XBurst® Instruction Set Architecture

MIPS extension/enhanced Unit 2

Programming Manual

Release Date: June 2, 2017

北京君正集成电路股份有限公司
Ingenic Semiconductor Co.,Ltd.

XBurst® Instruction Set Architecture

Programming Manual

Disclaimer

Ingenic Semiconductor Co., Ltd.

Ingenic Headquarters, East Bldg. 14, Courtyard #10
Xibeiwang East Road, Haidian District, Beijing, China,
Tel: 86-10-56345000
Fax:86-10-56345001
Http: //www.ingenic.com

CONTENTS

1 Overview of XBurst® ISA

XBurst® CPU's Instruction Set Architecture (ISA) is fully compatible with MIPS32 ISA. For more on the MIPS32 ISA, consult the following documentations:

- MD00082-2B-MIPS32INT-AFP-03.50 provides an introduction to the MIPS32 Architecture
- MD00086-2B-MIPS32BIS-AFP-03.51 provides detailed descriptions of the MIPS32 instruction set
- MD00090-2B-MIPS32PRA-AFP-03.50 defines the behavior of the privileged resources

The SIMD extensions introduced into the XBurst® ISA as the enhanced MIPS32 ISA is called MIPS extension/enhanced Unit 2(MXU2). The MXU2 instruction set is encoded with field codes which are available to licensed MIPS partner. The MXU2 floating-point implementation is compliant with the IEEE Standard for Floating-Point Arithmetic 754-2008. All standard floating-point operations are provided for 32-bit and 64-bit floating-point data.

XBurst® MXU2 instructions set is designed by Ingenic to address the need by video, graphical, image, signal processing which has inherent parallel computation feature. This document provides detailed description of XBurst® MXU2 instruction set.

2 MXU2 Programming model

This chapter describes MXU2 Programming model. This chapter includes the following sections:

2.1 MXU2 Data Formats

The MXU2 instructions support the following data type:

- 8-bit, 16-bit, 32-bit, and 64-bit signed and unsigned integers
- 16-bit Q15, 32-bit Q31 fixed-point
- 32-bit single-precision and 64-bit double-precision floating point
- half-precision floating point conversion

2.2 MXU2 Register File

In MXU2, a vector purpose registers (VPR) is composed of thirty-two 128-bit general purpose registers vr0 ~ vr31.

2.3 MXU2 control register

There are two control registers that can be accessed through CTCMXU and CFCMXU instructions. They are:

MIR - MXU2 implementation and revision register

MCSR - MXU2 control and status register

2.3.1 MXU Implementation Register (MIR, register 0)

The MIR Register is a 32-bit read-only register.

MIR

Bit	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
	Reserved																Processor ID								version							

Table 2.1 MXU_MIR Register Field Description

Bits	Name	Description	R/W
31:16	Reserved	Writing has no effect, read as zero.	R
15:8	Processor ID	Processor ID number	R
7:0	Version	Version number.	R

2.3.2 MXU Control and Status Register (MCSR, register 31)

MCSR

Bit	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
	Reserved							FS	Reserved						Causes						Enables					Flags					RM		

Table 2.2 MXU_MCSR Register Field Description

Bits	Name	Description	R/W
31:25	Reserved	Writing has no effect, read as zero.	R
24	FS	Flush to zero for denormalized number result. 0: do not flush to zero; 1: flush to zero.	RW
23:18	Reserved	Writing has no effect, read as zero.	R
17:12	Cause	Cause bits. These bits indicate the exception conditions arise during the execution of FPU arithmetic instructions. Setting 1 if corresponding exception condition arises otherwise setting 0.	RW
11:7	Enables	Enable exception (in IEEE754, enable trap) bits. The exception shall occur when both enable bit and corresponding cause bit are set during the execution of an arithmetic FPU instruction.	RW
6:2	Flags	Flag bits. When an arithmetic FPU operation arises an exception condition but does not trigger exception due to corresponding Enable bit is set value 0, then the corresponding bit in the Flag field is set value 1, while the others remains unchanged. The value of Flag field can last until software explicitly modifies it.	RW
1:0	RM	Round mode. 0: round to nearest. 1: round toward zero. 2: round toward $+\infty$. 3: round toward $-\infty$.	RW

2.4 Exceptions

MXU2 instruction can generate the following exceptions:

Reserved Instruction, if bit Config2.C2 (CP0 Register 16, Select 2, bit 6) is not set, the ExcCode field of CP0 Cause will set to 0xa.

Coprocessor Unusable, if bit Status.CU2 (CP0 Register 12, Select 0, bit 30) is not set, the ExcCode field of CP0 Cause will set to 0xb and CE field will set to 0x2 to indicate Coprocessor 2.

MXU Floating-Point Exception, an operand exception signaled by the MXU floating-point instruction and the ExcCode field of CP0 Cause will set to 0x10.

Address Error, an aligned load or aligned store to an address that is not naturally aligned for the data

item causes an Address Error exception, the ExcCode field of CP0 Cause will set to 0x4(load) or 0x5(store).

Table 2.3 MXU2 Exception Code Values

Mnemonic	ExcCode		Description
	Decimal	Hexadecimal	
AdEL	4	0x04	Address error exception(load)
AdES	5	0x05	Address error exception(store)
RI	10	0x0a	Reserved Instruction exception
CpU	11	0x0b	Coprocessor Unusable exception
MFPE	16	0x10	MXU floating-point exception

3 MXU2 Instruction Set

The MXU2 is implemented via Coprocessor2. All of instructions are encoded in the COP2 or SPECIAL2 major opcode space.

This chapter describes MXU2 Instruction. This chapter includes the following sections:

3.1 Instruction Summary

The MXU2 assembly language coding uses the following syntax elements:

- fmt: destination data format, which could be a byte(16B), halfword(8H) , word(4W), doubleword(2D), or the vector(1Q) itself.
- vrd: destination vector register
- vrs, vrt, vrr: source vector register.
- rs, rd: general purpose register(GPR).
- fs, fd: floating point register.
- mcsrs, mcsrd: MXU2 control register.
- imm :immediate value valid.
- offset: is sign-extended and added to the contents of the base register to form an effective address.
- base: the contents of GPR base.
- index: the contents of GPR index.
- vrs[imm], vrd[imm]:vector register element of index imm is a valid index value for elements of data format fmt. Table 3.1 show the element index valid values.

Table 3.1 Valid Element Index Values

Data Format	Element Index
Byte	n=0,...,15
Halfword	n=0,...,7
Word	n=0,...,3
Doubleword	n=0,1

3.2 Macro Functions of Instruction Description

This section defines some useful macro functions that will be frequently referenced in later chapters for instruction description.

- **Common**

Table 3.2 common macro functions

Name	Description
VPR[x]	vector operand register x
MLEN	128
esize	B(8),H(16),W(32),D(64)
X'Baa	X bits, aa is bit string
{X'Baa,Y'Bbb}	(X+Y)bits, return bit string
N{X'Baa}	(N*X)bits, return bit string
op[esize,i]	return op[(i+1)*esize-1, i*esize]
=	Assignment
!=	Logical inequality
==	Logical equality
+ - * /	Arithmetic
?:	Conditional
&&	Logical and
||	Logical or
~	Bitwise negation
&	Bitwise and
|	Bitwise inclusive or
^	Bitwise exclusive or

- **exception**

  MXU floating point exception (MFPE) 0x10

- **check_cop2_enable()**

```
if !config1.C2
        signalexception(RI)
elif config1.C2 && !Status.CU2:
        Cause.CE=2'B10
        signalexception(CpU)
```

- **check_fp_exception(bit [4:0] cause)**

  check_fp_exception(bit [4:0] cause):

  ```
  if MCSR.enables & cause:
      signalexception(MFPE)
  ```

- **abs(x)**

  Get the absolution of x.

  return 0-x

- **sats(x,size)**

  Represents the result of an operation as a signed value.

- **satu(x,size)**

  Represents the result of an operation as a unsigned value.

- **aves(x,y,round)**

  ```
  t=(x>>1)+(y>>1)
  if((x&0x1) || (y&0x1))
    return t+1
  return t;
  ```

- **aveu(x,y,round)**

  ```
  if((x&0x1) && (y&0x1))
      return (x>>1)+(y>>1)+round;
  else
      return(a>>1)+(b>>1);
  ```

- **loc(x,n)**

  ```
  t=0;
  for(k=n-1;k>=0;k--){
    t=x>>k;
    t=t&0x1;
    if(t==0) break;
  }
  return (n-k)
  ```

- **lzc(x,n)**

  ```
  t=0;
  for(k=n-1;k>=0;k--){
    t=x>>k;
    t=t&0x1;
    if(t==1) break;
  }
  return (n-k)
  ```

- **bcnt(x,n)**

  ```
  count=0;
  t=0;
  for(k=n-1;k>=0;k--){
      t=x>>k;
      t=t&0x1;
  ```

```
        if(t==0x1) count++;
    }
    return count;
```

- **fpadd(x,y)**
  return the result of adding the floating-point values x and y. The operation is performed according to the IEEE standard for Floating-Point Arithmetic 754.
- **fpsub(x,y)**
  return the result of subtract the floating-point values x and y. The operation is performed according to the IEEE standard for Floating-Point Arithmetic 754.
- **fpmul(x,y)**
  return the result of multiplied the floating-point values x and y. The operation is performed according to the IEEE standard for Floating-Point Arithmetic 754.
- **fpdiv(x,y)**
  return the result of division the floating-point values x and y. The operation is performed according to the IEEE standard for Floating-Point Arithmetic 754.
- **fpsqrt(x)**
  return the result of square roots the floating-point values x . The operation is performed according to the IEEE standard for Floating-Point Arithmetic 754.
- **fpmadd(x,y,z)**
  return the result of multiply-add the floating-point values x,y,z. The operation is performed according to the IEEE standard for Floating-Point Arithmetic 754.
- **fpmsub(x,y,z)**
  return the result of multiply-subtract the floating-point values x,y,z. The operation is performed according to the IEEE standard for Floating-Point Arithmetic 754.
- **fpabs(x)**
  return the result of absolute the floating-point values x. The operation is performed according to the IEEE standard for Floating-Point Arithmetic 754.
- **fpmax(x,y)**
  return the result of maximum the floating-point values x and y. The operation is performed according to the IEEE standard for Floating-Point Arithmetic 754.
- **fpmin(x,y)**
  return the result of minimum the floating-point values x and y. The operation is performed according to the IEEE standard for Floating-Point Arithmetic 754.
- **fpclass(x,y)**
  return the result of class mask the floating-point values x and y. The operation is performed according to the IEEE standard for Floating-Point Arithmetic 754.
- **fpeq(x,y)**
  if x is compareQuiteEqual to y, return all bits 1, else return all bits 0. The operation compareQuiteEqual is performed according to the IEEE standard for Floating-Point Arithmetic 754.
- **fple(x,y)**
  if x is compareQuiteLessEqual to y, return all bits 1, else return all bits 0. The operation compareQuiteLessEqual is performed according to the IEEE standard for Floating-Point

Arithmetic 754.

- **fplt(x,y)**
  if x is compareQuiteLess to y, return all bits 1, else return all bits 0. The operation compareQuiteLess is performed according to the IEEE standard for Floating-Point Arithmetic 754.
- **fpor(x,y)**
  if x and y both are compareQuiteOrdered, return all bits 1, else return all bits 0. The operation compareQuiteOrdered is performed according to the IEEE standard for Floating-Point Arithmetic 754.
- **single_to_half(x)**
  return half-precision format of single-precision.
- **double_to_single(x)**
  return single -precision format of double -precision.
- **half_to_single(x)**
  return single -precision format of half-precision.
- **single_to_double(x)**
  return double -precision format of single -precision.
- **sint_to_single(x)**
  return signed integer format of single-precision.
- **sint_to_double(x)**
  return signed integer format of double-precision.
- **uint_to_single(x)**
  return unsigned integer format of single-precision.
- **uint_to_double(x)**
  return unsigned integer format of double-precision.
- **single_to_sint(x)**
  return signed integer format of single-precision
- **double_to_sint(x)**
  return double-precision format of signed integer.
- **single_to_uint(x)**
  return single-precision format of unsigned integer.
- **double_to_uint(x)**
  return double-precision format of unsigned integer.
- **single_rto_sint(x)**
  return single-precision format of rounded signed integer.
- **double_rto_sint(x)**
  return double-precision format of rounded signed integer.
- **single_tto_sint(x)**
  return single-precision format of truncated signed integer.
- **double_tto_sint(x)**
  return double-precision format of truncated signed integer.
- **q15_to_single(x)**
  return Q15 format of single-precision.

- **q31_to_double(x)**
  return Q31 format of double-precision.
- **single_to_q15(x)**
  return single-precision format of Q15.
- **double_to_q31(x)**
  return double-precision format of Q31.
- **mulx_signed(x,y,n)**

```
BIT[n:0]       x_ext;
BIT[n:0]       y_ext;
x_ext[n]       =x[n-1]    ;
x_ext[n-1:0]=x[n-1:0] ;
y_ext[n]       =y[n-1]    ;
y_ext[n-1:0] =y[n-1:0] ;
return    (x_ext*y_ext);
```

- **qmadd(x,y,z,n)**
  return the result of multiply-add the fixed-point values x,y,z.

```
BIT [2n-1:0] p;
BIT[n:0]temp;
BIT[n:0] z_ext;
BIT[n-1:0]result;
z_ext[n]       =z[n-1];
z_ext[n-1:0]=z[n-1:0];
p=mulx_signed(x,y,n);
temp=z_ext+(p>>n-1);
if(temp[n:n-1]==2'B01){
        result[n-1]=0;
        result[n-2:0]=(n-1)'b1;
        return result
}else if(temp[n:n-1]==2'B10){
        result[n-1]=1;
        result[n-2:0]=(n-1]'b0;
        return result
}else
        return (temp[n-1:0])
```

- **qmaddr(x,y,z,n)**
  return the rounded result of multiply-add the fixed-point values x,y,z.

```
BIT [2n-1:0] p;
BIT[n:0]temp;
BIT[n:0] z_ext;
BIT[n-1:0]result;
BIT[n-2:0]    round;
round[n-2]    =1;
round[n-3:0]=0;
```

```
z_ext[n]       =z[n-1];
z_ext[n-1:0]=z[n-1:0];
p=mulx_signed(x,y,n);
temp=z_ext+(p>>n-1)+round;
if(temp[n:n-1]==2'B01){
        result[n-1]=0;
        result[n-2:0]=(n-1)'b1;
        return result
}else if(temp[n:n-1]==2'B10){
        result[n-1]=1;
        result[n-2:0]=(n-1]'b0;
        return result
}else
        return (temp[n-1:0])
```

– **qmub(x,y,z)**

return the result of multiply-subtract the fixed-point values x,y,z.

```
BIT [2n-1:0] p;
BIT[n:0]temp;
BIT[n:0] z_ext;
BIT[n-1:0]result;
z_ext[n]       =z[n-1];
z_ext[n-1:0]=z[n-1:0];
p=mulx_signed(x,y,n);
temp=z_ext-(p>>n-1);
if(temp[n:n-1]==2'B01){
        result[n-1]=0;
        result[n-2:0]=(n-1)'b1;
        return result
}else if(temp[n:n-1]==2'B10){
        result[n-1]=1;
        result[n-2:0]=(n-1]'b0;
        return result
}else
        return (temp[n-1:0])
```

– **qmubr(x,y,z,n)**

return the rounded result of multiply-subtract the fixed-point values x,y,z.

```
BIT [2n-1:0] p;
BIT[n:0]temp;
BIT[n:0] z_ext;
BIT[n-1:0]result;
BIT[n-2:0]    round;
round[n-2]    =1;
```

```
round[n-3:0]=0;
z_ext[n]       =z[n-1];
z_ext[n-1:0]=z[n-1:0];
p=mulx_signed(x,y,n);
temp=z_ext-(p>>n-1)+round;
if(temp[n:n-1]==2'B01){
        result[n-1]=0;
        result[n-2:0]=(n-1)'b1;
        return result
}else if(temp[n:n-1]==2'B10){
        result[n-1]=1;
        result[n-2:0]=(n-1]'b0;
        return result
}else
        return (temp[n-1:0])
```

– **qmul(x,y,n)**

return the result of multiply the fixed-point values x and y.

```
BIT[2n-1:0] temp;
BIT[n-1:0]    result;
if((x[n-1]==1) &&(x[n-2:0]==0)) &&((y[n-1]==1) && y[n-2:0]==0)){
        result[n-1    ]=1;
        result[n-2:0]=0;
        return result
}else{
        temp[2n-1:0]=mulx_signed(x,y,n)
        result[n-1:0] =(temp>>n-1);
        return result;
}
```

– **qmulr(x,y,n)**

return the rounded result of multiply the fixed-point values x and y.

```
BIT[2n-1:0] temp;
BIT[n-1:0]    result;
BIT[n-2:0]    round;
round[n-2]    =1;
round[n-3:0]=0;
if((x[n-1]==1) &&(x[n-2:0]==0)) &&((y[n-1]==1) && y[n-2:0]==0)){
        result[n-1    ]=1;
        result[n-2:0]=0;
        return result
}else{
        temp[2n-1:0]=mulx_signed(x,y,n)+round;
        result[n-1:0] =(temp_1>>n-1);
        return result;
```

```
}
```

3.3 List of MXU2 Instructions

Table 3.3 through Table 3.14 provide a list of instructions grouped by category. Individual instruction descriptions follow the tables.

