	-	CCi	#c	**11**	GRj
CAND$_{cc}$		GRk	**1011011**	GRi	CCi	#c	**00**	GRj
COR$_{cc}$		GRk	**1011011**	GRi	CCi	#c	**01**	GRj
CXOR$_{cc}$		GRk	**1011011**	GRi	CCi	#c	**10**	GRj
CSLL		GRk	**1011100**	GRi	CCi	#c	**00**	GRj
CSRL		GRk	**1011100**	GRi	CCi	#c	**01**	GRj
CSRA		GRk	**1011100**	GRi	CCi	#c	**10**	GRj
CSLLcc		GRk	**1011101**	GRi	CCi	#c	**00**	GRj
CSRLcc		GRk	**1011101**	GRi	CCi	#c	**01**	GRj
CSRAcc		GRk	**1011101**	GRi	CCi	#c	**10**	GRj
CLDSB		GRk	**1011110**	GRi	CCi	#c	**00**	GRj
CLDUB		GRk	**1011110**	GRi	CCi	#c	**01**	GRj
CLDSH		GRk	**1011110**	GRi	CCi	#c	**10**	GRj
CLDUH		GRk	**1011110**	GRi	CCi	#c	**11**	GRj
CLD		GRk	**1011111**	GRi	CCi	#c	**00**	GRj
CLDD		GRk	**1011111**	GRi	CCi	#c	**01**	GRj
CLDBF		FRk	**1100000**	GRi	CCi	#c	**00**	GRj
CLDHF		FRk	**1100000**	GRi	CCi	#c	**01**	GRj
CLDF		FRk	**1100000**	GRi	CCi	#c	**10**	GRj
CLDDF		FRk	**1100000**	GRi	CCi	#c	**11**	GRj
CLDSBU		GRk	**1100001**	GRi	CCi	#c	**00**	GRj
CLDUBU		GRk	**1100001**	GRi	CCi	#c	**01**	GRj
CLDSHU		GRk	**1100001**	GRi	CCi	#c	**10**	GRj
CLDUHU		GRk	**1100001**	GRi	CCi	#c	**11**	GRj
CLDU		GRk	**1100010**	GRi	CCi	#c	**00**	GRj
CLDDU		GRk	**1100010**	GRi	CCi	#c	**01**	GRj
CLDBFU		FRk	**1100011**	GRi	CCi	#c	**00**	GRj
CLDHFU		FRk	**1100011**	GRi	CCi	#c	**01**	GRj
CLDFU		FRk	**1100011**	GRi	CCi	#c	**10**	GRj
CLDDFU		FRk	**1100011**	GRi	CCi	#c	**11**	GRj

	31	30 29 28 27 26 25	24 23 22 21 20 19 18 17 16 15 14 13 12 11 10	9	8	7 6	5 4 3 2 1 0
CSTB		GRk	**1100100**	GRi	CCi	#c	**00**	GRj
CSTH		GRk	**1100100**	GRi	CCi	#c	**01**	GRj
CST		GRk	**1100100**	GRi	CCi	#c	**10**	GRj
CSTD		GRk	**1100100**	GRi	CCi	#c	**11**	GRj
CSWAP		GRk	**1100101**	GRi	CCi	#c	**10**	GRj
CSCAN		GRk	**1100101**	GRi	CCi	#c	**11**	GRj
CSTBF		FRk	**1100110**	GRi	CCi	#c	**00**	GRj
CSTHF		FRk	**1100110**	GRi	CCi	#c	**01**	GRj
CSTF		FRk	**1100110**	GRi	CCi	#c	**10**	GRj
CSTDF		FRk	**1100110**	GRi	CCi	#c	**11**	GRj
CSTBU		GRk	**1100111**	GRi	CCi	#c	**00**	GRj
CSTHU		GRk	**1100111**	GRi	CCi	#c	**01**	GRj
CSTU		GRk	**1100111**	GRi	CCi	#c	**10**	GRj
CSTDU		GRk	**1100111**	GRi	CCi	#c	**11**	GRj
CSTBFU		FRk	**1101000**	GRi	CCi	#c	**00**	GRj
CSTHFU		FRk	**1101000**	GRi	CCi	#c	**01**	GRj
CSTFU		FRk	**1101000**	GRi	CCi	#c	**10**	GRj
CSTDFU		FRk	**1101000**	GRi	CCi	#c	**11**	GRj
CMOVGF		FRk	**1101001**	-	CCi	#c	**00**	GRj
CMOVGFD		FRk	**1101001**	-	CCi	#c	**01**	GRj
CMOVFG		FRk	**1101001**	-	CCi	#c	**10**	GRj
CMOVFGD		FRk	**1101001**	-	CCi	#c	**11**	GRj
CCKicc		#cond CCx-4	**1101010**	-	CCi	*	**00**	- ICCi
CFCKfcc		#cond CCx	**1101010**	-	CCi	*	**01**	- FCCi
CJMPL		- 0	**1101010**	GRi	CCi	#c	**10**	GRj
CCALLL		- 1	**1101010**	GRi	CCi	#c	**10**	GRj