Table 3.3 Branch Instructions

Mnemonic	Assembler Format
BNEZ16B,BNEZ8H,BNEZ4W,BNEZ2D	vrs, offset
BNEZ1Q	vrs, offset
BEQZ16B,BEQZ8H,BEQZ4W,BEQZ2D	vrs, offset
BEQZ1Q	vrs, offset

Table 3.4 Compare Instructions

Mnemonic	Assembler Format
CEQB, CEQH, CEQW, CEQD	vrd, vrs, vrt
CEQZB, CEQZH, CEQZW, CEQZD	vrd, vrs
CNEB, CNEH, CNEW, CNED	vrd, vrs, vrt
CNEZB, CNEZH, CNEZW, CNEZD	vrd, vrs
CLESB, CLESH, CLESW, CLESD	vrd, vrs, vrt
CLEUB, CLEUH, CLEUW, CLEUD	vrd, vrs, vrt
CLEZB, CLEZH, CLEZW, CLEZD	vrd, vrs
CLTSB, CLTSH, CLTSW, CLTSD	vrd, vrs, vrt
CLTUB, CLTUH, CLTUW, CLTUD	vrd, vrs, vrt
CLTZB, CLTZH, CLTZW, CLTZD	vrd, vrs

Table 3.5 Integer Arithmetic Instructions

Mnemonic	Assembler Format
ADDAB,ADDAH,ADDAW,ADDAD	vrd, vrs, vrt
ADDASB,ADDASH,ADDASW,ADDASD	vrd, vrs, vrt
ADDSSB, ADDSSH, ADDSSW, ADDSSD	vrd, vrs, vrt
ADDUUB, ADDUUH, ADDUUW, ADDUUD	vrd, vrs, vrt
ADDB, ADDH, ADDW, ADDD	vrd, vrs, vrt
SUBSAB, SUBSAH, SUBSAW, SUBSAD	vrd, vrs, vrt
SUBUAB, SUBUAH, SUBUAW, SUBUAD	vrd, vrs, vrt
SUBSSB, SUBSSH, SUBSSW, SUBSSD	vrd, vrs, vrt
SUBUUB, SUBUUH, SUBUUW, SUBUUD	vrd, vrs, vrt
SUBUSB, SUBUSH, SUBUSW, SUBUSD	vrd, vrs, vrt
SUBB, SUBH, SUBW, SUBD	vrd, vrs, vrt
AVESB, AVESH, AVESW, AVESD	vrd, vrs, vrt
AVEUB, AVEUH, AVEUW, AVEUD	vrd, vrs, vrt

AVERSB, AVERSH, AVERSW, AVERSD	vrd, vrs, vrt
AVERUB,AVERUH,AVERUW,AVERUD	vrd, vrs, vrt
DIVRSB,DIVRSH,DIVRSW,DIVRSD	vrd, vrs, vrt
DIVUB,DIVUH,DIVUW,DIVUD	vrd, vrs, vrt
MODSB,MODSH,MODSW,MODSD	vrd, vrs, vrt
MODUB,MODUH,MODUW,MODUD	vrd, vrs, vrt
MADDB,MADDH,MADDW,MADDD	vrd, vrs, vrt
MSUBB,MSUBH,MSUBW,MSUBD	vrd, vrs, vrt
MULB,MULH,MULW,MULD	vrd, vrs, vrt
MAXAB,MAXAH,MAXAW,MAXAD	vrd, vrs, vrt
MAXSB,MAXSH,MAXSW,MAXSD	vrd, vrs, vrt
MAXUB,MAXUH,MAXUW,MAXUD	vrd, vrs, vrt
MINAB,MINAH,MINAW,MINAD	vrd, vrs, vrt
MINSB,MINSH,MINSW,MINSD	vrd, vrs, vrt
MINUB,MINUH,MINUW,MINUD	vrd, vrs, vrt
SATSB, SATSH,SATSW,SATSD	vrd, vrs, imm
SATUB, SATUH,SATUW,SATUD	vrd, vrs, imm

Table 3.6 Dot Product Instructions

Mnemonic	Assembler Format
DOTPSH,DOTPSW,DOTPSD	vrd, vrs, vrt
DOTPUH,DOTPUW,DOTPUD	vrd, vrs, vrt
DADDSH,DADDSW,DADDSD	vrd, vrs, vrt
DADDUH,DADDUW,DADDUD	vrd, vrs, vrt
DSUBSH,DSUBSW,DSUBSD	vrd, vrs, vrt
DSUBUH,DSUBUW,DSUBUD	vrd, vrs, vrt

Table 3.7 Bitwise Instructions

Mnemonic	Assembler Format
LOCB,LOCH,LOCW,LOCD	vrd, vrs
LZCB,LZCH,LZCW,LZCD	vrd, vrs
BCNTB,BCNTH,BCNTW,BCNTD	vrd, vrs
ANDV	vrd, vrs, vrt
ANDIB	vrd, vrs, imm
BSELV	vrd, vrs, vrt, vrr
NORV	vrd, vrs, vrt
NORIB	vrd, vrs, imm
ORV	vrd, vrs, vrt
ORIB	vrd, vrs, imm
XORV	vrd, vrs, vrt
XORIB	vrd, vrs, imm

Table 3.8 Floating Point Arithmetic Instructions

Mnemonic	Assembler Format
FADDW,FADDD	vrd, vrs, vrt
FSUBW,FSUBD	vrd, vrs, vrt
FMULW,FMULD	vrd, vrs, vrt
FDIVW,FDIVRD	vrd, vrs, vrt
FSQRTW,FSQRTD	vrd, vrs
FMADDW,FMADDD	vrd, vrs, vrt
FMSUBW,FMSUBD	vrd, vrs, vrt
FMAXW,FMAXD	vrd, vrs, vrt
FMAXAW,FMAXAD	vrd, vrs, vrt
FMINW,FMIND	vrd, vrs, vrt
FMINAW,FMINAD	vrd, vrs, vrt
FCLASSW,FCLASSD	vrd, vrs

Table 3.9 Floating Point Compare Instructions

Mnemonic	Assembler Format
FCEQW,FCEQD	vrd, vrs, vrt
FCLEW,FCLED	vrd, vrs, vrt
FCLTW,FCLTD	vrd, vrs, vrt
FCORW,FCORD	vrd, vrs, vrt

Table 3.10 Floating Point Conversion Instructions

Mnemonic	Assembler Format
VCVTHS,VCVTSD	vrd, vrs, vrt
VCVTESH,VCVTEDS	vrd, vrs
VCVTOSH,VCVTODS	vrd, vrs
VCVTSSW,VCVTSDL	vrd, vrs
VCVTUSW,VCVTUDL	vrd, vrs
VCVTSWS,VCVTSLD	vrd, vrs
VCVTUWS,VCVTULD	vrd, vrs
VCVTRWS,VCVTRLD	vrd, vrs
VRTRUNCSWS,VRTRUNCSLD	vrd, vrs
VRTRUNCUWS,VRTRUNCULD	vrd, vrs
VCVTQESH,VCVTQEDW	vrd, vrs
VCVTQOSH,VCVTQODW	vrd, vrs
VCVTQHS,VCVTQWD	vrd, vrs, vrt

Table 3.11 Fixed-Point Multiplication Instructions

Mnemonic	Assembler Format
MADDQH,MADDQW	vrd, vrs, vrt
MADDQRH,MADDQRW	vrd, vrs, vrt
MSUBQH,MSUBQW	vrd, vrs, vrt
MSUBQRH,MSUBQRW	vrd, vrs, vrt
MULQH,MULQW	vrd, vrs, vrt
MULQRH,MULQRW	vrd, vrs, vrt

Table 3.12 Shift Instructions

Mnemonic	Assembler Format
SLLB,SLLH,SLLW,SLLD	vrd, vrs, vrt
SLLIB,SLLIH,SLLIW,SLLID	vrd, vrs, imm
SRAB,SRAH,SRAW,SRAD	vrd, vrs, vrt
SRAIB,SRAIH,SRAIW,SRAID	vrd, vrs, imm
SRARB,SRARH,SRARW,SRARD	vrd, vrs, vrt
SRARIB,SRARIH,SRARIW,SRARID	vrd, vrs, imm
SRLB,SRLH,SRLW,SRLD	vrd, vrs, vrt
SRLIB,SRLIH,SRLIW,SRLID	vrd, vrs, imm
SRLRB,SRLRH,SRLRW,SRLRD	vrd, vrs, vrt
SRLRIB,SRLRIH,SRLRIW,SRLRID	vrd, vrs, imm

Table 3.13 Element Permute Instructions

Mnemonic	Assembler Format
INSFCPUB,INSFCPUH,INSFCPUW	vrd[imm], rs
INSFFPUW,INSFFPUD	vrd[imm], fs
INSFMXUB,INSFMXUH,INSFMXUW,INSFMXUD	vrd[imm],vrs[0]
REPXB,REPXH,REPXW	vrd, vrs[rt]
REPIB,REPIH,REPIW,REPID	vrd, vrs[imm]
SHUFV	vrd, vrs, vrt, vrr

Table 3.14 Load/Store Instructions

Mnemonic	Assembler Format
LU1Q,LA1Q	vrd, offset(base)
LU1QX,LA1QX	vrd, index(base)
LIB,LIH,LIW,LID	vrd, imm
SU1Q,SA1Q	vrd, offset(base)
SU1QX,SA1QX	vrd, index(base)
MTCPUSB,MTCPUSH,MTCPUSW	rd, vrs[imm]
MTCPUUB,MTCPUUH,MTCPUUW	rd, vrs[imm]
MFCPUB,.MFCPUH,MFCPUW	vrd, rs
MTFPUW,MTFPUD	fd, vrs[imm]
MFFPUW,MFFPUD	vrd, fs
CTCMXU	mcsrd, rs
CFCMXU	rd, mcsrs

3.4 Branch

3.4.1 BNEZ<fmt>

Branch if all elements are not equal to zero

Bit	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
	011100						fmt		100			vrs					offset											101000					

Syntax:

BNEZ16B vrs, offset

BNEZ8H vrs, offset

BNEZ4W vrs, offset

BNEZ2D vrs, offset

Description：

PC-relative branch if all elements in vrs are not equal to zero. The branch instruction has a delay slot. Offset is signed count of 32-bit instructions from the PC of the delay slot. A branch should not place in the delay slot of the instruction.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [31:0] imm={20{offset[9]},offset[9:0],2'B00}}
bit cond=1
for i in MLEN/esize
    cond=cond && (op1[esize,i] != 0)
if cond:
    I:
    I+1:pc=pc+imm
```

Exceptions:

RI, CpU

3.4.2 BNEZ1Q

Branch if vector value is not equal to zero

Bit	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
	011100	00	100	vrs	offset	101001

Syntax:

BNEZ1Q vrs, offset

Description：

PC-relative branch if at least one bit in vrs is not zero, The branch instruction has a delay slot. Offset is signed count of 32-bit instructions from the PC of the delay slot. A branch should not place in the delay slot of the instruction.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [31:0] imm={20{offset[9]},offset[9:0],2'B00}
bit cond=(op1 != 0)
if cond:
    I:
    I+1:pc=pc+imm
```

Exceptions:

RI, CpU

3.4.3 BEQZ<fmt>

Branch if at least one element is to zero

Bit	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
	011100	fmt	000	vrs	offset 101000

Syntax:

BEQZ16B vrs, offset

BEQZ8H vrs, offset

BEQZ4W vrs, offset

BEQZ2D vrs, offset

Description：

PC-relative branch if at least one element in vrs is zero,. The branch instruction has a delay slot. Offset is signed count of 32-bit instructions from the PC of the delay slot. A branch should not place in the delay slot of the instruction.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [31:0] imm={20{offset[9]},offset[9:0],2'B00}
bit cond=1
for i in MLEN/esize
    cond=cond || (op1[esize,i] == 0)
if cond:
    I:
    I+1:pc=pc+imm
```

Exceptions:

RI, CpU

3.4.4 BEQZ1Q

Branch if vector value is equal to zero

Bit	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
	011100	00	000	vrs	offset 101001

Syntax:

BEQZ1Q vrs, offset

Description：

PC-relative branch if all bits in vrs are zero. The branch instruction has a delay slot. Offset is signed count of 32-bit instructions from the PC of the delay slot. A branch should not place in the delay slot of the instruction.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [31:0] imm={20{offset[9]},offset[9:0],2'B00}
bit cond=(op1 == 0)
if cond:
    I:
    I+1:pc=pc+imm
```

Exceptions:

RI, CpU

3.5 Compare

3.5.1 CEQ<fmt>

Compare Equal **3RINT**

Bit	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
	010010						10000					vrt					vrs					vrd					1010				fmt	

Syntax:

CEQB vrd, vrs, vrt

CEQH vrd, vrs, vrt

CEQW vrd, vrs, vrt

CEQD vrd, vrs, vrt

Description：

Take each element in vrs vector register, and compare it with the corresponding elements of vrt vector register. If they are equal .the corresponding element in the vrd is set to all ones. otherwise it is set to all zeros.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd] [esize,i]= esize {op1[esize,i] == op2[esize,i]}
```

Exceptions:

RI, CpU

3.5.2 CEQZ<fmt>

Compare Equal to Zero **2RINT**

Bit	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
	010010	11110	00000	vrs	vrd	0000	fmt

Syntax:

CEQZB vrd, vrs

CEQZH vrd, vrs

CEQZW vrd, vrs

CEQZD vrd, vrs

Description：

Take each element in vrs vector register, and compare it with zero. If vrs vector register elements is equal to zero, the corresponding element in the vrd is set to all ones. otherwise it is set to all zeros.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/esize
    VRP[vrd][esize,i]= esize{op1[esize,i] == 0}
```

Exceptions:

RI, CpU

3.5.3 CNE<fmt>

Compare Not Equal **3RINT**

Bit	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
	010010	10000	vrt	vrs	vrd	1011	fmt

Syntax:

CNEB vrd, vrs, vrt

CNEH vrd, vrs, vrt

CNEW vrd, vrs, vrt

CNED vrd, vrs, vrt

Description：

Take each element in vrs vector register, and compare it with the corresponding elements of vrt vector register. If they are not equal. the corresponding element in the vrd is set to all ones. otherwise it is set to all zeros.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= esize{op1[esize,i] != op2[esize,i]}
```

Exceptions:

RI, CpU

3.5.4 CNEZ<fmt>

Compare Not Equal to Zero **2RINT**

Bit	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
	010010	11110	00000	vrs	vrd	0001	fmt

Syntax:

CNEZB vrd, vrs

CNEZH vrd, vrs

CNEZW vrd, vrs

CNEZD vrd, vrs

Description：

Take each element in vrs vector register, and compare it with zero. If vrs vector register elements is not equal to zero, the corresponding element in the vrd is set to all ones, otherwise it is set to all zeros.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/esize
    VRP [vrd][esize,i]= esize{op1[esize,i] != 0}
```

Exceptions:

RI, CpU

3.5.5 CLES<fmt>

Compare Less-Than-or-Equal Signed **3RINT**

Bit	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
	010010	10000	vrt	vrs	vrd	1110	fmt

Syntax:

CLESB vrd, vrs, vrt

CLESH vrd, vrs, vrt

CLESW vrd, vrs, vrt

CLESD vrd, vrs, vrt

Description：

Take each element in vrs vector register, and compare it with the corresponding elements of vrt vector register. If they are signed less than or equal. the corresponding element in the vrd is set to all ones. otherwise it is set to all zeros.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= esize{signed(op1[esize,i]) <= signed(op2[esize,i])}
```

Exceptions:

RI, CpU

3.5.6 CLEU<fmt>

Compare Less-Than-or-Equal Unsigned **3RINT**

Bit	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
	010010	10000	vrt	vrs	vrd	1111	fmt

Syntax:

CLEUB vrd, vrs, vrt

CLEUH vrd, vrs, vrt

CLEUW vrd, vrs, vrt

CLEUD vrd, vrs, vrt

Description：

Take each element in vrs vector register, and compare it with the corresponding elements of vrt vector register. If they are unsigned less than or equal. the corresponding element in the vrd is set to all ones. otherwise it is set to all zeros.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i] = esize{unsigned(op1[esize,i]) <= unsigned(op2[esize,i])}
```

Exceptions:

RI, CpU

3.5.7 CLEZ<fmt>

Compare Less-Than-or-Equal to Zero **2RINT**

Bit	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
	010010	11110	00000	vrs	vrd	0011	fmt

Syntax:

CLEZB vrd, vrs

CLEZH vrd, vrs

CLEZW vrd, vrs

CLEZD vrd, vrs

Description：

Take each element in vrs vector register, and compare it with zero. If vrs vector register elements is less than or equal to zero, the corresponding element in the vrd is set to all ones. otherwise it is set to all zeros.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/esize
    VRP[vrd][esize,i]= esize{signed(op1[esize,i]) <= 0}
```

Exceptions:

RI, CpU

3.5.8 CLTS<fmt>

Compare Less-Than Signed **3RINT**

Bit	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
	010010	10000	vrt	vrs	vrd	1100	fmt

Syntax:

CLTSB vrd, vrs, vrt

CLTSH vrd, vrs, vrt

CLTSW vrd, vrs, vrt

CLTSD vrd, vrs, vrt

Description：

Take each element in vrs vector register, and compare it with the corresponding elements of vrt vector register. If they are signed less than . the corresponding element in the vrd is set to all ones. otherwise it is set to all zeros.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= esize{signed(op1[esize,i]) < signed(op2[esize,i])}
```

Exceptions:

RI, CpU

3.5.9 CLTU<fmt>

Compare Less-Than Unsigned **3RINT**

Bit	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
	010010	10000	vrt	vrs	vrd	1101	fmt

Syntax:

CLTUB vrd, vrs, vrt

CLTUH vrd, vrs, vrt

CLTUW vrd, vrs, vrt

CLTUD vrd, vrs, vrt

Description：

Take each element in vrs vector register, and compare it with the corresponding elements of vrt vector register. If they are unsigned less than. the corresponding element in the vrd is set to all ones. otherwise it is set to all zeros.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= esize{unsigned(op1[esize,i]) < unsigned(op2[esize,i])}
```

Exceptions:

RI, CpU

3.5.10 CLTZ<fmt>

Compare Less-Than to Zero **2RINT**

Bit	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
	010010	11110	00000	vrs	vrd	0010	fmt

Syntax:

CLTZB vrd, vrs

CLTZH vrd, vrs

CLTZW vrd, vrs

CLTZD vrd, vrs

Description：

Take each element in vrs vector register, and compare it with zero. If vrs vector register elements is signed less than zero, the corresponding element in the vrd is set to all ones. otherwise it is set to all zeros.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/esize
    VRP[vrd][esize,i]= esize{signed(op1[esize,i]) < 0}
```

Exceptions:

RI, CpU

3.6 Integer Arithmetic

3.6.1 ADDA<fmt>

Add Absolute Values **3RINT**

Bit	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
	010010	10001	vrt	vrs	vrd	0000	fmt

Syntax:

ADDAB vrd, vrs, vrt

ADDAH vrd, vrs, vrt

ADDAW vrd, vrs, vrt

ADDAD vrd, vrs, vrt

Description：

The absolute values of the elements in vrt vector register are added to the absolute values of the corresponding elements in vrs vector register. And place the result into the corresponding elements of the vrd vector register.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]=abs(op1[esize,i] )+ abs(op2[esize,i])
```

Exceptions:

RI, CpU

3.6.2 ADDAS<fmt>

Saturated Add Absolute Values **3RINT**

Bit	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
	010010	10001	vrt	vrs	vrd	0010	fmt

Syntax:

ADDASB vrd, vrs, vrt

ADDASH vrd, vrs, vrt

ADDASW vrd, vrs, vrt

ADDASD vrd, vrs, vrt

Description：

The absolute values of the elements in vrt vector register are added to the absolute values of the corresponding elements in vrs vector register. And accumulate the signed saturated values of results into the corresponding elements of the vrd vector register.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i] = sats(abs(op1[esize,i]) + abs(op2[esize,i]))
```

Exceptions:

RI, CpU

3.6.3 ADDSS<fmt>

Signed Saturate Add **3RINT**

Bit	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
	010010	10001	vrt	vrs	vrd	0100	fmt

Syntax:

ADDSSB vrd, vrs, vrt

ADDSSH vrd, vrs, vrt

ADDSSW vrd, vrs, vrt

ADDSSD vrd, vrs, vrt

Description：

The elements in vrt vector register are added to the elements in vrs vector register. The result is perform signed arithmetic and accumulate signed saturated, then writing the result into the corresponding elements of the vrd vector register.