#c :#cond
* :#ccond

CMAND	FRk	**1110000**	FRi	CCi	#c	**00**	FRj
CMOR	FRk	**1110000**	FRi	CCi	#c	**01**	FRj
CMXOR	FRk	**1110000**	FRi	CCi	#c	**10**	FRj
CMNOT	FRk	**1110000**	-	CCi	#c	**11**	FRj
CMADDHSS	FRk	**1110001**	FRi	CCi	#c	**00**	FRj
CMADDHUS	FRk	**1110001**	FRi	CCi	#c	**01**	FRj
CMSUBHSS	FRk	**1110001**	FRi	CCi	#c	**10**	FRj
CMSUBHUS	FRk	**1110001**	FRi	CCi	#c	**11**	FRj
CMMULHS	ACCk	**1110010**	FRi	CCi	#c	**00**	FRj
CMMULHU	ACCk	**1110010**	FRi	CCi	#c	**01**	FRj
CMMACHS	ACCk	**1110010**	FRi	CCi	#c	**10**	FRj
CMMACHU	ACCk	**1110010**	FRi	CCi	#c	**11**	FRj
CMQADDHSS	FRk	**1110011**	FRi	CCi	#c	**00**	FRj
CMQADDHUS	FRk	**1110011**	FRi	CCi	#c	**01**	FRj
CMQSUBHSS	FRk	**1110011**	FRi	CCi	#c	**10**	FRj
CMQSUBHUS	FRk	**1110011**	FRi	CCi	#c	**11**	FRj
CMQMULHS	ACCk	**1110100**	FRi	CCi	#c	**00**	FRj
CMQMULHU	ACCk	**1110100**	FRi	CCi	#c	**01**	FRj
CMQMACHS	ACCk	**1110100**	FRi	CCi	#c	**10**	FRj
CMQMACHU	ACCk	**1110100**	FRi	CCi	#c	**11**	FRj
CMCPXRS	ACCk	**1110101**	FRi	CCi	#c	**00**	FRj
CMCPXRU	ACCk	**1110101**	FRi	CCi	#c	**01**	FRj
CMCPXIS	ACCk	**1110101**	FRi	CCi	#c	**10**	FRj
CMCPXIU	ACCk	**1110101**	FRi	CCi	#c	**11**	FRj
CMEXPDHW	FRk	**1110110**	FRi	CCi	#c	**10**	#s6
CMEXPDHD	FRk	**1110110**	FRi	CCi	#c	**11**	#s6
CMBTOH	FRk	**1110111**	-	CCi	#c	**00**	FRj
CMHTOB	FRk	**1110111**	-	CCi	#c	**01**	FRj

#c :#cond

	31	30 29 28 27	26 25 24 23 22 21 20	19 18 17 16 15	14 13 12 11 10 9 8 7	6 5 4 3 2 1 0
MQXMACHS		ACCk	**1111000**	FRi	**000000**	FRj
MQXMACXHS		ACCk	**1111000**	FRi	**000001**	FRj
MQMACXHS		ACCk	**1111000**	FRi	**000010**	FRj
MADDACCS		ACCk	**1111000**	ACCi	**000100**	-
MSUBACCS		ACCk	**1111000**	ACCi	**000101**	-
MDADDACCS		ACCk	**1111000**	ACCi	**000110**	-
MDSUBACCS		ACCk	**1111000**	ACCi	**000111**	-
MASACCS		ACCk	**1111000**	ACCi	**001000**	-
MDASACCS		ACCk	**1111000**	ACCi	**001001**	-
MABSHS		FRk	**1111000**	-	**001010**	FRj
MDROTLI		FRk	**1111000**	FRi	**001011**	#s6
MCPLHI		FRk	**1110000**	FRi	**001100**	#s6
MCPLI		FRk	**1110000**	FRi	**001101**	#s6
MDCUTSSI		FRk	**1110000**	ACCi	**001110**	#s6
MQSATHS		FRk	**1111000**	FRi	**001111**	FRj
MQLCLRHS		FRk	**1111000**	FRi	**010000**	FRj
MQSLLHI		FRk	**1111000**	FRi	**010001**	#s6
MQSRAHI		FRk	**1111000**	FRi	**010011**	#s6
MQLMTHS		FRk	**1111000**	FRi	**010100**	FRj
MHSETLOS		FRk	**1111000**	#u6_1	**100000**	#u6_2
MHSETLOH		FRk	**1111000**	-	**100001**	- #s5
MHSETHIS		FRk	**1111000**	#u6_1	**100010**	#u6_2
MHSETHIH		FRk	**1111000**	-	**100011**	- #s5
MHDSETS		FRk	**1111000**	#u6_1	**100100**	#u6_2
MHDSETH		FRk	**1111000**	-	**100101**	- #s5