Operation：

```
check_cop2_enable();
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]=sats(signed(op1[esize,i]) + signed(op2[esize,i]))
```

Exceptions:

RI, CpU

3.6.4 ADDUU<fmt>

Unsigned Saturate Add **3RINT**

Bit	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
	010010	10001	vrt	vrs	vrd	0110	fmt

Syntax:

ADDUUB vrd, vrs, vrt

ADDUUH vrd, vrs, vrt

ADDUUW vrd, vrs, vrt

ADDUUD vrd, vrs, vrt

Description：

The elements in vrt vector register are added to the elements in vrs vector register. The result is perform unsigned arithmetic and accumulate unsigned saturated, then writing the result into the corresponding elements of the vrd vector register.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]=satu(unsigned(op1[esize,i]) + unsigned(op2[esize,i]))
```

Exceptions:

RI, CpU

3.6.5 ADD<fmt>

Add						3RINT
Bit 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		
010010	10001	vrt	vrs	vrd	1000	fmt

Syntax:

ADDB vrd, vrs, vrt

ADDH vrd, vrs, vrt

ADDW vrd, vrs, vrt

ADDD vrd, vrs, vrt

Description：

The elements value in vrt are added to the elements value in vrs. The result is written to vrd..

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= op1[esize,i] + op2[esize,i]
```

Exceptions:

RI, CpU

3.6.6 SUBSA<fmt>

Absolute Value of Signed Subtract **3RINT**

Bit	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
	010010	10001	vrt	vrs	vrd	0001	fmt

Syntax:

SUBSAB vrd, vrs, vrt

SUBSAH vrd, vrs, vrt

SUBSAW vrd, vrs, vrt

SUBSAD vrd, vrs, vrt

Description：

The signed values of the elements in vrt vector register are subtracted from signed values of the corresponding elements in vrs vector register. And accumulate the absolute values of results into the corresponding elements of the vrd vector register.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]=abs(signed(op1[esize,i]) - signed(op2[esize,i]))
```

Exceptions:

RI, CpU

3.6.7 SUBUA<fmt>

Absolute Values of Unsigned Subtract **3RINT**

Bit	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
	010010	10001	vrt	vrs	vrd	0011	fmt

Syntax:

SUBUAB vrd, vrs, vrt

SUBUAH vrd, vrs, vrt

SUBUAW vrd, vrs, vrt

SUBUAD vrd, vrs, vrt

Description：

The unsigned values of the elements in vrt vector register are subtracted from unsigned values of the corresponding elements in vrs vector register. And accumulate the absolute values of results into the corresponding elements of the vrd vector register.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]=abs(unsigned(op1[esize,i]) - unsigned(op2[esize,i]))
```

Exceptions:

RI, CpU

3.6.8 SUBSS<fmt>

Signed Saturated Signed Subtract **3RINT**

Bit	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
	010010	10001	vrt	vrs	vrd	0101	fmt

Syntax:

SUBSSB vrd, vrs, vrt

SUBSSH vrd, vrs, vrt

SUBSSW vrd, vrs, vrt

SUBSSD vrd, vrs, vrt

Description：

The elements in vrt vector register are subtract from the the elements in vrs vector register. The result is perform signed arithmetic and accumulate signed saturated, then writing the result into the corresponding elements of the vrd vector register.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]=sats(signed(op1[esize,i]) - signed(op2[esize,i]))
```

Exceptions:

RI, CpU

3.6.9 SUBUU<fmt>

Unsigned Saturate Unsigned Subtract **3RINT**

Bit	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
	010010	10001	vrt	vrs	vrd	0111	fmt

Syntax:

SUBUUB vrd, vrs, vrt

SUBUUH vrd, vrs, vrt

SUBUUW vrd, vrs, vrt

SUBUUD vrd, vrs, vrt

Description：

The elements in vrt vector register are subtract from the the elements in vrs vector register. The result is perform unsigned arithmetic and accumulate signed saturated, then writing the result into the corresponding elements of the vrd vector register.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]=satu(unsigned(op1[esize,i]) - unsigned(op2[esize,i]))
```

Exceptions:

RI, CpU

3.6.10 SUBUS<fmt>

Unsigned Saturated Signed Subtract **3RINT**

Bit	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
	010010	10001	vrt	vrs	vrd	1001	fmt

Syntax:

SUBUSB vrd, vrs, vrt

SUBUSH vrd, vrs, vrt

SUBUSW vrd, vrs, vrt

SUBUSD vrd, vrs, vrt

Description：

The unsigned values of the elements in vrt vector register are subtract from the unsigned values of the corresponding elements in vrs vector register. And accumulate the signed saturated values of results into the corresponding elements of the vrd vector register.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]=sats(unsigned(op1[esize,i]) - unsigned(op2[esize,i]))
```

Exceptions:

RI, CpU

3.6.11 SUB<fmt>

Sub						3RINT
Bit 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		
010010	10001	vrt	vrs	vrd	1011	fmt

Syntax:

SUBB vrd, vrs, vrt

SUBH vrd, vrs, vrt

SUBW vrd, vrs, vrt

SUBD vrd, vrs, vrt

Description：

The elements in vrt are subtracted from the elements in vrs. The result is written to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= op1[esize,i] - op2[esize,i]
```

Exceptions:

RI, CpU

3.6.12 AVES<fmt>

Signed Average **3RINT**

Bit	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
	010010	10001	vrt	vrs	vrd	1100	fmt

Syntax:

AVESB vrd, vrs, vrt

AVESH vrd, vrs, vrt

AVESW vrd, vrs, vrt

AVESD vrd, vrs, vrt

Description：

The elements in vrt are add to the elements in vrs. The addition is done signed with full precision, Signed division by 2(or arithmetic shift right by one bit) is performed before writing the result to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= aves(op1[esize,i] , op2[esize,i], 0)
```

Exceptions:

RI, CpU

3.6.13 AVEU<fmt>

Unsigned Average **3RINT**

Bit	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
	010010	10001	vrt	vrs	vrd	1110	fmt

Syntax:

AVEUB vrd, vrs, vrt

AVEUH vrd, vrs, vrt

AVEUW vrd, vrs, vrt

AVEUD vrd, vrs, vrt

Description：

The elements in vrt are add to the elements in vrs. The addition is done unsigned with full precision, Unsigned division by 2(or logic shift right by one bit) is performed before writing the result to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= aveu(op1[esize,i] , op2[esize,i], 0)
```

Exceptions:

RI, CpU

3.6.14 AVERS<fmt>

Signed Average with Rounding **3RINT**

Bit	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
	010010	10001	vrt	vrs	vrd	1101	fmt

Syntax:

AVERSB vrd, vrs, vrt

AVERSH vrd, vrs, vrt

AVERSW vrd, vrs, vrt

AVERSD vrd, vrs, vrt

Description：

The elements in vrt are add to the elements in vrs. The addition of the elements plus 1(for rounding) is done signed with full precision, Signed division by 2(or arithmetic shift right by one bit) is performed before writing the result to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= aves(op1[esize,i] , op2[esize,i], 1)
```

Exceptions:

RI, CpU

3.6.15 AVERU<fmt>

Unsigned Average with Rounding **3RINT**

Bit	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
	010010	10001	vrt	vrs	vrd	1111	fmt

Syntax:

AVERUB vrd, vrs, vrt

AVERUH vrd, vrs, vrt

AVERUW vrd, vrs, vrt

AVERUD vrd, vrs, vrt

Description：

The elements in vrt are add to the elements in vrs. The addition of the elements plus 1(for rounding) iis done unsigned with full precision, Unsigned division by 2(or logic shift right by one bit) is performed before writing the result to vrd

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= aveu(op1[esize,i] , op2[esize,i], 1)
```

Exceptions:

RI, CpU

3.6.16 DIVS<fmt>

Signed Divide **3RINT**

Bit	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
	010010	10010	vrt	vrs	vrd	0000	fmt

Syntax:

DIVSB vrd, vrs, vrt

DIVSH vrd, vrs, vrt

DIVSW vrd, vrs, vrt

DIVSD vrd, vrs, vrt

Description：

The signed integer element value of vrt vector register are divided by the corresponding signed integer element value of vrs vector register. And place the result into the corresponding elements of the vrd vector register

If a divisor element of vector vrt is zero, the result value is UNPREDICTABLE.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= signed(op1[esize,i]) / signed(op2[esize,i])
```

Exceptions:

RI, CpU

3.6.17 DIVU<fmt>

Unsigned Divied **3RINT**

Bit	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
	010010	10010	vrt	vrs	vrd	0010	fmt

Syntax:

DIVUB vrd, vrs, vrt

DIVUH vrd, vrs, vrt

DIVUW vrd, vrs, vrt

DIVUD vrd, vrs, vrt

Description：

The unsigned integer element value of vrt vector register are divived by the corresponding unsigned integer element value of vrs vector register. And place the result into the corresponding elements of the vrd vector register

If a divisor element of vector vrt is zero, the result value is UNPREDICTABLE.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= unsigned(op1[esize,i]) / unsigned(op2[esize,i])
```

Exceptions:

RI, CpU

3.6.18 MODS<fmt>

Signed Remainder **3RINT**

Bit	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
	010010	10010	vrt	vrs	vrd	0100	fmt

Syntax:

MODSB vrd, vrs, vrt

MODSH vrd, vrs, vrt

MODSW vrd, vrs, vrt

MODSD vrd, vrs, vrt

Description：

The signed integer elements in vrs are divided by signed integer elements in vrt. The remainder of the same sign as the dividend is written to vrd. If a divisor element vector vrt is zero, the result value is UNPREDICTABLE.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= signed(op1[esize,i]) % signed(op2[esize,i])
```

Exceptions:

RI, CpU

3.6.19 MODU<fmt>

Unsigned Remainder **3RINT**

Bit	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
	010010	10010	vrt	vrs	vrd	0110	fmt

Syntax:

MODUB vrd, vrs, vrt

MODUH vrd, vrs, vrt

MODUW vrd, vrs, vrt

MODUD vrd, vrs, vrt

Description：

The unsigned integer elements in vrs are divided by unsigned integer elements in vrt. The remainder is written to vrd.If a divisor element vector vrt is zero, the result value is UNPREDICTABLE.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= unsigned(op1[esize,i]) % unsigned(op2[esize,i])
```

Exceptions:

RI, CpU

3.6.20 MADD<fmt>

Multiply-Add **3RINT**

Bit	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
	010010	10010	vrt	vrs	vrd	0011	fmt

Syntax:

MADDB vrd, vrs, vrt

MADDH vrd, vrs, vrt

MADDW vrd, vrs, vrt

MADDD vrd, vrs, vrt

Description：

The integer elements in vrt are multiplied by integer elements in vrs and added to the integer elements in vrd. The most significant half of multiplication result is discarded.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= VRP[vrd][esize,i] + op1[esize,i] * op2[esize,i]
```

Exceptions:

RI, CpU

3.6.21 MSUB<fmt>

Multiply-Subtract **3RINT**

Bit	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
	010010	10010	vrt	vrs	vrd	0101	fmt

Syntax:

MSUBB vrd, vrs, vrt

MSUBH vrd, vrs, vrt

MSUBW vrd, vrs, vrt

MSUBD vrd, vrs, vrt

Description：

The integer elements in vrt are multiplied by integer elements in vrs and subtracted from the integer elements in vrd. The most significant half of multiplication result is discarded.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= VRP[vrd][esize,i] - op1[esize,i] * op2[esize,i]
```

Exceptions:

RI, CpU

3.6.22 MUL<fmt>

Multiply **3RINT**

Bit	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
	010010	10010	vrt	vrs	vrd	0001	fmt

Syntax:

MULB vrd, vrs, vrt

MULH vrd, vrs, vrt

MULW vrd, vrs, vrt

MULD vrd, vrs, vrt

Description：

The integer elements in vrt are multiplied by integer elements in vrd .The result is written to vrd .The most significant half of multiplication result is discarded.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= op1[esize,i] * op2[esize,i]
```

Exceptions:

RI, CpU

3.6.23 MAXA<fmt>

Maximum of Absolute Values **3RINT**

Bit	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
	010010	10000	vrt	vrs	vrd	0000	fmt

Syntax:

MAXAB vrd, vrs, vrt

MAXAH vrd, vrs, vrt

MAXAW vrd, vrs, vrt

MAXAD vrd, vrs, vrt

Description：

Compare corresponding elements absolute value in vrs and vrt vector register, and copies the larger of the each pair into the corresponding element in vrt destination vector register.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit cond = 0
for i in MLEN/esize
    cond = abs(op1[esize,i])>abs(op2[esize,i])
    VRP[vrd][esize,i]=cond ? op1[esize,i] : op2[esize,i]
```

Exceptions:

RI, CpU

3.6.24 MAXS<fmt>

Signed Maximum **3RINT**

Bit	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
	010010	10000	vrt	vrs	vrd	0010	fmt

Syntax:

MAXSB vrd, vrs, vrt

MAXSH vrd, vrs, vrt

MAXSW vrd, vrs, vrt

MAXSD vrd, vrs, vrt

Description：

Compare corresponding elements signed value in vrs and vrt vector register, and copies the larger of the each pair into the corresponding element in vrt destination vector register.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit cond=0
for i in MLEN/esize
    cond= signed(op1[esize,i])>signed(op2[esize,i])
    VRP[vrd][esize,i]=cond ? op1[esize,i] : op2[esize,i]
```

Exceptions:

RI, CpU

3.6.25 MAXU<fmt>

Unsigned Maximum **3RINT**

Bit	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
	010010	10000	vrt	vrs	vrd	0100	fmt

Syntax:

MAXUB vrd, vrs, vrt

MAXUH vrd, vrs, vrt

MAXUW vrd, vrs, vrt

MAXUD vrd, vrs, vrt

Description：

Compare corresponding elements unsigned value in vrs and vrt vector register, and copies the larger of the each pair into the corresponding element in vrt destination vector register.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit cond=0
for i in MLEN/esize
    cond= unsigned(op1[esize,i])>unsigned(op2[esize,i])
    VRP[vrd][esize,i]=cond ? op1[esize,i] : op2[esize,i]
```

Exceptions:

RI, CpU

3.6.26 MINA<fmt>

Minimum of Absolute Vaules **3RINT**

Bit	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
	010010	10000	vrt	vrs	vrd	0001	fmt

Syntax:

MINAB vrd, vrs, vrt

MINAH vrd, vrs, vrt

MINAW vrd, vrs, vrt

MINAD vrd, vrs, vrt

Description：

Compare corresponding elements absolute value in vrs and vrt vector register, and copies the smaller of the each pair into the corresponding element in vrt destination vector register.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit cond = 0
for i in MLEN/esize
    cond = abs(op1[esize,i])<abs(op2[esize,i])
    VRP[vrd][esize,i]=cond ? op1[esize,i] : op2[esize,i]
```

Exceptions:

RI, CpU

3.6.27 MINS<fmt>

Signed Minimum **3RINT**

Bit	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
	010010	10000	vrt	vrs	vrd	0011	fmt

Syntax:

MINSB vrd, vrs, vrt

MINSH vrd, vrs, vrt

MINSW vrd, vrs, vrt

MINSD vrd, vrs, vrt

Description：

Compare corresponding elements signed value in vrs and vrt vector register, and copies the smaller of the each pair into the corresponding element in vrt destination vector register.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit cond=0
for i in MLEN/esize
    cond= signed(op1[esize,i])<signed(op2[esize,i])
    VRP[vrd][esize,i]=cond ? op1[esize,i] : op2[esize,i]
```

Exceptions:

RI, CpU

3.6.28 MINU<fmt>

Unsigned Minimum **3RINT**

Bit	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
	010010	10000	vrt	vrs	vrd	0101	fmt

Syntax:

MINUB vrd, vrs, vrt

MINUH vrd, vrs, vrt

MINUW vrd, vrs, vrt

MINUD vrd, vrs, vrt

Description：

Compare corresponding elements unsigned value in vrs and vrt vector register, and copies the smaller of the each pair into the corresponding element in vrt destination vector register.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit cond=0
for i in MLEN/esize
    cond= unsigned(op1[esize,i])<unsigned(op2[esize,i])
    VRP[vrd][esize,i]=cond ? op1[esize,i] : op2[esize,i]
```

Exceptions:

RI, CpU

3.6.29 SATS<fmt>

Signed Saturate **2R6I**

Bit	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
	011100	fmt	00	imm	vrs	vrd	111000

Syntax:

SATSB vrd, vrs , imm

SATSH vrd, vrs , imm

SATSW vrd, vrs , imm

SATSD vrd, vrs , imm

Description：

Signed elements in vrs are saturated to signed values of imm+1 bits without changing the data width. The result is written to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit imm=imm[5:0]
bit [log2(esize):0] sat_width=imm[log2(esize)-1:0]+1
for i in MLEN/esize
    VRP[vrd][esize,i]=sats(op1[esize,i], sat_width)
```

Exceptions:

RI, CpU

3.6.30 SATU<fmt>

Unsigned Saturate **2R6I**

Bit	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
	011100	fmt	01	imm	vrs	vrd	111000

Syntax:

SATUB vrd, vrs , imm

SATUH vrd, vrs , imm

SATUW vrd, vrs , imm

SATUD vrd, vrs , imm

Description：

Unsigned elements in vrs are saturated to unsigned values of imm+1 bits without changing the data width. The result is written to vrd.

Operation：

```
check_cop2_enable()
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit imm=imm[5:0]
bit [log2(esize):0] sat_width=imm[log2(esize)-1:0]+1
for i in MLEN/esize
    VRP[vrd][esize,i]=satu(op1[esize,i], sat_width)
```

Exceptions:

RI, CpU

3.7 Dot Product

3.7.1 DOTPS<fmt>

Vector Signed Dot Product **3RINT**

Bit	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
	010010						10010					vrt					vrs					vrd					1000				fmt	

Syntax:

DOTPSH vrd, vrs, vrt

DOTPSW vrd, vrs, vrt

DOTPSD vrd, vrs, vrt

Description：

The signed integer elements in vrt are multiplied by signed integer elements in vrs producing a result twice the size of the input operands. The multiplication results of adjacent odd/even elements are added and stored to the destination.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [esize-1:0] p0
bit [esize-1:0] p1
for i in MLEN/esize
    p0 = signed(op1[esize/2:2*i])*signed(op2[esize/2:2*i])
    p1 = signed(op1[esize/2:2*i+1])*signed(op2[esize/2:2*i+1])
    VRP[vrd][esize,i]= p1 + p0
```

Exceptions:

RI, CpU

3.7.2 DOTPU<fmt>

Vector Unsigned Dot Product **3RINT**

Bit	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
	010010	10010	vrt	vrs	vrd	1010	fmt

Syntax:

DOTPUH vrd, vrs, vrt

DOTPUW vrd, vrs, vrt

DOTPUD vrd, vrs, vrt

Description：

The unsigned integer elements in vrt are multiplied by unsigned integer elements in vrs producing a result twice the size of the input operands. The multiplication results of adjacent odd/even elements are added and stored to the destination.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [esize-1:0] p0
bit [esize-1:0] p1
for i in MLEN/esize
    p0 = unsigned(op1[esize/2:2*i])*unsigned(op2[esize/2:2*i])
    p1 = unsigned(op1[esize/2:2*i+1])*unsigned(op2[esize/2:2*i+1])
    VRP[vrd][esize,i]= p1 + p0
```

Exceptions:

RI, CpU

3.7.3 DADDS<fmt>

Vector Signed Dot Product and Add **3RINT**

Bit	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
	010010	10010	vrt	vrs	vrd	1001	fmt

Syntax:

DADDSH vrd, vrs, vrt

DADDSW vrd, vrs, vrt

DADDSD vrd, vrs, vrt

Description：

The signed integer elements in vrt are multiplied by signed integer elements in vrs producing a result twice the size of the input operands. The multiplication results of adjacent odd/even elements are added to the integer elements in vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [esize-1:0] p0
bit [esize-1:0] p1
for i in MLEN/esize
    p0 = signed(op1[esize/2:2*i])*signed(op2[esize/2:2*i])
    p1 = signed(op1[esize/2:2*i+1])*signed(op2[esize/2:2*i+1])
    VRP[vrd][esize,i]= VRP[vrd][esize,i] + p1 + p0
```

Exceptions:

RI, CpU

3.7.4 DADDU<fmt>

Vector Unsigned Dot Product and Add **3RINT**

Bit	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
	010010	10010	vrt	vrs	vrd	1011	fmt

Syntax:

DADDUH vrd, vrs, vrt

DADDUW vrd, vrs, vrt

DADDUD vrd, vrs, vrt

Description：

The unsigned integer elements in vrt are multiplied by unsigned integer elements in vrs producing a result twice the size of the input operands. The multiplication results of adjacent odd/even elements are added to the integer elements in vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [esize-1:0] p0
bit [esize-1:0] p1
for i in MLEN/esize
    p0 = unsigned(op1[esize/2:2*i])*unsigned(op2[esize/2:2*i])
    p1 = unsigned(op1[esize/2:2*i+1])*unsigned(op2[esize/2:2*i+1])
    VRP[vrd][esize,i]= VRP[vrd][esize,i] + p1 + p0
```

Exceptions:

RI, CpU

3.7.5 DSUBS<fmt>

Vector Signed Dot Product and Subtract **3RINT**

Bit	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
	010010	10010	vrt	vrs	vrd	1101	fmt

Syntax:

DSUBSH vrd, vrs, vrt

DSUBSW vrd, vrs, vrt

DSUBSD vrd, vrs, vrt

Description：

The signed integer elements in vrt are multiplied by signed integer elements in vrs producing a signed result twice the size of the input operands. The sum of multiplication results of adjacent odd/even elements is subtract from the integer elements in vrd to a signed result.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [esize-1:0] p0
bit [esize-1:0] p1
for i in MLEN/esize
    p0 = signed(op1[esize/2:2*i])*signed(op2[esize/2:2*i])
    p1 = signed(op1[esize/2:2*i+1])*signed(op2[esize/2:2*i+1])
    VRP[vrd][esize,i]= VRP[vrd][esize,i] - (p1 + p0)
```

Exceptions:

RI, CpU

3.7.6 DSUBU<fmt>

Vector Unsigned Dot Product and Subtract **3RINT**

Bit	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
	010010	10010	vrt	vrs	vrd	1111	fmt

Syntax:

DSUBUH vrd, vrs, vrt

DSUBUW vrd, vrs, vrt

DSUBUD vrd, vrs, vrt

Description：

The unsigned integer elements in vrt are multiplied by unsigned integer elements in vrs producing a result twice the size of the input operands. The sum of multiplication results of adjacent odd/even elements is subtract from the integer elements in vrd to a signed result.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [esize-1:0] p0
bit [esize-1:0] p1
for i in MLEN/esize
    p0 = unsigned(op1[esize/2:2*i])*unsigned(op2[esize/2:2*i])
    p1 = unsigned(op1[esize/2:2*i+1])*unsigned(op2[esize/2:2*i+1])
    VRP[vrd][esize,i]= VRP[vrd][esize,i] -( p1 + p0)
```

Exceptions:

RI, CpU

3.8 Bitwise

3.8.1 LOC<fmt>

Leading One bits Count **2RINT**

Bit	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
	010010											11110					00000					vrs					vrd				0100		fmt	

Syntax:

LOCB vrd, vrs

LOCH vrd, vrs

LOCW vrd, vrs

LOCD vrd, vrs

Description：

Count the number of leading most significant one bits in vrs vector register and return the result into vrd destination vector register.