	31	30 29 28	27 26 25	24 23 22 21 20 19 18	17 16 15 14 13 12	11 10 9 8 7 6	5 4 3 2 1 0
MAND		FRk		1111011	FRi	000000	FRj
MOR		FRk		1111011	FRi	000001	FRj
MXOR		FRk		1111011	FRi	000010	FRj
MNOT		FRk		1111011	-	000011	FRj
MROTLI		FRk		1111011	FRi	000100	#s6
MROTRI		FRk		1111011	FRi	000101	#s6
MWCUT		FRk		1111011	FRi	000110	FRj
MWCUTI		FRk		1111011	FRi	000111	#u6
MAVEH		FRk		1111011	FRi	001000	FRj
MSLLHI		FRk		1111011	FRi	001001	#s6
MSRLHI		FRk		1111011	FRi	001010	#s6
MSRAHI		FRk		1111011	FRi	001011	#s6
MSATHS		FRk		1111011	FRi	001100	FRj
MSATHU		FRk		1111011	FRi	001101	FRj
MCMPSH		-	FCCi	1111011	FRi	001110	FRj
MCMPUH		-	FCCi	1111011	FRi	001111	FRj
MADDHSS		FRk		1111011	FRi	010000	FRj
MADDHUS		FRk		1111011	FRi	010001	FRj
MSUBHSS		FRk		1111011	FRi	010010	FRj
MSUBHUS		FRk		1111011	FRi	010011	FRj
MMULHS		ACCk		1111011	FRi	010100	FRj
MMULHU		ACCk		1111011	FRi	010101	FRj
MMACHS		ACCk		1111011	FRi	010110	FRj
MMACHU		ACCk		1111011	FRi	010111	FRj
MQADDHSS		FRk		1111011	FRi	011000	FRj
MQADDHUS		FRk		1111011	FRi	011001	FRj
MQSUBHSS		FRk		1111011	FRi	011010	FRj
MQSUBHUS		FRk		1111011	FRi	011011	FRj
MQMULHS		ACCk		1111011	FRi	011100	FRj
MQMULHU		ACCk		1111011	FRi	011101	FRj
MQMACHS		ACCk		1111011	FRi	011110	FRj
MQMACHU		ACCk		1111011	FRi	011111	FRj
MCPXRS		ACCk		1111011	FRi	100000	FRj
MCPXRU		ACCk		1111011	FRi	100001	FRj
MCPXIS		ACCk		1111011	FRi	100010	FRj
MCPXIU		ACCk		1111011	FRi	100011	FRj
MQCPXRS		ACCk		1111011	FRi	100100	FRj
MQCPXRU		ACCk		1111011	FRi	100101	FRj
MQCPXIS		ACCk		1111011	FRi	100110	FRj
MQCPXIU		ACCk		1111011	FRi	100111	FRj
MMULXHS		ACCk		1111011	FRi	101000	FRj
MMULXHU		ACCk		1111011	FRi	101001	FRj
MQMULXHS		ACCk		1111011	FRi	101010	FRj
MQMULXHU		ACCk		1111011	FRi	101011	FRj

	31	30 29 28 27 26 25	24 23 22 21 20 19	18 17 16	15 14 13 12 11 10	9 8 7 6 5 4 3 2 1 0
MCUT		FRk	**1111011**	ACCi	**101100**	FRj
MCUTSS		FRk	**1111011**	ACCi	**101101**	FRj
MCUTI		FRk	**1111011**	ACCi	**101110**	#s6
MCUTSSI		FRk	**1111011**	ACCi	**101111**	#s6
MMRDHS		ACCk	**1111011**	FRi	**110000**	FRj
MMRDHU		ACCk	**1111011**	FRi	**110001**	FRj
MEXPDHW		FRk	**1111011**	FRi	**110010**	#s6
MEXPDHD		FRk	**1111011**	FRi	**110011**	#s6
MPACKH		FRk	**1111011**	FRi	**110100**	FRj
MUNPACKH		FRk	**1111011**	FRi	**110101**	-
MDPACKH		FRk	**1111011**	FRi	**110110**	FRj
MBTOH		FRk	**1111011**	-	**111000**	FRj
MHTOB		FRk	**1111011**	-	**111001**	FRj
MCLRACC		ACCi	**1111011**	#A	- **111011**	-
MNOP		**111111**	**1111011**	1	- **111011**	-
MRDACC		FRk	**1111011**	ACCi	**111100**	-
MWTACC		ACCk	**1111011**	FRi	**111101**	-
MRDACCG		FRk	**1111011**	ACCGi	**111110**	-
MWTACCG		ACCGk	**1111011**	FRi	**111111**	-

2. Instruction Matrix

2.1. Primary Ope-code

op[6:3]	op[2:0]							
	000	001	010	011	100	101	110	111
0000	+++	+++	+++	+++	+++	*RETT*	Bicc	FBfcc
0001	CKicc	FCKfcc	*+++*	*SCAN*	JMPL	JMPIL	+++	CALL
0010	ADDI	ADDIcc	ADDXI	ADDXIcc	SUBI	SUBIcc	SUBXI	SUBXIcc
0011	SMULI	SMULIcc	UMULI	UMULIcc	TIicc	FTIfcc	SDIVI	UDIVI
0100	ANDI	ANDIcc	ORI	ORIcc	XORI	XORIcc	---	---
0101	SLLI	SLLIcc	SRLI	SRLIcc	SRAI	SRAIcc	---	---
0110	LDSBI	*LDSHI*	*LDI*	*LDDI*	---	*LDUBI*	*LDUHI*	---
0111	LDBFI	LDHFI	LDFI	LDDFI	---	*SETLO*	*SETHI*	*SETLOS*
1000	---	---	---	---	---	---	+++	*SCANI*
1001	---	---	---	---	---	*SWAPI*	STBFI	STHFI
1010	STBI	STHI	STI	STDI	---	STFI	STDFI	---
1011	+++	+++	+++	+++	+++	+++	+++	+++
1100	+++	+++	+++	+++	+++	+++	+++	+++
1101	+++	+++	+++	+++	+++	+++	+++	+++
1110	+++	+++	+++	+++	+++	+++	+++	+++
1111	---	+++	---	+++	---	---	---	---

The column of +++ has the secondary ope-code.

2.2. Secondary Opecode

Primary=0000000

ope[3]	ope[2:0]							
	000	001	010	011	100	101	110	111
0	ADD	ADDcc	ADDX	ADDXcc	SUB	SUBcc	SUBX	SUBXcc
1	SMUL	SMULcc	UMUL	UMULcc	CMPB	CMPBA	SDIV	UDIV

Primary=0000001

ope[3]	ope[2:0]							
	000	001	010	011	100	101	110	111
0	AND	ANDcc	OR	ORcc	XOR	XORcc	NOT	---
1	SLL	SLLcc	SRL	SRLcc	SRA	SRAcc	---	---

Primary=0000010

ope[5:3]	ope[2:0]							
	000	001	010	011	100	101	110	111
000	LDSB	LDUB	LDSH	LDUH	LD	LDD	---	---
001	LDBF	LDHF	LDF	LDDF	---	---	---	---
010	LDSBU	LDUBU	LDSHU	LDUHU	LDU	LDDU	---	---
011	LDBFU	LDHFU	LDFU	LDDFU	---	---		---
100	---	---	---	---	---	---	---	---
101	---	---	---	---	---	---	---	---
110	---	---	---	---	---	---	---	---
111	---	---	---	---	---	---	---	---