LOC differs from BCNT. For example, LOC will produce the vrs elements size when the vrs elements is zero.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/esize
    VRP[vrd][esize,i]= loc(op1[esize,i],esize)
```

Exceptions:

RI, CpU

3.8.2 LZC<fmt>

Leading Zero Bits Count **2RINT**

Bit	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
	010010	11110	00000	vrs	vrd	0101	fmt

Syntax:

LZCB vrd, vrs

LZCH vrd, vrs

LZCW vrd, vrs

LZCD vrd, vrs

Description：

Count the number of leading most significant zero bits in vrs vector register and return the result into vrd destination vector register.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/esize
    VRP[vrd][esize,i]= lzc(op1[esize,i],esize)
```

Exceptions:

RI, CpU

3.8.3 BCNT<fmt>

Population Count **2RINT**

Bit	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
	010010	11110	00000	vrs	vrd	1100	fmt

Syntax:

BCNTB vrd, vrs

BCNTH vrd, vrs

BCNTW vrd, vrs

BCNTD vrd, vrs

Description：

Count the vrs elements number of set bits, and return the result into the destination vector register vrd corresponding elements.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/esize
    VRP[vrd][esize,i]= bcnt(op1[esize,i],esize)
```

Exceptions:

RI, CpU

3.8.4 ANDV

Logical And **3RVEC**

Bit	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
	010010	10110	vrt	vrs	vrd	111000

Syntax:

ANDV vrd, vrs, vrt

Description：

A bitwise AND operation between two registers vrs and vrt ,and places the result in the destination register vrd.

Operation：

check_cop2_enable()

bit [127:0] op1=VPR[vrs]

bit [127:0] op2=VPR[vrt]

VRP[vrd] = op1 & op2

Exceptions:

RI, CpU

3.8.5 ANDIB

Immediate Logical And **2R8I**

Bit	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
	011100	00	imm	vrs	vrd	110000

Syntax:

ANDIB vrd, vrs, imm

Description：

Take each byte elements of vrs and a bitwise logical AND operation between vrs byte element and the 8-bit immediate imm ,place the result into the corresponding byte element vrd vector register.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [7:0] imm=imm[7:0]
for i in MLEN/esize
    VRP[vrd][esize,i]= op1[esize,i] & imm
```

Exceptions:

RI, CpU

3.8.6 NORV

Logical NOR **3RVEC**

Bit	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
	010010	10110	vrt	vrs	vrd	111001

Syntax:

NORV vrd, vrs, vrt

Description：

A bitwise NOR operation between two registers vrs and vrt ,and places the result in the destination register vrd.

Operation：

check_cop2_enable()

bit [127:0] op1=VPR[vrs]

bit [127:0] op2=VPR[vrt]

VRP[vrd] = ~(op1 | op2)

Exceptions:

RI, CpU

3.8.7 NORIB

Immediate Logical NOR **2R8I**

Bit	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
	opcode	fmt	imm	vrs	vrd	funct0

Syntax:

NORIB vrd, vrs, imm

Description：

Take each byte elements of vrs and a bitwise logical NOR operation between vrs byte element and the 8-bit immediate imm ,place the result into the corresponding byte element vrd vector register.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [7:0] imm=imm[7:0]
for i in MLEN/esize
    VRP[vrd][esize,i]= ~(op1[esize,i] | imm)
```

Exceptions:

RI, CpU

3.8.8 ORV

Logical OR **3RVEC**

Bit	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
	010010	10110	vrt	vrs	vrd	111010

Syntax:

ORV vrd, vrs, vrt

Description：

A bitwise OR operation between two registers vrs and vrt and places the result in the destination register vrd.

Operation：

check_cop2_enable()

bit [127:0] op1=VPR[vrs]

bit [127:0] op2=VPR[vrt]

VRP[vrd] = (op1 | op2)

Exceptions:

RI, CpU

3.8.9 ORIB

Immediate Logical OR **2R8I**

Bit	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
	opcode	fmt	imm	vrs	vrd	funct0

Syntax:

ORIB vrd, vrs, imm

Description：

Take each byte elements of vrs and a bitwise logical OR operation between vrs byte element and the 8-bit immediate imm ,place the result into the corresponding byte element vrd vector register.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [7:0] imm=imm[7:0]
for i in MLEN/esize
    VRP[vrd][esize,i]= (op1[esize,i] | imm)
```

Exceptions:

RI, CpU

3.8.10 XORV

Logical XOR **3RVEC**

Bit	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
	010010	10110	vrt	vrs	vrd	111011

Syntax:

XORV vrd, vrs, vrt

Description：

A bitwise XOR operation between two registers vrs and vrt and places the result in the destination register vrd.

Operation：

check_cop2_enable()

bit [127:0] op1=VPR[vrs]

bit [127:0] op2=VPR[vrt]

VRP[vrd] = (op1 ^ op2)

Exceptions:

RI, CpU

3.8.11 XORIB

Immediate Logical XOR **2R8I**

Bit	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
	011100	01	imm	vrs	vrd	110000

Syntax:

XORIB vrd, vrs, imm

Description：

Take each byte elements of vrs and a bitwise logical XOR operation between vrs byte element and the 8-bit immediate imm, place the result into the corresponding byte element vrd vector register.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [7:0] imm=imm[7:0]
for i in MLEN/esize
    VRP[vrd][esize,i]= (op1[esize,i] ^ imm)
```

Exceptions:

RI, CpU

3.8.12 BSELV

Bit Select **4R**

Bit	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
	011100						vrr					vrt					vrs					vrd					011001					

Syntax:

BSELV vrd, vrs, vrt, vrr

Description：

Selectively copy bits from the source vrs and vrt vector register into destination vrd vector based on the corresponding bit in vrr: if 0 copies the bit from vrs, if 1 copies the bit from vrt.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [127:0] op3=VPR[vrr]
for i in MLEN
    VRP[vrd][i]= op3[i] ? op2[i] : op1[i]
```

Exceptions:

RI, CpU

3.9 Floating Point Arithmetic

3.9.1 FADD<fmt>

Floating-Point Addition **3RFP**

Bit	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
	010010						11000					vrt					vrs					vrd					00000					fmt

Syntax:

FADDW vrd, vrs, vrt

FADDD vrd, vrs, vrt

Description：

Add corresponding floating-point elements in two vector register vrs and vrd, and place the result in the destination vector register vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= fpadd(op1[esize,i] , op2[esize,i])
```

Exceptions:

RI, CpU, MFPE

3.9.2 FSUB<fmt>

Floating-Point Subtraction **3RFP**

Bit	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
	010010	11000	vrt	vrs	vrd	00001	fmt

Syntax:

FSUBW vrd, vrs, vrt

FSUBD vrd, vrs, vrt

Description：

Subtracts the floating-point elements of vector register vrt from the corresponding floating-point elements of another vector register vrs, and place the result in the destination vector register vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= fpsub(op1[esize,i] , op2[esize,i])
```

Exceptions:

RI, CpU, MFPE

3.9.3 FMUL<fmt>

Floating-Point Multiplication **3RFP**

Bit	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
	010010	11000	vrt	vrs	vrd	00010	fmt

Syntax:

FMULW vrd, vrs, vrt

FMULD vrd, vrs, vrt

Description：

Multiplies corresponding elements in two vector register vrs and vrd and place the result in the destination vector register vrd.

The multiplication operation is defined by the IEEE standard for Floating-Point Arithmetic 754.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= fpmul(op1[esize,i] , op2[esize,i])
```

Exceptions:

RI, CpU, MFPE

3.9.4 FDIV<fmt>

Floating-Point Division **3RFP**

Bit	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
	010010	11000	vrt	vrs	vrd	00011	fmt

Syntax:

FDIVW vrd, vrs, vrt

FDIVD vrd, vrs, vrt

Description：

Divides the floating-point elements of vector register vrt by the corresponding floating-point elements of another vector register vrs and place the result in the destination vector register vrd.

The division operation is defined by the IEEE standard for Floating-Point Arithmetic 754.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= fpdiv(op1[esize,i] , op2[esize,i])
```

Exceptions:

RI, CpU, MFPE

3.9.5 FSQRT<fmt>

Floating-Point Square Root **2RFP**

Bit	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
	010010	11110	00001	vrs	vrd	00000	fmt

Syntax:

FSQRTW vrd, vrs

FSQRTD vrd, vrs

Description：

The square roots of floating-point elements in vrs are written to vrd.

The square roots operation is defined by the IEEE standard for Floating-Point Arithmetic 754.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/esize
    VRP[vrd][esize,i]= fpsqrt(op1[esize,i])
```

Exceptions:

RI, CpU, MFPE

3.9.6 FMADD<fmt>

Floating-Point Multiply-Add **3RFP**

Bit	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
	010010	11000	vrt	vrs	vrd	00100	fmt

Syntax:

FMADDW vrd, vrs, vrt

FMADDD vrd, vrs, vrt

Description：

The floating-point elements in vrt multiplied by floating-point elements in vrs are added to the floating-point elements in vrd. The operation is fused, i.e. computed as if with unbounded range and precision, rounding only once to the destination format.

The multiply add operation is defined by the IEEE standard for Floating-point Arithmetic 754.The multiplication between an infinity and a zero signals Invalid Operation exception. If the Invalid Operation exception is disabled, the result is the default quiet NaN.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [127:0] op3=VPR[vrd]
for i in MLEN/esize
    VRP[vrd][esize,i]= fpmadd(op1[esize,i] , op2[esize,i] , op3[esize,i])
```

Exceptions:

RI, CpU, MFPE

3.9.7 FMSUB<fmt>

Floating-Point Multiply-Subtract **3RFP**

Bit	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
	010010	11000	vrt	vrs	vrd	00101	fmt

Syntax:

FMSUBW vrd, vrs, vrt

FMSUBD vrd, vrs, vrt

Description：

The floating-point elements in vrt multiplied by floating-point elements in vector vrs are subtracted from the floating-point elements in vrd. The operation is fused, i.e. computed as if with unbounded range and precision, rounding only once to the destination format.

The multiply subtract operation is defined by the IEEE Standard for Floating-Point Arithmetic 754.The multiplication between an infinity and a zero signals Invalid Operation exception. If the Invalid Operation exception is disabled, the result is the default quiet NaN.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [127:0] op3=VPR[vrd]
for i in MLEN/esize
    VRP[vrd][esize,i]= fpmsub(op1[esize,i] , op2[esize,i] , op3[esize,i])
```

Exceptions:

RI, CpU, MFPE

3.9.8 FMAX<fmt>

Floating-Point Maximum **3RFP**

Bit	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
	010010	11000	vrt	vrs	vrd	01100	fmt

Syntax:

FMAXW vrd, vrs, vrt

FMAXD vrd, vrs, vrt

Description：

Compare corresponding floating-point elements value in vrs and vrt vector register, and copies the larger of the each pair into the corresponding floating-point element in vrt destination vector register. The largest value is defined by the maxNum operation in the IEEE standard for Floating-point Arithmetic 754.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= fpmax(op1[esize,i] , op2[esize,i])
```

Exceptions:

RI, CpU, MFPE

3.9.9 FMAXA<fmt>

Floating-Point Maximum Based of Absolute Values Comparison **3RFP**

Bit	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
	010010	11000	vrt	vrs	vrd	01101	fmt

Syntax:

FMAXAW vrd, vrs, vrt

FMAXAD vrd, vrs, vrt

Description：

Compare corresponding floating-point elements absolute value in vrs and vrt vector register, and copies the one with larger absolute value into the corresponding floating-point element in vrt destination vector register.

The largest absolute value is defined by the maxNumMag operation in the IEEE standard for Floating-point Arithmetic 754.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= fpabs(op1[esize,i]) >fpabs(op2[esize,i])? op1[esize,i]:op2[esize,i];
```

Exceptions:

RI, CpU, MFPE

3.9.10 FMIN<fmt>

Floating-Point Minimum **3RFP**

Bit	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
	010010	11000	vrt	vrs	vrd	01110	fmt

Syntax:

FMINW vrd, vrs, vrt

FMIND vrd, vrs, vrt

Description：

Compare corresponding floating-point elements value in vrs and vrt vector register, and copies the smaller of the each pair into the corresponding floating-point element in vrt destination vector register. The smallest value is defined by the minNum operation in the IEEE standard for Floating-point Arithmetic 754.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= fpmin(op1[esize,i] , op2[esize,i])
```

Exceptions:

RI, CpU, MFPE

3.9.11 FMINA<fmt>

Floating-Point Minimum Based on Absolute Values Comparison **3RFP**

Bit	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
	010010	11000	vrt	vrs	vrd	01111	fmt

Syntax:

FMINAW vrd, vrs, vrt

FMINAD vrd, vrs, vrt

Description：

Compare corresponding floating-point elements absolute value in vrs and vrt vector register, and copies the one with smaller absolute value into the corresponding floating-point element in vrt destination vector register.

The smallest absolute value is defined by the minNumMag operation in the IEEE standard for Floating-point Arithmetic 754.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= fpabs(op1[esize,i]) < fpabs(op2[esize,i])? op1[esize,i]:op2[esize,i];
```

Exceptions:

RI, CpU, MFPE

3.9.12 FCLASS<fmt>

Floating-Point Class Mask **2RFP**

Bit	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
	010010						11110					00001					vrs					vrd					00011					fmt

Syntax:

FCLASSW vrd, vrs,

FCLASSD vrd, vrs

Description：

Stored in each element of vrd with a bit mask reflecting the floating-point class of the corresponding element of vrs.

The mask has 10 bits as follows. Bits 0 and 1 indicate NAN values: signaling NaN(bit 0) and quiet NaN (bit 1).Bits 2,3,4,5 classify negative values: infinity(bit 2),normal(bit 3),subnormal(bit 4),and zero(bit 5).Bits 6,7,8,9 classify positive values: infinity (bit 6),normal(bit 7),subnormal(bit 8),and zero (bit 9(.

The input values and generated bit masks are not affected by the flush-to-zero bit FS in MXU2_MCSR register.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/esize
    VRP[vrd][esize,i]= fpclass(op1[esize,i])
```

Exceptions:

RI, CpU

3.10 Floating Point Compare

3.10.1 FCEQ<fmt>

Floating-Point Quiet Compare Equal **3RFP**

Bit	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
	010010	11000	vrt	vrs	Vrd	01001	fmt

Syntax:

FCEQW vrd, vrs, vrt

FCEQD vrd, vrs, vrt

Description：

Take each floating-point element in vrs vector register, and compare it with the corresponding floating-point elements of vrt vector register. If they are ordered and equal .the corresponding element in the vrd is set to all ones. Otherwise, it is set to all zeros.

The quiet compare operation is defined by the IEEE Standard for Floating-point Arithmetic 754.

The Inexact Exception is not signaled when subnormal input operands are flushed based on the flush-to-zero bit FS in MCSR register. In case of a floating-point exception, the default result has all bits set to 0.

FCEQW instruction’s operands are values in single floating-point data format and its results are values in 32-bit integer data format.

FCEQD instruction’s operands are values in double floating-point data format and its results are values in 64-bit integer data format.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= fpeq(op1[esize,i] , op2[esize,i])
```

Exceptions:

RI, CpU, MFPE

3.10.2 FCLE<fmt>

Floating-Point Quiet Compare Less or Equal **3RFP**

Bit	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
	010010	11000	vrt	vrs	vrd	01011	fmt

Syntax:

FCLEW vrd, vrs, vrt

FCLED vrd, vrs, vrt

Description：

Take each floating-point element in vrs vector register, and compare it with the corresponding elements of vrt vector register. If they are ordered and either less than or equal, the corresponding element in the vrd is set to all ones. Otherwise, it is set to all zeros.

The quiet compare operation is defined by the IEEE Standard for Floating-point Arithmetic 754.

The Inexact Exception is not signaled when subnormal input operands are flushed based on the flush-to-zero bit FS in MCSR register. In case of a floating-point exception, the default result has all bits set to 0.

FCLEW instruction's operands are values in single floating-point data format and its results are values in 32-bit integer data format.

FCLED instruction's operands are values in double floating-point data format and its results are values in 64-bit integer data format.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= fple(op1[esize,i] , op2[esize,i])
```

Exceptions:

RI, CpU, MFPE

3.10.3 FCLT<fmt>

Floating-Point Quiet Compare Less Than **3RFP**

Bit	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
	010010	11000	vrt	vrs	vrd	01010	fmt

Syntax:

FCLTW vrd, vrs, vrt

FCLTD vrd, vrs, vrt

Description：

Take each floating-point element in vrs vector register, and compare it with the corresponding elements of vrt vector register. If they are ordered and less than, the corresponding element in the vrd is set to all ones. Otherwise, it is set to all zeros.

The quiet compare operation is defined by the IEEE Standard for Floating-point Arithmetic 754.

The Inexact Exception is not signaled when subnormal input operands are flushed based on the flush-to-zero bit FS in MCSR register. In case of a floating-point exception, the default result has all bits set to 0.

FCLTW instruction's operands are values in single floating-point data format and its results are values in 32-bit integer data format.