Primary=0000011

ope[5:3]	ope[2:0]							
	000	001	010	011	100	101	110	111
000	STB	STH	ST	STD	---	*SWAP*	*MOVGS*	*MOVSG*
001	STBF	STHF	STF	STDF	---	*MOVFG*	*MOVFGD*	---
010	STBU	STHU	STU	STDU	---	*MOVGF*	*MOVGFD*	---
011	STBFU	STHFU	STFU	STDFU	---	---	---	---
100	LRAI	LRAD	---	---	TLBPR	---	---	---
101	---	---	---	---	---	---	---	---
110	ICPL	ICUL	---	---	DCPL	DCUL	---	---
111	ICI	ICEI	DCEI	DCEF	DCI	DCF	*BAR*	*MEMBAR*

Primary=0000100

	ope[1:0]			
	00	01	10	11
---	Ticc	FTfcc	*MTRAP*	BREAK

Primary=0001010

ope[5:3]	ope[2:0]							
	000	001	010	011	100	101	110	111
000	CLRGR	CLRGA	CLRFR	CLRFA	---	---	---	---
001	ANDCR	ORCR	XORCR	NOTCR	NANDCR	NORCR	---	---
010	ANDNCR	ORNCR	---	---	NANDNCR	NORNCR	---	---
011	---	---	---	---	---	---	---	---
100	---	---	---	---	---	---	---	---
101	---	---	---	---	---	---	---	---
110	---	---	---	---	---	---	---	---
111	---	---	---	---	---	---	---	---

Primary=0001110

ope[2:0]							
000	001	010	011	100	101	110	111
---	BctrLR	BiccLR	BCiccLR	---	---	FBfccLR	FCBfccLR