FCLTD instruction's operands are values in double floating-point data format and its results are values in 64-bit integer data format.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= fplt(op1[esize,i] , op2[esize,i])
```

Exceptions:

RI, CpU, MFPE

3.10.4 FCOR<fmt>

Floating-Point Quiet Compare Ordered **3RFP**

Bit	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
	010010	11000	vrt	vrs	vrd	01000	fmt

Syntax:

FCORW vrd, vrs, vrt

FCORD vrd, vrs, vrt

Description：

Take each floating-point element in vrs vector register, and compare it with the corresponding floating-point elements of vrt vector register. If they are ordered (both floating-point elements value are not NaN values) .the corresponding element in the vrd is set to all ones. Otherwise, it is set to all zeros. The quiet compare operation is defined by the IEEE Standard for Floating-point Arithmetic 754.

The Inexact Exception is not signaled when subnormal input operands are flushed based on the flush-to-zero bit FS in MCSR register. In case of a floating-point exception, the default result has all bits set to 0.

FCORW instruction's operands are values in single floating-point data format and its results are values in 32-bit integer data format.

FCORD instruction's operands are values in double floating-point data format and its results are values in 64-bit integer data format.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= fpor(op1[esize,i] , op2[esize,i])
```

Exceptions:

RI, CpU, MFPE

3.11 Floating Point Conversion

3.11.1 VCVTHS

Single-Precision Convert to Halfword **3RFP**

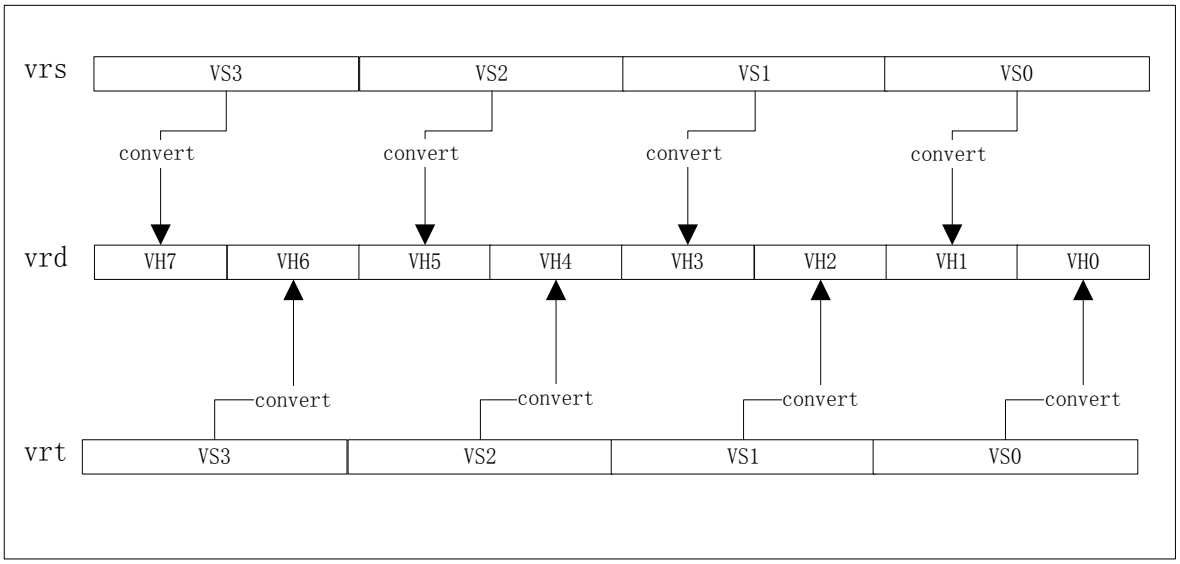
Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	010010								11000				vrt				vrs					vrd					00110					0

Syntax:

VCVTHS vrd, vrs, vrt

Description：

The single-precision floating-point elements in vrs are down-convert to 16bit interchange format, and store in vrd even elements. The single-precision floating-point elements in vrt are down-convert to 16 bit interchange format, and store in vrd odd elements.



Operation：

check_cop2_enable()

bit [127:0] op1=VPR[vrs]

bit [127:0] op2=VPR[vrt]

for i in MLEN/32:

VRP[vrd][16,2i+1]= single_to_half(op1[32,i])

VRP[vrd][16,2i]= single_to_half(op2[32,i])

Exceptions:

RI, CpU, MFPE

3.11.2 VCVTSD

Double-Precision Convert to Single-Precision **3RFP**

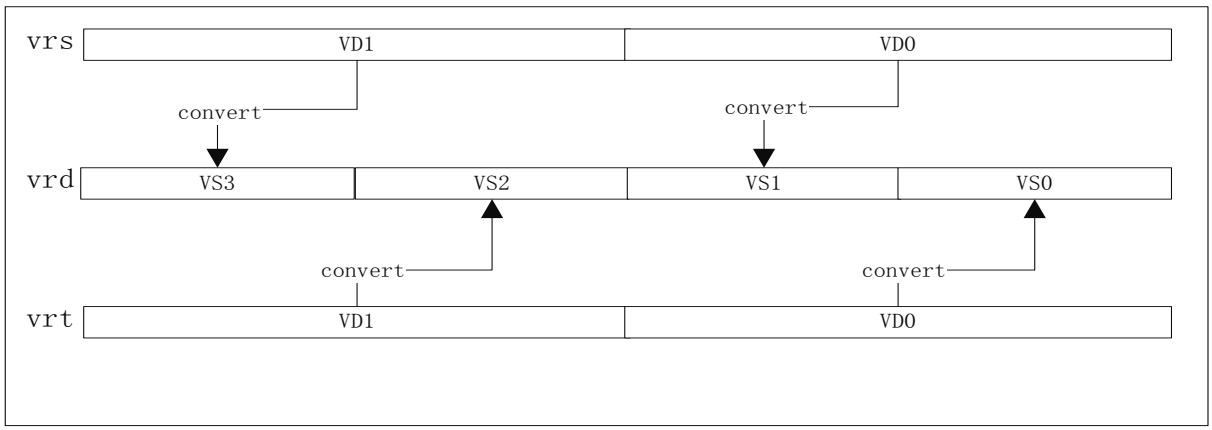
Bit	31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1	0
	010010	11000	vrt	vrs	vrd	00110	1

Syntax:

VCVTSD vrd, vrs, vrt

Description：

The double-precision floating-point elements in vrs are down-convert to single-precision floating-point, and store in vrd even elements. The double-precision floating-point elements in vrt are down-convert to single-precision floating-point, and store in vrd odd elements.



Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/64:
    VRP[vrd][32,2i+1]= double_to_single(op1[64,i])
    VRP[vrd][32,2i]= double_to_single(op2[64,i])
```

Exceptions:

RI, CpU, MFPE

3.11.3 VCVTESH

Even Halfword Convert to Single-Precision **2RFP**

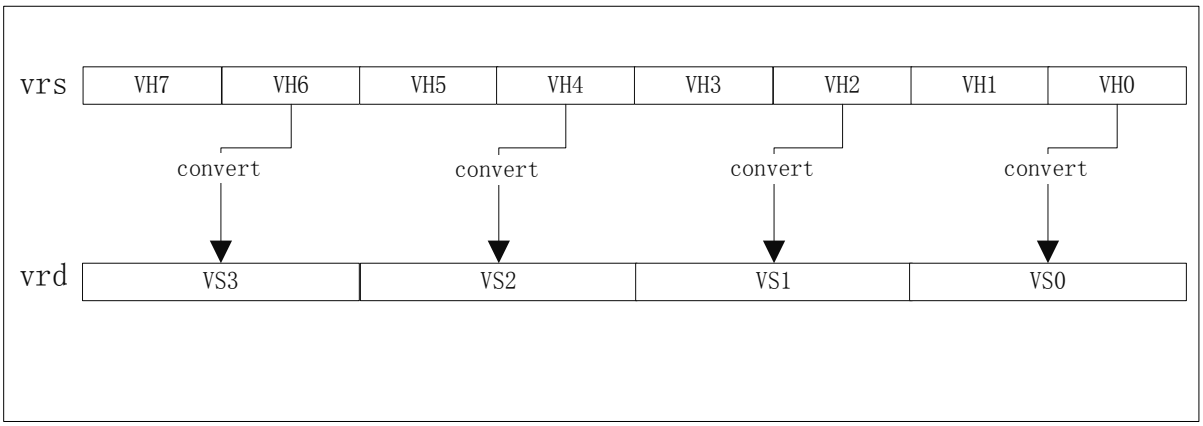
Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	010010												11110				00001						vrs				vrd				10000						0

Syntax:

VCVTESH vrd, vrs

Description：

Even elements in vrs convert to single-precision floating-point, and are written to vrd.



Operation：

check_cop2_enable()

bit [127:0] op1=VPR[vrs]

for i in MLEN/32:

VRP[vrd][32,i]= half_to_single(op1[16,2i])

Exceptions:

RI, CpU, MFPE

3.11.4 VCVTEDS

Even Single-Precision Convert to Double Precision **2RFP**

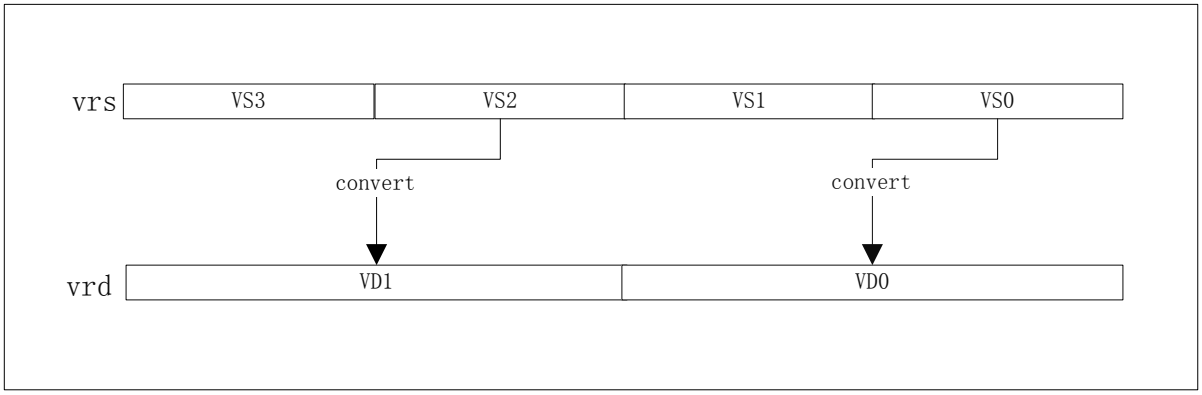
Bit	31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1	0
	010010	11110	00001	vrs	vrd	10000	1

Syntax:

VCVTEDS vrd, vrs

Description：

Even elements in vrs single-precision floating-point value convert to double-precision floating-point, and are written to vrd.



Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/64:
    VRP[vrd][64,i]= single_to_double(op1[32,2i])
```

Exceptions:

RI, CpU, MFPE

3.11.5 VCVTOSH

Odd Halfword Convert to Single-Precision **2RFP**

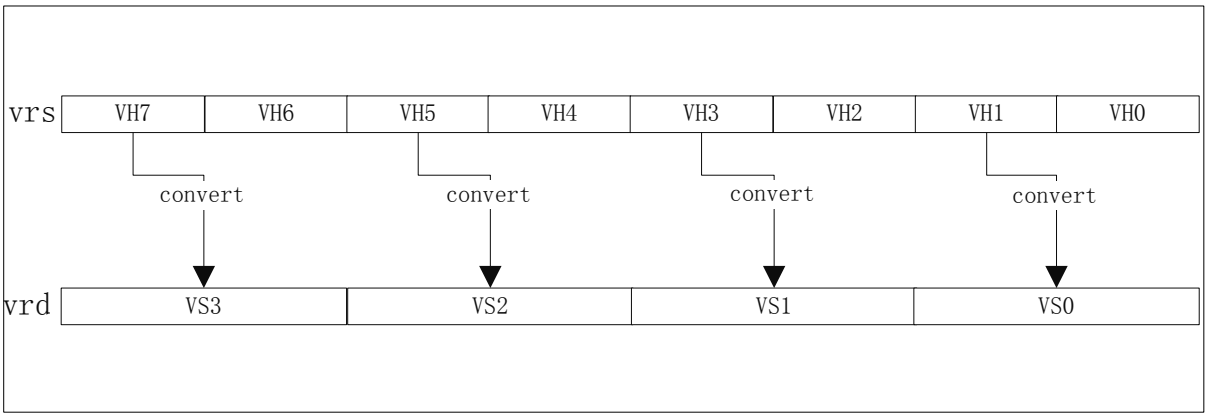
Bit	31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1	0
	010010	11110	00001	vrs	vrd	10100	0

Syntax:

VCVTOSH vrd, vrs

Description：

Odd elements in vrs convert to single-precision floating-point, and are written to vrd.



Operation：

check_cop2_enable()

bit [127:0] op1=VPR[vrs]

for i in MLEN/32:

VRP[vrd][32,i]= half_to_single(op1[16,2i+1])

Exceptions:

RI, CpU, MFPE

3.11.6 VCVTODS

Odd Single-Precision Convert to Double-Precision **2RFP**

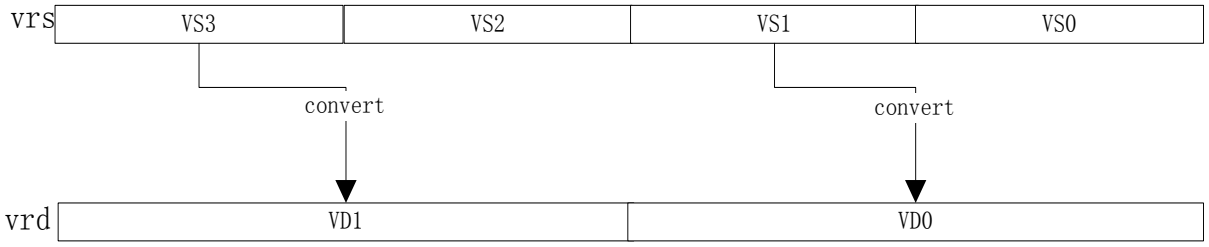
Bit	31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1	0
	010010	11110	00001	vrs	vrd	10100	1

Syntax:

VCVTLDS vrd, vrs

Description：

Odd elements in vrs single-precision floating-point value convert to double-precision floating-point, and are written to vrd.



Operation：

check_cop2_enable()

bit [127:0] op1=VPR[vrs]

for i in MLEN/64:

VRP[vrd][64,i]= single_to_double(op1[32,2i+1])

Exceptions:

RI, CpU, MFPE

3.11.7 VCVTSSW

Signed Integer Convert to Single-Precision **2RFP**

Bit	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
	010010	11110	00001	vrs	vrd	00100	0

Syntax:

VCVTSSW vrd, vrs

Description：

The signed 32-bit integer elements in vrs are converted to single-precision floating-point values. The result is written to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/32:
    VRP[vrd][32,i]= sint_to_single(op1[32,i])
```

Exceptions:

RI, CpU, MFPE

3.11.8 VCVTSDL

Signed Integer Convert to Double-Precision **2RFP**

Bit	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
	010010	11110	00001	vrs	vrd	00100	1

Syntax:

VCVTSDL vrd, vrs

Description：

The signed 64-bit integer elements in vrs are converted to double-precision floating-point values. The result is written to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/64:
    VRP[vrd][64,i]= sint_to_double(op1[64,i])
```

Exceptions:

RI, CpU, MFPE

3.11.9 VCVTUSW

Unsigned 32-bit Integer Convert to Single-Precision **2RFP**

Bit	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
	010010	11110	00001	vrs	vrd	00101	0

Syntax:

VCVTUSW vrd, vrs

Description：

The unsigned 32-bit integer elements in vrs are converted to single-precision floating-point values. The result is written to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/32:
    VRP[vrd][32,i]= uint_to_single(op1[32,i])
```

Description：

Exceptions:

RI, CpU, MFPE

3.11.10 VCVTUDL

Unsigned Integer Convert to Double-Precision **2RFP**

Bit	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
	010010	11110	00001	vrs	vrd	00101	1

Syntax:

VCVTUDL vrd, vrs

Description：

The unsigned 64-bit integer elements in vrs are converted to double-precision floating-point values. The result is written to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/64:
    VRP[vrd][64,i]= uint_to_double(op1[64,i])
```

Exceptions:

RI, CpU, MFPE

3.11.11 VCVTSWS

Single-Precision Convert to Signed Integer **2RFP**

Bit	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
	010010	11110	00001	vrs	vrd	00110	0

Syntax:

VCVTSWS vrd, vrs

Description：

The single-precision floating-point elements in vrs are rounded and convert to signed 32-bit integer values based on the rounding mode bits RM in MCSR register. The result is written to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/32:
    VRP[vrd][32,i]= single_to_sint(op1[32,i])
```

Exceptions:

RI, CpU, MFPE

3.11.12 VCVTSLD

Double-Precision Convert to Signed Integer **2RFP**

Bit	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
	010010	11110	00001	vrs	vrd	00110	1

Syntax:

VCVTSLD vrd, vrs

Description：

The double-precision floating-point elements in vrs are rounded and convert to signed 64-bit integer values based on the rounding mode bits RM in MCSR register. The result is written to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/64:
    VRP[vrd][64,i]= double_to_sint(op1[64,i])
```

Exceptions:

RI, CpU, MFPE

3.11.13 VCVTUWS

Single-Precision Convert to Unsigned Integer **2RFP**

Bit	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
	010010							11110				00001				vrs				vrd				00111						0

Syntax:

VCVTUWS vrd, vrs

Description：

The single-precision floating-point elements in vrs are rounded and convert to unsigned 32-bit integer values based on the rounding mode bits RM in MCSR register. The result is written to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/32:
    VRP[vrd][32,i]= single_to_uint(op1[32,i])
```

Exceptions:

RI, CpU, MFPE

3.11.14 VCVTULD

Single-Precision Convert to Signed Integer **2RFP**

Bit	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
	010010	11110	00001	vrs	vrd	00111	1

Syntax:

VCVTULD vrd, vrs

Description：

The double-precision floating-point elements in vrs are rounded and convert to unsigned 64-bit integer values based on the rounding mode bits RM in MCSR register. The result is written to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/64:
    VRP[vrd][64,i]= double_to_uint(op1[64,i])
```

Exceptions:

RI, CpU, MFPE

3.11.15 VCVTRWS

Single-Precision Rounding and Convert to Integer **2RFP**

Bit	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
	010010	11110	00001	vrs	vrd	01110	0

Syntax:

VCVTRWS vrd, vrs

Description：

The single-precision floating-point elements in vrs are rounded to a 32-bit integral value floating-point number based on the rounding mode bits RM in MCSR register. The result is written to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/32:
    VRP[vrd][32,i]= single_rto_ sint(op1[32,i])
```

Exceptions:

RI, CpU, MFPE

3.11.16 VCVTRLD

Double-Precision Rounding and Convert to Integer **2RFP**

Bit	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
	010010	11110	00001	vrs	vrd	01110	1

Syntax:

VCVTRLD vrd, vrs

Description：

The double-precision floating-point elements in vrs are rounded to a 64-bit integral value floating-point number based on the rounding mode bits RM in MCSR register. The result is written to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/64:
    VRP[vrd][64,i]= double_rto_ sint(op1[64,i])
```

Exceptions:

RI, CpU, MFPE

3.11.17 VTRUNCSWS

Single-Precision Truncate and Convert to Signed Integer **2RFP**

Bit	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
	010010										11110					00001				vrs				vrd					01010			0

Syntax:

VTRUNCSWS vrd, vrs

Description：

The single-precision floating-point elements in vrs are truncated ,i.e. rounded toward zero, to signed 32-bit integer values. The rounding mode bits RM in MCSR register are not used. The result is written to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/32:
    VRP[vrd][32,i]= single_tto_ sint(op1[32,i])
```

Exceptions:

RI, CpU, MFPE

3.11.18 VTRUNCSLD

Double-Precision Truncate and Convert to Signed Integer **2RFP**

Bit	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
	010010	11110	00001	vrs	vrd	01010	1

Syntax:

VTRUNCSLD vrd, vrs

Description：

The double-precision floating-point elements in vrs are truncated, i.e. rounded toward zero, to signed 64-bit integer values. The rounding mode bits RM in MCSR register are not used. The result is written to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/64:
    VRP[vrd][64,i]= double_tto_sint(op1[64,i])
```

Exceptions:

RI, CpU, MFPE

3.11.19 VTRUNCUWS

Single-Precision Truncate and Convert to Unsigned Integer **2RFP**

Bit	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
	010010	11110	00001	vrs	vrd	01011	0

Syntax:

VTRUNCUWS vrd, vrs

Description：

The single-precision floating-point elements in vrs are truncated, i.e. rounded toward zero, to unsigned 32-bit integer values. The rounding mode bits RM in MCSR register are not used. The result is written to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/32:
    VRP[vrd][32,i]= single_tto_ uint(op1[32,i])
```

Exceptions:

RI, CpU, MFPE

3.11.20 VTRUNCULD

Double-Precision Truncate and Convert to Unsigned Integer **2RFP**

Bit	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
	010010	11110	00001	vrs	vrd	01011	1

Syntax:

VTRUNCULD vrd, vrs

Description：

The double-precision floating-point elements in vrs are truncated, i.e. rounded toward zero, to unsigned 64-bit integer values. The rounding mode bits RM in MCSR register are not used. The result is written to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/64:
    VRP[vrd][64,i]= double_tto_uint(op1[64,i])
```

Exceptions:

RI, CpU, MFPE

3.11.21 VCVTQESH

Even Fixed-Point Q15 Convert to Single-Precision **2RFP**

Bit	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
	010010	11110	00001	vrs	vrd	11000	0

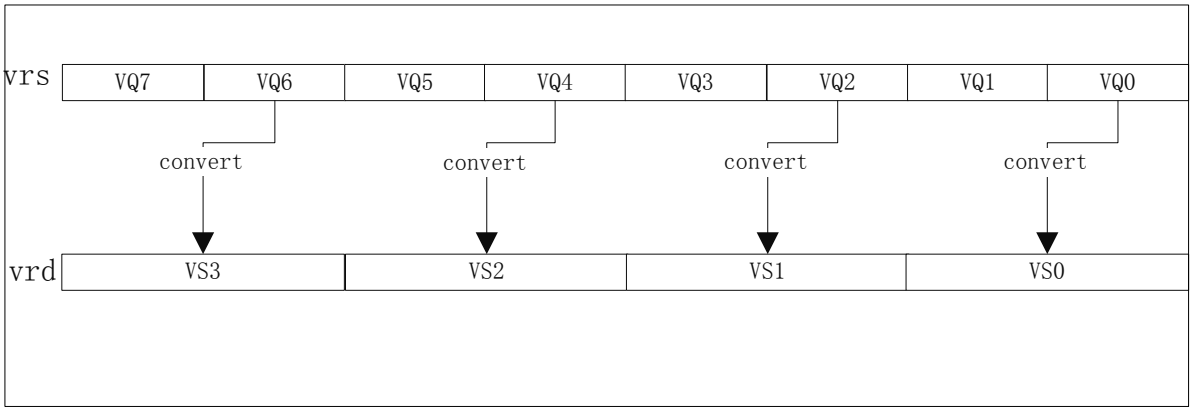
Syntax:

VCVTQESH vrd, vrs

Description：

The even 16-bit fixed-point Q15 elements in vrs are up-convert to single-precision floating-point format.

The fixed-point Q15 value is first convert to floating-point as a 16-bit integer (as though it was scaled up by 2^{15}) and then the resulting floating-point value is scaled down(divided by 2^{15})



Operation：

check_cop2_enable()

bit [127:0] op1=VPR[vrs]

for i in MLEN/32:

VRP[vrd][32,i]= q15_to_single(op1[16,2i])

Exceptions:

RI, CpU, MFPE

3.11.22 VCVTQEDW

Even Fixed-Point Q31 Convert to Double-Precision **2RFP**

Bit	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
	010010						11110					00001					vrs					vrd					11000					1

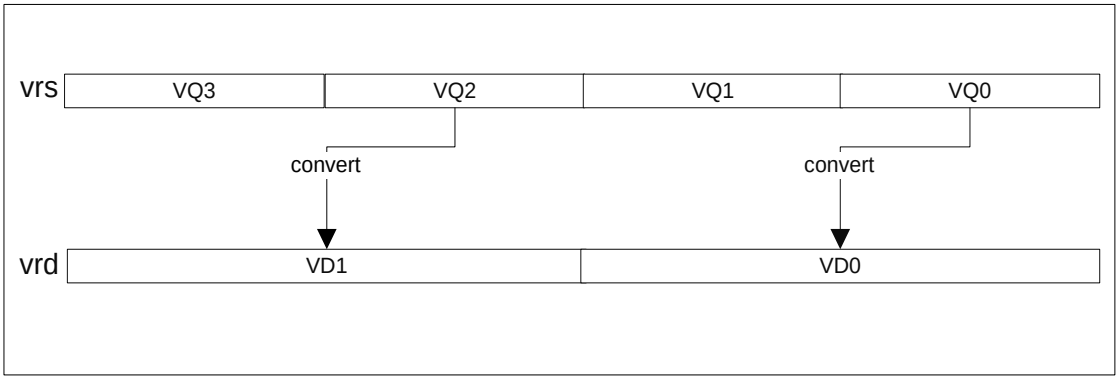
Syntax:

VCVTQEDW vrd, vrs

Description：

The even 32-bit fixed-point Q31 elements in vrs are up-convert to double-precision floating-point format.

The fixed-point Q31 value is first convert to floating-point as a 32-bit integer (as though it was scaled up by 2^{31}) and then the resulting floating-point value is scaled down(divided by 2^{31})



Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/64:
    VRP[vrd][64,i]= q31_to_double(op1[32,2i])
```

Exceptions:

RI, CpU, MFPE

3.11.23 VCVTQOSH

Odd Fixed-Point Q15 Convert to Single-Precision **2RFP**

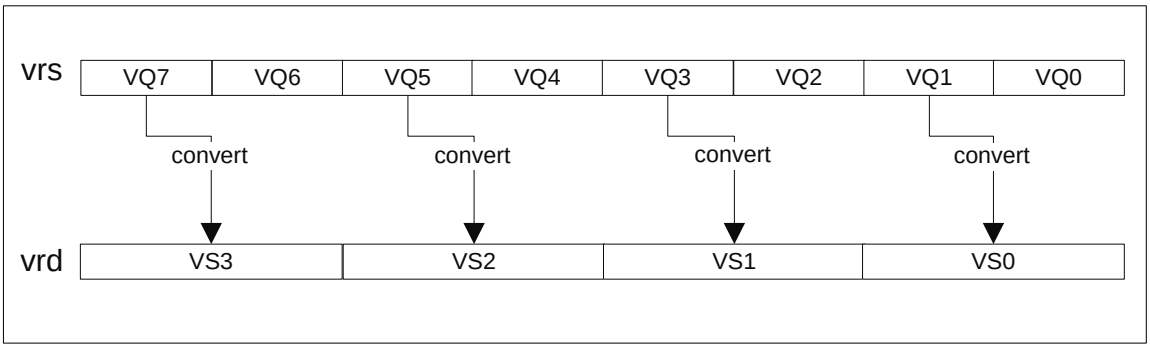
Bit	31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1	0
	010010	11110	00001	vrs	vrd	11100	0

Syntax:

VCVTQOSH vrd, vrs

Description:

The odd 16-bit fixed-point Q15 elements in vrs are up-convert to single-precision floating-point format. The fixed-point Q15 value is first convert to floating-point as a 16-bit integer (as though it was scaled up by 2^{15}) and then the resulting floating-point value is scaled down(divided by 2^{15})



Operation:

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/32:
    VRP[vrd][32,i]= q15_to_single(op1[16,2i+1])
```

Exceptions:

RI, CpU, MFPE

3.11.24 VCVTQODW

Odd Fixed-Point Q31 Convert to Double-Precision **2RFP**

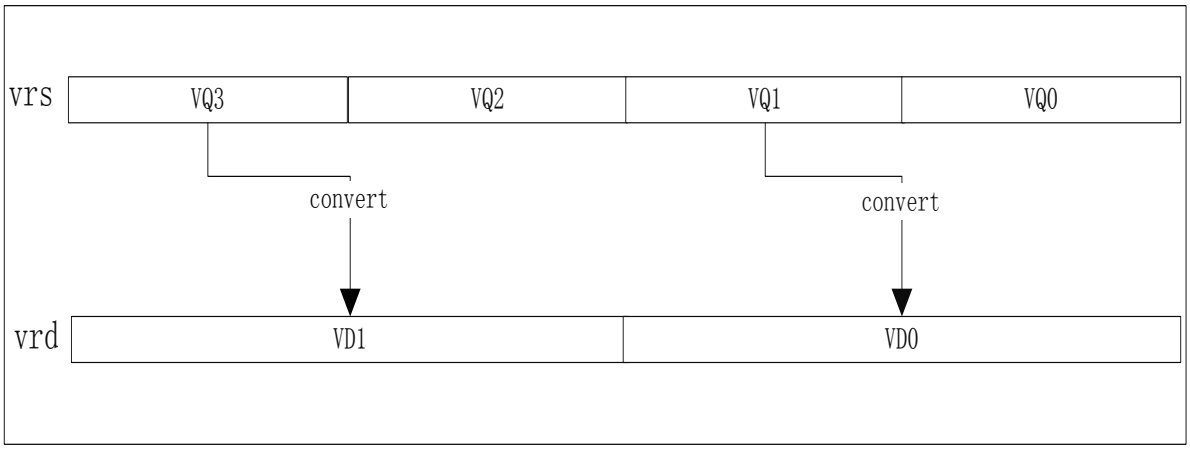
Bit	31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1	0
	010010	11110	00001	vrs	vrd	11100	1

Syntax:

VCVTQODW vrd, vrs

Description：

The odd 32-bit fixed-point Q31 elements in vrs are up-convert to double-precision floating-point format. The fixed-point Q31 value is first convert to floating-point as a 32-bit integer (as though it was scaled up by 2^{31}) and then the resulting floating-point value is scaled down(divided by 2^{31})



Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
for i in MLEN/64:
    VRP[vrd][32,i]= q31_to_double(op1[16,2i+1])
```

Exceptions:

RI, CpU, MFPE

3.11.25 VCVTQHS

Single-Precision Convert to Fixed-Point Q15 **3RFP**

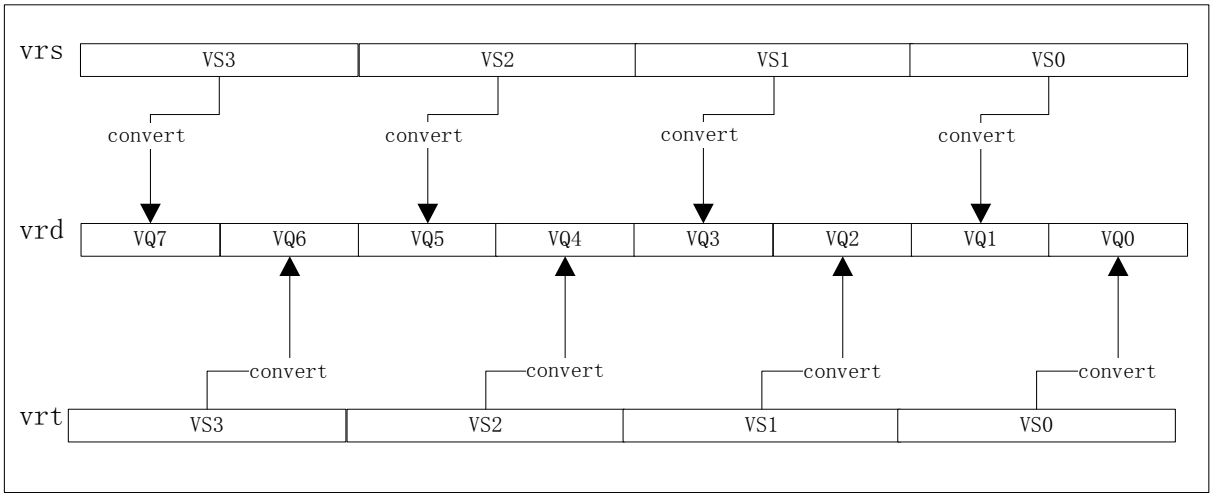
Bit	31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1	0
	010010	11000	vrt	vrs	vrd	00111	0

Syntax:

VCVTQHS vrd, vrs, vrt

Description:

The single-precision floating-point elements in vrs are down-convert to 16-bit Q15 fixed-point representation, and store in vrd even elements. The single-precision floating-point elements in vrt are down-convert to 16-bit Q15 fixed-point representation, and store in vrd odd elements.



Operation:

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/32:
    VRP[vrd][16,2i+1]= single_to_q15(op1[32,i])
    VRP[vrd][16,2i]= single_to_q15(op2[32,i])
```

Exceptions:

RI, CpU, MFPE

3.11.26 VCVTQWD

Double-Precision Convert to Fixed-Point Q31 **3RFP**

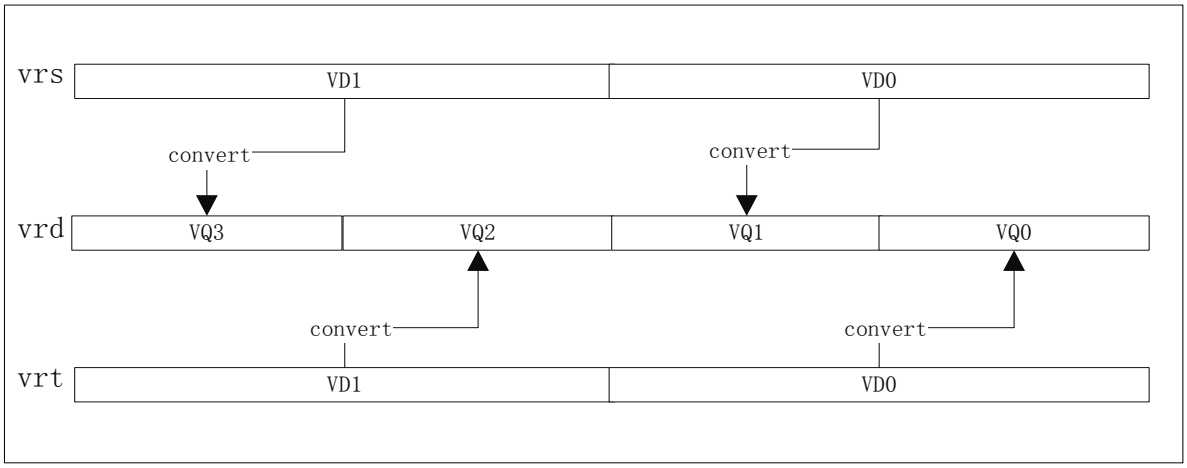
Bit	31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1	0
	010010	11000	vrt	vrs	vrd	00111	1

Syntax:

VCVTQWD vrd, vrs, vrt

Description：

The double-precision floating-point elements in vrs are down-convert to 32-bit Q31 fixed-point representation, and store in vrd even elements. The double-precision floating-point elements in vrt are down-convert to 32-bit Q31 fixed-point representation, and store in vrd odd elements.



Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/64:
    VRP[vrd][32,2i+1]= double_to_q31(op1[64,i])
    VRP[vrd][32,2i]= double_to_q31(op2[64,i])
```

Exceptions:

RI, CpU, MFPE

3.12 Fixed-Point Multiplication

3.12.1 MADDQ<fmt>

Fixed-Point Multiply and Add **3RFP**

Bit	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
	010010	11000	vrt	vrs	vrd	11000	fmt

Syntax:

MADDQH vrd, vrs, vrt

MADDQW vrd, vrs, vrt

Description：

The products of fixed-point elements in vrt by fixed-point elements in vrs are added to the fixed-point elements in vrd. The multiplication result is not saturated, i.e. exact(-1)*(-1)=1 is added to the destination. The saturated fixed-point results are stored back to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [127:0] op3=VRP[vrd]
for i in MLEN/esize
    VRP[vrd][esize,i]= qmadd(op1[esize,i] , op2[esize,i] , op3[esize,i])
```

Exceptions:

RI, CpU

3.12.2 MADDQR<fmt>

Fixed-Point Multiply and Add with Rounding **3RFP**

Bit	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
	010010	11000	vrt	vrs	vrd	11001	fmt

Syntax:

MADDQRH vrd, vrs, vrt

MADDQRW vrd, vrs, vrt

Description：

The products of fixed-point elements in vrt by fixed-point elements in vrs are added to the fixed-point elements in vrd. The multiplication result is not saturated, i.e. exact(-1)*(-1)=1 is added to the destination. The rounded and saturated fixed point results are stored back to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [127:0] op3=VRP[vrd]
for i in MLEN/esize
    VRP[vrd][esize,i]= qmaddr(op1[esize,i] , op2[esize,i] , op3[esize,i])
```

Exceptions:

RI, CpU

3.12.3 MSUBQ<fmt>

Fixed-Point Multiply and Subtract **3RFP**

Bit	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
	010010	11000	vrt	vrs	vrd	11010	fmt

Syntax:

MSUBQH vrd, vrs, vrt

MSUBQW vrd, vrs, vrt

Description：

The products of fixed-point elements in vrt by fixed-point elements in vrs are subtracted from the fixed-point elements in vrd. The multiplication result is not saturated, i.e. exact(-1)*(-1)=1 is subtracted from the destination. The saturated fixed-point results are stored back to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [127:0] op3=VRP[vrd]
for i in MLEN/esize
    VRP[vrd][esize,i]= qmsub(op1[esize,i] , op2[esize,i] , op3[esize,i])
```

Exceptions:

RI, CpU

3.12.4 MSUBQR<fmt>

Fixed-Point Multiply and Subtract with Rounding **3RFP**

Bit	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
	010010	11000	vrt	vrs	vrd	11011	fmt

Syntax:

MSUBQRH vrd, vrs, vrt

MSUBQRW vrd, vrs, vrt

Description：

The products of fixed-point elements in vrt by fixed-point elements in vrs are subtracted from the fixed-point elements in vrd. The multiplication result is not saturated, i.e. exact(-1)*(-1)=1 is subtracted from the destination. The saturated fixed-point results are stored back to vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [127:0] op3=VRP[vrd]
for i in MLEN/esize
    VRP[vrd][esize,i]= qmsubr(op1[esize,i] , op2[esize,i] , op3[esize,i])
```

Exceptions:

RI, CpU

3.12.5 MULQ<fmt>

Fixed-Point Multiply **3RFP**

Bit	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
	010010	11000	vrt	vrs	vrd	10100	fmt

Syntax:

MULQH vrd, vrs, vrt

MULQW vrd, vrs, vrt

Description：

The fixed-point elements in vrt are multiplied by fixed-point elements in vrs. The results are written to vrd.

Fixed-point multiplication for 16-bit 15Q and 32-bit 31Q is a regular signed multiplication followed by one bit shift left with saturation. Only the most significant of the result is preserved.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= qmul(op1[esize,i] , op2[esize,i],esize)
```

Exceptions:

RI, CpU

3.12.6 MULQR<fmt>

Fixed-Point Multiply with Rounding **3RFP**

Bit	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
	010010	11000	vrt	vrs	vrd	10101	fmt

Syntax:

MULQRH vrd, vrs, vrt

MULQRW vrd, vrs, vrt

Description：

The fixed-point elements in vrt are multiplied by fixed-point elements in vrs. The rounded results are written to vrd.

Fixed-point multiplication for 16-bit 15Q and 32-bit 31Q is a regular signed multiplication followed by one bit shift left with saturation. Only the most significant of the result is preserved.