Primary=1000110

ope[2:0]							
000	001	010	011	100	101	110	111
ADDSS	SUBSS	SLASS	---	SCUTSS	SMU	SMASS	SMSSS

Primary=1011000

ope[1:0]			
00	01	10	11
CADD	CSUB	CSMUL	CSDIV

Primary=1011001

ope[1:0]			
00	01	10	11
CADDcc	CSUBcc	CSMULcc	CUDIV

Primary=1011010

ope[1:0]			
00	01	10	11
CAND	COR	CXOR	CNOT

Primary=1011011

ope[1:0]			
00	01	10	11
CANDcc	CORcc	CXORcc	---

Primary=1011100

ope[1:0]			
00	01	10	11
CSLL	CSRL	CSRA	---

Primary=1011101

ope[1:0]			
00	01	10	11
CSLLcc	CSRLcc	CSRAcc	---

Primary=1011110

ope[1:0]			
00	01	10	11
CLDSB	CLDUB	CLDSH	CLDUH

Primary=1011111

ope[1:0]			
00	01	10	11
CLD	CLDD	---	---

Primary=1100000

ope[1:0]			
00	01	10	11
CLDBF	CLDHF	CLDF	CLDDF

Primary=1100001

ope[1:0]			
00	01	10	11
CLDSBU	CLDUBU	CLDSHU	CLDUHU

Primary=1100010

ope[1:0]			
00	01	10	11
CLDU	CLDDU	---	---

Primary=1100011

ope[1:0]			
00	01	10	11
CLDBFU	CLDHFU	CLDFU	CLDDFU

Primary=1100100

ope[1:0]			
00	01	10	11
CSTB	CSTH	CST	CSTD

Primary=1100101

ope[1:0]			
00	01	10	11
---	---	CSWAP	CSCAN

Primary=1100110

ope[1:0]			
00	01	10	11
CSTBF	CSTHF	CSTF	CSTDF

Primary=1100111

ope[1:0]			
00	01	10	11
CSTBU	CSTHU	CSTU	CSTDU

Primary=1101000

ope[1:0]			
00	01	10	11
CSTBFU	CSTHFU	CSTFU	CSTDFU

Primary=1101001

ope[1:0]			
00	01	10	11
CMOVGF	CMOVGFD	CMOVFG	CMOVFGD

Primary=1101010

ope[1:0]			
00	01	10	11
CCKicc	CFCKfcc	CJMPL	---

Primary=1110000

ope[1:0]			
00	01	10	11
CMAND	CMOR	CMXOR	CMNOT

Primary=1110001

ope[1:0]			
00	01	10	11
CMADDHSS	CMADDHUS	CMSUBHSS	CMSUBHUS

Primary=1110010

ope[1:0]			
00	01	10	11
CMMULHS	CMMULHU	CMMACHS	CMMACHU

Primary=1110011

ope[1:0]			
00	01	10	11
CMQADDHSS	CMQADDHUS	CMQSUBHSS	CMQSUBHUS

Primary=1110100

ope[1:0]			
00	01	10	11
CMQMULHS	CMQMULHU	CMQMACHS	CMQMACHU

Primary=1110101

ope[1:0]			
00	01	10	11
CMCPXRS	CMCPXRU	CMCPXIS	CMCPXIU

Primary=1110110

ope[1:0]			
00	01	10	11
---	---	CMEXPDHW	CMEXPDHD

Primary=1110111

ope[1:0]			
00	01	10	11
CMBTOH	CMHTOB	---	---

Primary=1111000

ope[5:3]	ope[2:0]							
	000	001	010	011	100	101	110	111
000	MQXMACHS	MQXMACXHS	MQMACXHS	---	MADDACCS	MSUBACCS	MDADDACCS	MDSUBACCS
001	MASACCS	MDASACCS	MABSHS	MDROTLI	MCPLHI	MCPLI	MDCUTSSI	MQSATHS
010	MQLCLRHS	MQSLLHI	---	MQSRAHI	MQLMTHS	---	---	---
011	---	---	---	---	---	---	---	---
100	MHSETLOS	MHSETLOH	MHSETHIS	MHSETHIH	MHDSETS	MHDSETH	---	---
101	---	---	---	---	---	---	---	---
110	---	---	---	---	---	---	---	---
111	---	---	---	---	---	---	---	---

Primary=1111011

ope[5:3]	ope[2:0]							
	000	001	010	011	100	101	110	111
000	MAND	MOR	MXOR	MNOT	MROTLI	MROTRI	MWCUT	MWCUTI
001	MAVEH	MSLLHI	MSRLHI	MSRAHI	MSATHS	MSATHU	MCMPSH	MCMPUH
010	MADDHSS	MADDHUS	MSUBHSS	MSUBHUS	MMULHS	MMULHU	MMACHS	MMACHU
011	MQADDHSS	MQADDHU	MQSUBHSS	MQSUBHUS	MQMULHS	MQMULHU	MQMACHS	MQMACHU
100	MCPXRS	MCPXRU	MCPXIS	MCPXIU	MQCPXRS	MQCPXRU	MQCPXIS	MQCPXIU
101	MMULXHS	MMULXHU	MQMULXHS	MQMULXH	MCUT	MCUTSS	MCUTI	MCUTSSI
110	MMRDHS	MMRDHU	MEXPDHW	MEXPDHD	MPACKH	MUNPACKH	MDPACKH	---
111	MBTOH	MHTOB	---	MCLRACC	MRDACC	MWTACC	MRDACCG	MWTACCG

3. Instruction / Device No. Correspondence table

Instruction	Device No.					
	MB93401	MB93401A	MB93402	MB93403	MB93405	MB93451
ADDSS	N/A	N/A	N/A	N/A	available	available
SUBSS	N/A	N/A	N/A	N/A	available	available
SLASS	N/A	N/A	N/A	N/A	available	available
SCUTSS	N/A	N/A	N/A	N/A	available	available
SMU	N/A	N/A	N/A	N/A	available	available
SMASS	N/A	N/A	N/A	N/A	available	available
SMSSS	N/A	N/A	N/A	N/A	available	available
LRAI	N/A	N/A	N/A	N/A	N/A	available
LRAD	N/A	N/A	N/A	N/A	N/A	available
TLBPR	N/A	N/A	N/A	N/A	N/A	available
MQLCLRHS	N/A	N/A	N/A	N/A	N/A	available
MQLMTHS	N/A	N/A	N/A	N/A	N/A	available
MQSRAHI	N/A	N/A	N/A	N/A	N/A	available
MQSLLHI	N/A	N/A	N/A	N/A	N/A	available
other	available	available	available	available	available	available

4. IACC0 special rule

IACC0(SPR N0. 280-281) is added for MB93405/MB93451. It doesn't exist on MB93401/MB93401A/MB93402/MB93403.
Special rule for IACC0 is below.

1. On MB93401/MB93401A/MB93402/MB93403, when movsg instruction reads IACC0, value is 0. Serialization is occurred.
2. On MB93401/MB93401A/MB93402/MB93403, when movgs instruction writes IACC0, any register is not updated. Serialization is occurred.
3. On MB93405/MB93451, when movsg instruction reads IACC0, correct value is read. Serialization is not occurred.
4. On MB93405/MB93451, when movgs instruction writes IACC0, IACC0 is updated. Serialization is not occurred.

To achive common context save/restore routine for all FR400 series without checking CPU version, should only saving IACC0 by movsg and restoreing IACC0 by movgs.

Index