The rounding is done by adding 1 to the most significant bit that is going to be discarded prior to shifting left the full multiplication result.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
for i in MLEN/esize
    VRP[vrd][esize,i]= qmulr(op1[esize,i] , op2[esize,i])
```

Exceptions:

RI, CpU

3.13 Shift

3.13.1 SLL<fmt>

Shift Left **3RINT**

Bit	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
	010010						10001					vrt					vrs					vrd					1010				fmt	

Syntax:

SLLB vrd, vrs, vrt

SLLH vrd, vrs, vrt

SLLW vrd, vrs, vrt

SLLD vrd, vrs, vrt

Description：

Take each element in vrs vector register, left shift them by the number of corresponding element in vrt vector register, the empty lower-order bits are set to all 0, and place the result in the destination vector register vrd.

Bits shifted out of the left of each element are lost.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [log2(esize)-1:0] shift_mount
bit[esize-1:0] temp
for i in MLEN/esize
    shift_mount = op2[esize,i] [log2(esize)-1:0]
    temp        =op1[esize,i]
    VRP[vrd][esize,i]={temp[esize-1-shift_mount:0] ,shift_mount{0}}
```

Exceptions:

RI, CpU

3.13.2 SLLI<fmt>

Immediate Shift Left **2R6I**

Bit	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
	011100	fmt	10	imm	vrs	vrd	111000

Syntax:

SLLIB vrd, vrs, imm

SLLIH vrd, vrs, imm

SLLIW vrd, vrs, imm

SLLID vrd, vrs, imm

Description：

Take each element in vrs vector register, left shift them by an immediate value imm, the empty lower-order bits are set to all 0, and place the result in the destination vector register vrd.

Bits shifted out of the left of each element are lost.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [5:0] imm=imm[5:0]
bit [log2(esize)-1:0] shift_mount=imm[log2(esize)-1:0]
bit[esize-1:0] temp
for i in MLEN/esize
    temp        =op1[esize,i];
    VRP[vrd][esize,i]={temp[esize-1-shift_mount:0] ,shift_mount{0}}
```

Exceptions:

RI, CpU

3.13.3 SRA<fmt>

Arithmetic Shift Right **3RINT**

Bit	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
	010010	10000	vrt	vrs	vrd	0110	fmt

Syntax:

SRAB vrd, vrs, vrt

SRAH vrd, vrs, vrt

SRAW vrd, vrs, vrt

SRAD vrd, vrs, vrt

Description：

Take each element in vrs vector register, right shift them by the number of corresponding element in vrt vector register, the empty high-order bits are filled with the initial value of the sign bit of the data element(arithmetic shift right), and places the result in the destination vector register vrd.

Bits shifted out of the right of each element are lost.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [log2(esize)-1:0] shift_mount
bit[esize-1:0] temp
bit sign
for i in MLEN/esize
    shift_mount = op2[esize,i] [log2(esize)-1:0]
    temp        = op1[esize,i]
    sign        = op1[esize-1]
    VRP[vrd][esize,i]={shift_mount{sign}, temp[esize-1:shift_mount]}
```

Exceptions:

RI, CpU

3.13.4 SRAI<fmt>

Arithmetic Shift Right Immediate **2R6I**

Bit	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
	011100	fmt	00	imm	vrs	vrd	111001

Syntax:

SRAIB vrd, vrs, imm

SRAIH vrd, vrs, imm

SRAIW vrd, vrs, imm

SRAID vrd, vrs, imm

Description：

Take each element in vrs vector register, right shift them by an immediate value imm, ,the empty high-order bits are filled with the initial value of the sign bit of the data element(arithmetic shift right), and places the result in the destination vector register vrd.

Bits shifted out of the right of each element are lost.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [5:0] imm=imm[5:0]
bit [log2(esize)-1:0] shift_mount=imm[log2(esize)-1:0]
bit[esize-1:0] temp
bit sign
for i in MLEN/esize
    temp        = op1[esize,i]
    sign        = op1[esize-1]
    VRP[vrd][esize,i]={ shift_mount{sign} , temp[esize-1:shift_mount]}
```

Exceptions:

RI, CpU

3.13.5 SRAR<fmt>

Arithmetic Rounding Shift Right **3RINT**

Bit	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
	010010	10000	vrt	vrs	vrd	1000	fmt

Syntax:

SRARB vrd, vrs, vrt

SRARH vrd, vrs, vrt

SRARW vrd, vrs, vrt

SRARD vrd, vrs, vrt

Description：

Take each element in vrs vector register, right shift them by the number of corresponding element in vrt vector register, the empty high-order bits are filled with the initial value of the sign bit of the data element(arithmetic shift right), then add shift value to the most significant discarded bit(for rounding), and places the result in the destination vector register vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [log2(esize)-1:0] shift_mount
bit[esize-1:0] temp
bit sign
bit round
for i in MLEN/esize
    shift_mount = op2[esize,i] [log2(esize)-1:0]
    temp        = op1 [esize,i]
    sign        = op1 [esize-1]
    round       =op1 [shift_mount-1]
    VRP[vrd][esize,i]={ shift_mount{sign} , temp[esize-1:shift_mount]}+round
```

Exceptions:

RI, CpU

3.13.6 SRARI<fmt>

Arithmetic Immediate Shift Right **2R6I**

Bit	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
	011100	fmt	01	imm	vrs	vrd	111001

Syntax:

SRARIB vrd, vrs, imm

SRARIH vrd, vrs, imm

SRARIW vrd, vrs, imm

SRARID vrd, vrs, imm

Description：

Take each element in vrs vector register, right shift them by an immediate value imm, the empty high-order bits are filled with the initial value of the sign bit of the data element(arithmetic shift right), then add shift value to the most significant discarded bit(for rounding), and places the result in the destination vector register vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [5:0] imm=imm[5:0]
bit [log2(esize)-1:0] shift_mount=imm[log2(esize)-1:0]
bit[esize-1:0] temp
bit sign
bit round
for i in MLEN/esize
    temp        = op1 [esize,i]
    sign        = op1 [esize-1]
    round       =op1 [shift_mount-1]
    VRP[vrd][esize,i]={ shift_mount{sign} , temp[esize-1:shift_mount]}+round
```

Exceptions:

RI, CpU

3.13.7 SRL<fmt>

Logical Shift Right **3RINT**

Bit	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
	010010	10000	vrt	vrs	vrd	0111	fmt

Syntax:

SRLB vrd, vrs, vrt

SRLH vrd, vrs, vrt

SRLW vrd, vrs, vrt

SRLD vrd, vrs, vrt

Description：

Take each element in vrs vector register, right shift them by the number of corresponding element in vrt vector register, the empty higher-order bits are set to all 0(logical shift right), and places the result in the destination vector register vrd.

Bits shifted out of the right of each element are lost.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [log2(esize)-1:0] shift_mount
bit[esize-1:0] temp
for i in MLEN/esize
    shift_mount = op2[esize,i] [log2(esize)-1:0]
    VRP[vrd][esize,i]={ shift_mount{0} , temp[esize-1:shift_mount]}
```

Exceptions:

RI, CpU

3.13.8 SRLI<fmt>

Logical Shift Right Immediate **2R6I**

Bit	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
	011100	fmt	10	imm	vrs	vrd	111001

Syntax:

SRLIB vrd, vrs, imm

SRLIH vrd, vrs, imm

SRLIW vrd, vrs, imm

SRLID vrd, vrs, imm

Description：

Take each element in vrs vector register, right shift them by an immediate number imm, the empty higher-order bits are set to all 0(logical shift right), and places the result in the destination vector register vrd.

Bits shifted out of the right of each element are lost.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [5:0] imm=imm[5:0]
bit [log2(esize)-1:0] shift_mount=imm[log2(esize)-1:0]
bit[esize-1:0] temp
for i in MLEN/esize
    VRP[vrd][esize,i]={ shift_mount{0} , temp[esize-1:shift_mount]}
```

Exceptions:

RI, CpU

3.13.9 SRLR<fmt>

Logical Rounding Shift Right **3RINT**

Bit	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
	010010	10000	vrt	vrs	vrd	1001	fmt

Syntax:

SRLRB vrd, vrs, vrt

SRLRH vrd, vrs, vrt

SRLRW vrd, vrs, vrt

SRLRD vrd, vrs, vrt

Description：

Take each element in vrs vector register, right shift them by the number of corresponding element in vrt vector register, the empty higher-order bits are set to all 0(logical shift right), then add shift value to the most significant discarded bit(for rounding), and places the result in the destination vector register vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [log2(esize)-1:0] shift_mount
bit[esize-1:0] temp
bit round
for i in MLEN/esize
    shift_mount = op2[esize,i] [log2(esize)-1:0]
    round       =op1 [shift_mount-1]
    VRP[vrd][esize,i]= { shift_mount{0} , temp[esize-1:shift_mount]}+round
```

Exceptions:

RI, CpU

3.13.10 SRLRI<fmt>

Logical Rounding Shift Right Immediate **2R6I**

Bit	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
	011100	fmt	11	imm	vrs	vrd	111001

Syntax:

SRLRIB vrd, vrs, imm

SRLRIH vrd, vrs, imm

SRLRIW vrd, vrs, imm

SRLRID vrd, vrs, imm

Description：

Take each element in vrs vector register, right shift them by an immediate number imm, the empty higher-order bits are set to all 0(logical shift right), then add shift value to the most significant discarded bit(for rounding), and places the result in the destination vector register vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit [5:0] imm=imm[5:0]
bit [log2(esize)-1:0] shift_mount=imm[log2(esize)-1:0]
bit[esize-1:0] temp
bit round
for i in MLEN/esize
    round          =op1 [shift_mount-1]
    VRP[vrd][esize,i]= { shift_mount{0} , temp[esize-1:shift_mount]}+round
```

Exceptions:

RI, CpU

3.14 Element Permute

3.14.1 SHUFV

Shuffle **4R**

Bit	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
	011100	vrr	vrt	vrs	vrd	011000

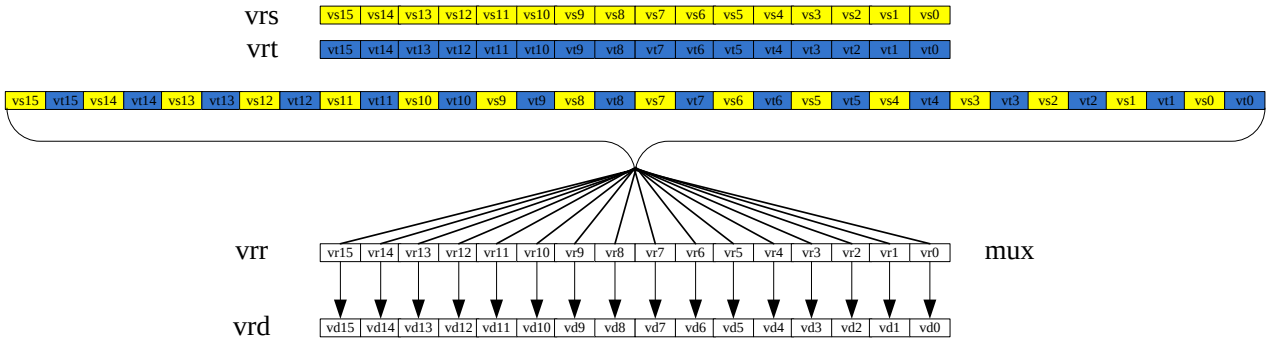
Syntax:

SHUFV vrd, vrs, vrt, vrr

Description：

The vector shuffle instruction selectively copy elements from the vrs and vrt into vrd based on the corresponding control element in vrr.

The least significant 6 bits in vrr control elements modulo the number of elements in the concatenated vrs, vrt specify the index of the source. If bit 7 is 1, there will be no copy, but rather the destination elements is set to 0.



Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [127:0] op2=VPR[vrt]
bit[255:0] op
for i in MLEN/8:
    op[16,i] = { op1[8,i] , op2[8,i] }
for i in MLEN/8
    bit [log2(esize)-1:0][esize-1:0] sel=VPR[VRR][esize,i]
    VRP[vrd][esize,i]= (sel[esize]==1'b1)?0:op[esize,sel]
```

Exceptions:

RI, CpU

3.14.2 INSFCPU<fmt>

Insert GPR to Vector Element **2R8I**

Bit	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
	011100	fmt	imm	rs	vrd	110001

Syntax:

INSFCPUB vrd[imm], rs

INSFCPUH vrd[imm], rs

INSFCPUW vrd[imm], rs

Description：

Select imm element in vrd vector register, and place element value from GPR register rs .All other elements in vrd are unchanged .If the source GPR is wider than the destination data format, the destination's elements will be set to the least significant bits of the GPR.

Operation：

check_cop2_enable()

bit [31:0] op1=GPR[rs]

bit [4:0] imm=imm[4:0]

bit [log2(MLEN/esize)-1:0] sel=imm[log2(MLEN/esize)-1:0]

VRP[vrd][esize,sel]= op1[esize-1:0]

Exceptions:

RI, CpU

3.14.3 INSFFPU<fmt>

Insert FPR to Vector Element **2R5I**

Bit	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
	010010	11111	imm	fs	vrd	00000	fmt

Syntax:

INSFFPUW vrd[imm], fs

INSFFPUD vrd[imm], fs

Description：

Select imm elements in vrd vector register,and replace element value by FPR register fs. All other elements in vrd are unchanged .If the source FPR is wider than the destination data format, the destination's elements will be set to the least significant bits of the FPR.

Operation：

check_cop2_enable()

bit [127:0] op1=FPR[fs]

bit [4:0] imm=imm[4:0]

bit [log2(MLEN/esize)-1:0] sel=imm[log2(MLEN/esize)-1:0]

VRP[vrd][esize,sel]= op1[esize-1:0]

Exceptions:

RI, CpU

3.14.4 INSFMXU<fmt>

Insert MXU2 element 0 to Vector Element **2R8I**

Bit	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
	011100	fmt	imm	vrs	vrd	110010

Syntax:

INSFMXUB vrd[imm],vrs[0]

INSFMXUH vrd[imm],vrs[0]

INSFMXUW vrd[imm],vrs[0]

INSFMXUD vrd[imm],vrs[0]

Description：

Set elements imm in vrd to elements 0 in vrs value. All other elements in vrd are unchanged.

Operation：

check_cop2_enable()

bit [127:0] op1=VPR[vrs]

bit [4:0] imm=imm[4:0]

bit [log2(MLEN/esize)-1:0] sel=imm[log2(MLEN/esize)-1:0]

VRP[vrd][esize,sel]= op1[esize,0]

Exceptions:

RI, CpU

3.14.5 REPX<fmt>

Replicate Vector Element **3RINT**

Bit	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
	010010	10010	rt	vrs	vrd	0111	fmt

Syntax:

REPXB vrd, vrs[rt]

REPXH vrd, vrs[rt]

REPXW vrd, vrs[rt]

REPXD vrd, vrs[rt]

Description：

Replicate vrs elements with index given by GPR rt to all elements in vrd.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [log2(MLEN/esize)-1:0] sel=GPR[rt][log2(MLEN/esize)-1:0]
for i in MLEN/esize
    VRP[vrd][esize,i]= op1[esize,sel]
```

Exceptions:

RI, CpU

3.14.6 REPI<fmt>

Immediate Replicate Vector Element **2R8I**

Bit	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
	011100	fmt	imm	vrs	vrd	110101

Syntax:

REPIB vrd, vrs[imm]

REPIH vrd, vrs[imm]

REPIW vrd, vrs[imm]

REPID vrd, vrs[imm]

Description：

Immediate selected elements of in vrs vector register and replicated in all destination elements in vrd vector register.

Operation：

```
check_cop2_enable()
bit [127:0] op1=VPR[vrs]
bit [4:0] imm=imm[4:0]
bit [log2(MLEN/esize)-1:0] sel=imm[log2(MLEN/esize)-1:0]
for i in MLEN/esize
    VRP[vrd][esize,i]= op1[esize,sel]
```

Exceptions:

RI, CpU

3.15 Load/Store

3.15.1 MTCPUS<fmt>

Move Signed Elements Value in MXU2 to GPR **2R8I**

Bit	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
	011100						fmt		imm								rd					vrs					110011					

Syntax:

MTCPUSB rd, vrs[imm]

MTCPUSH rd, vrs[imm]

MTCPUSW rd, vrs[imm]

Description：

Element value sign extended and copied to GPR.

Operation：

check_cop2_enable()

bit [127:0] op1=VPR[vrs]

bit [4:0] imm=imm[4:0]

bit [log2(MLEN/esize)-1:0] sel=imm[log2(MLEN/esize)-1:0]

GPR[rd]= signed(op1[esize,sel])

Exceptions:

RI, CpU

3.15.2 MTCPUU<fmt>

Move Unsigned Elements Value in MXU2 to GPR **2R8I**

Bit	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
	011100	fmt	imm	rd	vrs	110100

Syntax:

MTCPUUB rd, vrs[imm]

MTCPUUH rd, vrs[imm]

MTCPUUW rd, vrs[imm]

Description：

Zero-extend element imm of vrs and copy the result to GPR rd.

Operation：

check_cop2_enable()

bit [127:0] op1=VPR[vrs]

bit [4:0] imm=imm[4:0]

bit [log2(MLEN/esize)-1:0] sel=imm[log2(MLEN/esize)-1:0]

GPR[rd]= unsigned(op1[esize,sel])

Exceptions:

RI, CpU

3.15.3 MFCPU<fmt>

MXU2 Elements Filled by GPR **2RINT**

Bit	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
	010010	11110	00000	rs	vrd	1111	fmt

Syntax:

MFCPUB vrd, rs

MFCPUH vrd, rs

MFCPUW vrd, rs

Description：

Replicated GPR rs value into all elements in vrd. If the source GPR is wider than the destination data format, the destination's elements will be set to the least significant bits of the GPR.

Operation：

```
check_cop2_enable()
bit [31:0] op1=GPR[rs]
for i in MLEN/esize
    VRP[vrd][esize,i]= op1[esize,0]
```

Exceptions:

RI, CpU

3.15.4 MTFPU<fmt>

Move Elements Value in MXU2 to FPR **2R5I**

Bit	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
	010010	11111	imm	vrs	fd	00010	fmt

Syntax:

MTFPUW fd, vrs[imm]

MTFPUD fd, vrs[imm]

Description：

Element imm of vrs and copy the result to FPR fd.

Operation：

check_cop2_enable()

bit [127:0] op1=VPR[vrs]

bit [4:0] imm=imm[4:0]

bit [log2(MLEN/esize)-1:0] sel=imm[log2(MLEN/esize)-1:0]

FPR[fd]= op1[esize,sel]

Exceptions:

RI, CpU

3.15.5 MFFPU<fmt>

MXU2 Elements Filled by FPR **2RFP**

Bit	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
	010010	11110	00001	fs	vrd	11111	fmt

Syntax:

MFFPUW vrd, fs

MFFPUD vrd, fs

Description：

Replicated FPR fs value into all elements in vrd. If the source FPR is wider than the destination data format, the destination's elements will be set to the least significant bits of the FPR.

Operation：

```
check_cop2_enable()
bit [esize-1:0] op1=FPR[fs]
for i in MLEN/esize
    VRP[vrd][esize,i]= op1 [esize,0]
```

Exceptions:

RI, CpU

3.15.6 CTCMXU

GPR copy to MXU2 Control Register **2RFP**

Bit	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
	010010	11110	00001	rs	mcsrd	11110	0

Syntax:

CTCMXU mcd, rs

Description：

The content of GPR rs is copied to MXU2 control register mcd.

Operation：

check_cop2_enable()

bit [31:0] op1=GPR[rs]

MCSR[mcd]= op1

Exceptions:

RI, CpU, MFPE

3.15.7 CFCMXU

GPR Copy from MXU2 Control Regisrer **2RFP**

Bit	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
	010010						11110					00001				rd					mcsrs					11110						1	

Syntax:

CFCMXU rd, mcs

Description：

The content of MXU2 control register mcs is copied to GPR rd.

Operation：

check_cop2_enable()

bit [31:0] op1=MCSR[mcs]

GPR[rd]= op1

Exceptions:

RI, CpU

3.15.8 LU1Q

Load Unaligned 1 Quad-word **2R10I**

Bit	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
	011100						base					offset										vrd					010100					

Syntax:

LU1Q vrd, offset(base)

Description：

The MLEN/8 bytes at the effective memory location addressed by the base and the 10-bit signed immediate offset are fetched in vrd.

The effective memory location address has no alignment restrictions.

Operation：

check_cop2_enable()

bit [31:0] op1=GPR[base]

bit [31:0] addr=op1+{22{offset[9]},offset[9:0]}

VPR[vrd]=mem[MLEN:addr]

Exceptions:

RI, CpU

3.15.9 LU1QX

Load Unaligned 1 Quad-word Indexed **3R**

Bit	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
	011100	base	index	00000	vrd	000111

Syntax:

LU1QX vrd, index(base)

Description：

The MLEN/8 bytes at the effective memory location addressed by the base and index offset given by GPR are fetched in vrd.

The effective memory location address has no alignment restrictions.

Operation：

check_cop2_enable()

bit [31:0] op1=GPR[base]

bit [31:0] op2=GPR[index]

bit [31:0] addr=op1+op2

VPR[vrd]=mem[MLEN:addr]

Exceptions:

RI, CpU

3.15.10 LA1Q

Load Aligned 1 Quad-word **2R10I**

Bit	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
	011100	base	offset	vrd	101100	

Syntax:

LA1Q vrd, offset(base)

Description：

The MLEN/8 bytes at the memory location specified by the aligned effective address are fetched and placed in *vrd*.

10-bit signed *offset* multiplied the number of bytes (MLEN/8) in vrd, then add the contents of GPR *base* to form the effective address.

An Address Error exception occurs if $EffectiveAddress_{3...0} \neq 0$ (not quad-word aligned).

In the assembly language syntax, *offset* is in bytes and has to be multiple of MELN/8. The assembler determines the *offset* bit field value of instruction encode dividing the byte offset by MLEN/8.

Operation：

```
check_cop2_enable()
bit [31:0] op1=GPR[base]
bit [31:0] addr=op1+sign_extend(offset) * MLEN/8
if addr[3:0] != 0:
    signalexception(AdEL)
VPR[vrd]=mem[MLEN:addr]
```

Exceptions:

RI, CpU, AdEL

3.15.11 LA1QX

Load Aligned 1 Quad-word Indexed **3R**

Bit	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
	011100	base	index	10000	vrd	000111

Syntax:

LA1QX vrd, index(base)

Description：

The MLEN/8 bytes at the memory location specified by the aligned effective address are fetched and placed in *vrd.*

The content of GPR *index* and GPR *base* are added to form the effective address.

An Address Error exception occurs if $EffectiveAddress_{3...0} \neq 0$ (not quad-word aligned).

Operation：

```
check_cop2_enable()
bit [31:0] op1=GPR[base]
bit [31:0] op2=GPR[index]
bit [31:0] addr=op1*+op2
if addr[3:0] != 0:
    signalexception(AdEL)
VPR[vrd]=mem[MLEN:addr]
```

Exceptions:

RI, CpU, AdEL

3.15.12 SU1Q

Store Unaligned 1 Quad-word **2R10I**

Bit	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
	011100	base	offset	vrd	011100	

Syntax:

SU1Q vrd, offset(base)

Description：

The 16 bytes in vrd are stored at the effective memory location addressed by the base and the 10-bit signed immediate offset.

The effective memory location address has no alignment restrictions.

Operation：

```
check_cop2_enable()
bit [31:0] op1=GPR[base]
bit [31:0] addr=op1+{22{offset[9]},offset[9:0]}
mem[MLEN:addr]= VPR[vrd]
```

Exceptions:

RI, CpU

3.15.13 SU1QX

Store Unaligned 1 Quad-word Indexed **3R**

Bit	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
	011100	base	index	00100	vrd	000111

Syntax:

SU1QX vrd, index(base)

Description：

The MLEN/8 bytes in vrd are stored at the effective memory location addressed by the base and index offset given by GPR .

The effective memory location address has no alignment restrictions.

Operation：

check_cop2_enable()

bit [31:0] op1=GPR[base]

bit [31:0] op2=GPR[index]

bit [31:0] addr=op1+op2

mem[MLEN:addr] =VPR[vrd]

Exceptions:

RI, CpU

3.15.14 SA1Q

Store Aligned 1 Quad-word **2R10I**

Bit	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
	011100	base	offset	vrd	111100	

Syntax:

SA1Q vrd, offset(base)

Description：

The MLEN/8 bytes in vrd are stored in memory at the location specified by the aligned effective address.

10-bit signed *offset* multiplied the number of bytes (MLEN/8) in vrd, then add the contents of GPR *base* to form the effective address.

An Address Error exception occurs if $EffectiveAddress_{3...0} \neq 0$ (not quad-word aligned).

In the assembly language syntax, *offset* is in bytes and has to be multiple of MELN/8. The assembler determines the *offset* bit field value of instruction encode dividing the byte offset by MLEN/8.

Operation：

```
check_cop2_enable()
bit [31:0] op1=GPR[base]
bit [31:0] addr=op1+sign_extend(offset) * MLEN/8
if addr[3:0] != 0:
    signalexception(AdES)
mem[MLEN:addr]= VPR[vrd]
```

Exceptions:

RI, CpU, AdES

3.15.15 SA1QX

Store Aligned 1 Quad-word Indexed **3R**

Bit	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
	011100	base	index	10100	vrd	000111

Syntax:

SA1QX vrd, index(base)

Description：

The MLEN/8 bytes from *vrd* are stored in memory at the location specified by the aligned effective address.

The content of GPR *index* and GPR *base* are added to form the effective address.

An Address Error exception occurs if $EffectiveAddress_{3...0} \neq 0$ (not quad-word aligned).

Operation：

```
check_cop2_enable()
bit [31:0] op1=GPR[base]
bit [31:0] op2=GPR[index]
bit [31:0] addr=op1+op2
if addr[3:0] != 0:
    signalexception(AdES)
mem[MLEN:addr] =VPR[vrd]
```

Exceptions:

RI, CpU, AdES

3.15.16 LI<fmt>

Immediate Load **1R15I**

Bit	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
	011100	imm	vrd	0011	fmt

Syntax:

LIB vrd, imm

LIH vrd, imm

LIW vrd, imm

LID vrd, imm

Description：

The signed immediate imm is replicated in all vrd elements. For byte elements, only the least significant 8 bits of imm will be used.

Operation：

```
check_cop2_enable()
bit [14:0] imm=imm[14:0]
bit [esize-1:0] data
for i in MLEN/esize
    if (esize==8)
        data=imm[7:0]
    else
        data={(esize-15){imm[14]},imm[14:0]}
    VRP[vrd][esize,i]= data
```

Exceptions:

RI, CpU

Appendix A

A.1 Instruction Formats

- 1R10I: instruction with an 10-bit immediate value and data format fmt code in bits25..24

1 Register,10-bit Immediate **1R10I**

Bit	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
	opcode						fmt		funt1			vrs					offset											funct0					

- 1R15I: instruction with an 15-bit immediate value and 1 register ,data format fmt code in bit 1..0

1 Register,15-bit Immediate **1R15I**

Bit	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
	opcode																imm										funct0				fmt	

- 4R: instruction with 4 registers.

4 Registers **4R**

Bit	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
	opcode						vrr					vrt					vrs						vrd				funct0					

- 3R: instruction with 3 registers.

3 Registers **3R**

Bit	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
	opcode						base					index					funct1						vrd				funct0					

- 2R10I: instruction with 10 immediate and 2 registers.

2 Registers,10-bit Immediate **2R10I**

Bit	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
	opcode						base					offset											vrd				funct0					

- 3RVEC: instruction with 3 registers

3 Registers, vector **3RVEC**

Bit	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
	opcode						funct1					vrt					vrs						vrd				funct0					

- 2R8I: instruction with 8-bit immediate and 2 registers. data format fmt code in bits 25..24

2 Register, 8-bit Immediate **2R8I**

Bit	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
	opcode						fmt		imm								vrs						vrd				funct0					

- 2R6I: instruction with 6-bit immediate and 2 registers. data format fmt code in bits 25..24

2 Register, 6-bit Immediate **2R6I**

Bit	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

opcode	fmt	funct1	imm	vrs	vrd	funct0

- 2R5I: instruction with 5-bit immediate and 2 registers. data format fmt code in bit 0

2 Register, 5-bit Immediate **2R5I**

Bit	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
opcode														funct1						imm								fs/vrs				vrd/fd						funct0								fmt

- 3RINT: instruction 3 registers. data format fmt code in bits 1..0

3 Register, integer **3RINT**

Bit	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
opcode												funct1					vrt								vrs								vrd								funct0								fmt

- 2RINT:instruction 2 registers. data format fmt code in bits 1..0

2 Registers, integer **2RINT**

Bit	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
opcode												funct2					funct1								vrs								vrd								funct0								fmt

- 3RFP:instruction 3 registers. data format fmt code in bits 0

3 Registers, floating point **3RFP**

Bit	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
opcode												funct1					vrt								vrs								vrd								funct0								fmt

- 2RFP:instruction 2 registers. data format fmt code in bits

2 Registers, floating point **2RFP**

Bit	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
opcode												funct2					funct1								vrs								vrd								funct0								fmt

A.2 Instruction Bit Encoding

This chapter describe the bit encoding tables used for MXU2. These tables only list the instruction encoding for the MXU2 instruction.

A instruction's encoding is found at the below tables.

- MXU2 Encoding of the Opcode Field

opcode	***bits[28:26]***							
bits[31:29]	000	001	010	011	100	101	110	111
000								
001								
010			COP2					
011					SPECILA2			
100								
101								
110								
111								

- MXU2 SPECIAL2 Encoding of funct0 Field

funct0	***bits[2:0]***							
bits[5:3]	000	001	010	011	100	101	110	111
000	MADD	MADDU	MUL	θ	MSUB	MSUBU	θ	3R
001	ε	θ	θ	θ	LIB	LIH	LIW	LID
010	θ	θ	θ	θ	LU1Q	θ	θ	θ
011	SHUFV	BSELV	θ	θ	SU1Q	θ	θ	θ
100	CLZ	CLO	θ	θ	β	β	θ	θ
101	1R10I-0	1R10I-1	θ	θ	LA1Q	θ	θ	θ
110	2R8I-0	2R8I-1	2R8I-2	2R8I-3	2R8I-4	2R8I-5	θ	θ
111	2R6I-0	2R6I-1	θ	θ	SA1Q	θ	θ	SDBBP Pσ

- Encoding of funct1 Field for MXU2 3R Instruction Formats

funct1	***bits[13:11]***							
bits[15:14]	000	001	010	011	100	101	110	111
00	LU1QX	θ	θ	θ	SA1QX	θ	θ	θ
01	θ	θ	θ	θ	θ	θ	θ	θ
10	LA1QX	θ	θ	θ	SX1QX	θ	θ	θ
11	θ	θ	θ	θ	θ	θ	θ	θ

- Encoding of fmt and fuunct1 Field for MXU2 1R10I-0 Instruction Formats

fmt,funct1	***bits[23:21]***							
bits[25:24]	000	001	010	011	100	101	110	111
00	BEQZ16B				BNEZ8H			

01	BEZQ8H				BNEZ4W			
10	BEQZ4W				BNEZ2D			
11	BEQZ2D				BNEZ2D			

- Encoding of fmt and funct1 Field for MXU2 1R10I-1 Instruction Formats

fmt,funct1	***bits[23:21]***							
bits[25:24]	000	001	010	011	100	101	110	111
00	BEQZ1Q				BNEZ1Q			
01								
10								
11								

- Encoding of fmt and funct0 Field for MXU2 2R8I Instruction Formats

fmt,funct0	***bits[5:0]***					
bits[25:24]	2R8I-0	2R8I-1	2R8I-2	2R8I-3	2R8I-4	2R8I-5
00	ANDIB	INSFCPUB	INSFMXUB	MTCPUSB	MTCPUUB	REPIB
01	NORIB	INSFCPUH	INSFMXUH	MTCPUSH	MTCPUUH	REPIH
10	ORIB	INSFCPUW	INSFMXUW	MTCPUSW	MTCPUUW	REPIW
11	XORIB	INSFCPUB	INSFMXUD			REPID

- Encoding of fmt and funct1 Field for MXU2 2R6I-0 Instruction Formats

fmt,funct1	***bits[23:22]***			
bits[25:24]	00	01	10	11
00	SATSB	SATUB	SLLIB	
01	SATSH	SATUH	SLLIH	
10	SATSW	SATUW	SLLIW	
11	SATSD	SATUD	SLLID	

- Encoding of fmt and funct1 Field for MXU2 2R6I-1 Instruction Formats

fmt,funct1	***bits[23:22]***			
bits[25:24]	00	01	10	11
00	SRAIB	SRARIB	SRLIB	SRLRIB
01	SRAIH	SRARIH	SRLIH	SRLRIH
10	SRAIW	SRARIW	SRLIW	SRLRIW
11	SRAID	SRARID	SRLID	SRLRID

- MXU2 COP2 Encoding of funct1 Field

funct1	***bits[23:21]***							
bits[25:24]	000	001	010	011	100	101	110	111
00	MFC2	β	CFC2	MHFC2	MTC2	β	CTC2	MTHC2
01	BC2	*	*	*	*	*	*	*

10	3RINT-0	3RINT-1	3RINT-2				3RVEC	
11	3RFP						2R	2R5I

- Encoding of funct0 and fmt Field for MXU2 3RINT-0 Instruction Formats

fmt,funct0	***bits[2:0]***							
bits[5:3]	000	001	010	011	100	101	110	111
000	MAXAB	MAXAH	MAXAW	MAXAD	MINAB	MINAH	MINAW	MINAD
001	MAXSB	MAXSH	MAXSW	MAXSD	MINSB	MINSH	MINSW	MINSD
010	MAXUB	MAXUH	MAXUW	MAXUD	MINUB	MINUH	MINUW	MINUD
011	SRAB	SRAH	SRAW	SRAD	SRLB	SRLH	SRLW	SRLD
100	SRARB	SRARH	SRARW	SRARD	SRLRB	SRLRH	SRLRW	SRLRD
101	CEQB	CEQH	CEQW	CEQD	CNEB	CNEH	CNEW	CNED
110	CLTSB	CLTSH	CLTSW	CLTSD	CLTUB	CLTUH	CLTUW	CLTUD
111	CLESB	CLESH	CLESW	CLESD	CLEUB	CLEUH	CLEUW	CLEUD

- Encoding of funct0 and fmt Field for MXU2 3RINT-1 Instruction Formats

fmt,funct0	***bits[2:0]***							
bits[5:3]	000	001	010	011	100	101	110	111
000	ADDAB	ADDAH	ADDAW	ADDAD	SUBSAB	SUBSAH	SUBSAW	SUBSAD
001	ADDASB	ADDASH	ADDASW	ADDASD	SUBUAB	SUBUAH	SUBUAW	SUBUAD
010	ADDSSB	ADDSSH	ADDSSW	ADDSSD	SUBSSB	SUBSSH	SUBSSW	SUBSSD
011	ADDUUB	ADDUUH	ADDUUW	ADDUUD	SUBUUB	SUBUUH	SUBUUW	SUBUUD
100	ADDB	ADDH	ADDW	ADDD	SUBUSB	SUBUSH	SUBUSW	SUBUSD
101	SLLB	SLLH	SLLW	SLLD	SUBB	SUBH	SUBW	SUBD
110	AVESB	AVESH	AVESW	AVESD	AVERSB	AVERSH	AVERSW	AVERSD
111	AVEUB	AVEUH	AVEUW	AVEUD	AVERUB	AVERUH	AVERUW	AVERUD

- Encoding of funct0 and fmt Field for MXU2 3RINT-2 Instruction Formats

funct0,fmt	***bits[2:0]***							
bits[5:3]	000	001	010	011	100	101	110	111
000	DIVRSB	DIVRSH	DIVRSW	DIVRSD	MULB	MULH	MULW	MULD
001	DIVUB	DIVUH	DIVUW	DIVUD	MADDB	MADDH	MADDW	MADDD
010	MODSB	MODSH	MODSW	MODSD	MSUBB	MSUBH	MSUBW	MSUBD
011	MODUB	MODUH	MODUW	MODUD	REPXB	REPXH	REPXW	REPXD
100		DOTPSH	DOTPSW	DOTPSD		DADDSH	DADDSW	DADDSD
101		DOTPUH	DOTPUW	DOTPUD		DADDUH	DADDUW	DADDUD
110						DSUBSH	DSUBSW	DSUBSD

- Encoding of funct0 Field for MXU2 3RVEC Instruction Formats

funct0	***bits[2:0]***							
bits[5:3]	000	001	010	011	100	101	110	111

000								
001								
010								
011								
100								
101								
110								
111	ANDV	NORV	ORV	XORV				

- Encoding of funct0 Field for MXU2 3RFP Instruction Formats

funct0	***bits[2:0]***							
bits[5:3]	000	001	010	011	100	101	110	111
000	FADDW	FADDD	FSUBW	FSUBD	FMULW	FMULD	FDIVW	FDIVRD
001	FMADDW	FMADDD	FMSUBS	FMSUBD	VCVTHS	VCVTSD	VCVTQHS	VCVTQWD
010	FCORW	FCORD	FCEQW	FCEQD	FCLTW	FCLTD	FCLEW	FCLED
011	FMAXS	FMAXD	FMAXAS	FMAXAD	FMINS	FMIND	FMINAS	FMINAD
100								
101	MULQH	MULQW	MULQRH	MULQRW				
110	MADDQH	MADDQW	MADDQRH	MADDQRW	MSUBQH	MSUBQW	MSUBQRH	MSUBQRW
111								

- Encoding of funct0 and fmt Field for MXU2 2R Instruction Formats when funct1=0(2RINT)

funct0,fmt	***bits[2:0]***							
bits[5:3]	000	001	010	011	100	101	110	111
000	CEQZB	CEQZH	CEQZW	CEQZD	CNEZB	CNEZH	CNEZW	CNEZD
001	CLTZB	CLTZH	CLTZW	CLTZD	CLEZB	CLEZH	CLEZW	CLEZD
010	LOCB	LOCH	LOCW	LOCD	LZCB	LZCH	LZCW	LZCD
011								
100								
101								
110	BCNTB	BCNTH	BCNTW	BCNTD				
111					MFCPUB	MFCPUH	MFCPUW	

- Encoding of funct0 and fmt Field for MXU2 2R Instruction Formats when funct1=1（2RFP)

funct0,fmt	***bits[2:0]***							
bits[5:3]	000	001	010	011	100	101	110	111
000	FSQRTW	FSQRTD					FCLASSW	FCLASSD
001	VCVTSSW	VCVTSDL	VCVTUSW	VCVTUDL	VCVTSWS	VCVTSLD	VCVTUWS	VCVTULD
010					VTRUNCSWS	VTRUNCSLD	VTRUNCUWS	VTRUNCULD
011					VCVTRWS	VCVTRLD		
100	VCVTESH	VCVTEDS						
101	VCVTOSH	VCVTODS						

110	VCVTQESH	VCVTQEDW						
111	VCVTQOSH	VCVTQDW			CTCMXU	CTCMXU	MFFPUW	MFFPUD

- Encoding of funct0 and fmt Field for MXU2 2R5I Instruction Formats

funct0,fmt	***bits[2:0]***							
bits[5:3]	000	001	010	011	100	101	110	111
000	INSFFPUW	INSFFPUD			MTFPUW	MTFPUD		
001								
010								
011								
100								
101								
110								
111								

Revision History

Revision	Date	Description
02.15	June,2,2017	● No technical content changes ● Change the name MXU into MXU2
02.16	Spe,4,2017	● Replace OR1Q to ORV