

Playstation 2 Vector Unit Instruction Manual

by BigBoss

Layout changes and additions by Jules (jules@mouthshut.net)

Updates

1/7/2003 - Initial Version

Vector Units : Introduction

GENERAL

The VU0 is the first of two Vector Processing Units in PS2. It's designed to operate in two modes : as the second coprocessor for the 5900 core or independently as a microprocessor. Operating as a coprocessor the VU0 utilises either the upper or the lower instruction in each 64-bit microcode stored in it's instruction cache. Operating as an independent microprocessor the VU0 receives data fed from DMA to VIF0 and finally to the VU0 instruction or data cache. In microprocessor mode the VU0 utilises both upper and lower microcode instructions.

The VU1 is an independent microprocessor operating in parallel to the CPU. It executes the program in its instruction cache on demand of a VIF1 program. VU1 programmes use both upper and lower microcode instructions and has additional commands to the VU0, because of the EPU and the path to the GIF.

REGISTERS

The VU0 has 32 128 bit floating point registers , 16 32 bit integer data registers and 16 control registers.

FLOATING POINT REGISTERS

The floating point registers are 128 bits wide and each is divided in 4 fields of 32 bits : x , y , z and w.

FLOATING POINT REGISTERS FIELD LAYOUT

Word 0				Word 1				Word 2				Word 3			
31				0 63				32 95				64 127			
x				y				z				w			

VU0F00 AND VU1F00 REGISTERS

Word 0				Word 1				Word 2				Word 3			
31				0 63				32 95				64 127			
0.0				0.0				0.0				1.0			

The Floating registers can be represented in 4 fixed point formats:

Mode	Meaning
Fixed 0	fixed point 0 bits
Fixed 4	fixed point 4 bits
Fixed 12	fixed point 12 bits
Fixed 15	fixed point 15 bits

INTEGER DATA REGISTERS

The Integer data registers are 32 bits wide but only the lower halfword is used as the integer registers work as pointers to the instruction or data cache.

INTEGER CONTROL REGISTERS

The Integer control registers are:

Status flag
MAC flag
clipping flag
reserved
R
I
Q
reserved
reserved
reserved
TPC
CMSAR0
FBRST
VPU_STAT
reserved
CMSAR1

Status Flag

Bits : 32
Bits used : lower 12
Instructions : FSAND rt,12 bit immediate
FSEQ rt,12 bit immediate
FSOR rt,12 bit immediate
FSSET 12 bit immediate

MAC flag

Bits : 32
Bits used : lower 16
Instructions : FMAND rt,rs
 FMEQ rt,rs
 FMOR rt,rs

Clipping flag

Bits : 32
Bits used : lower 24
Instructions : FCAND rt,24 bit immediate
 FCEQ rt,24 bit immediate
 FCGET rt
 FCOR rt,24 bit immediate
 FCSET 24 bit immediate

R register

Bits : 32
Bits used : 32
Instructions : RINIT R,Frs.e
 RGET Frt,R
 RNEXT Frt,R
 RXOR R,Frs

I register

Bits : 32
Bits used : 32 (floating point)
Instructions : ADDI Upper
 SUBI Upper
 MULI Upper
 MADDI Upper
 MSUBI Upper
 ADDAI Upper
 SUBAI Upper
 MULAI Upper
 MADDAI Upper
 MSUBAI Upper
 MAXII Upper
 MINII Upper

Q register

Bits : 32
Bits used : 32 (floating point)
Instructions : ADDQ Upper
SUBQ Upper
MULQ Upper
MADDQ Upper
MSUBQ Upper
ADDAQ Upper
SUBAQ Upper
MULAQ Upper
MADDAQ Upper
MSUBAQ Upper
DIV Q,rt,rs Lower
SQRT Q,rs Lower
RSQRT Q,rs Lower

TPC ????? 2 Bytes used
FBRST ????? sceGSreset sets the bits in this disabling break
VPU-STAT ?????
CMSAR0 ?????
CMSAR1 ?????

Reading the manual

It can be quite confusing reading the manual for the first time so here is a quick introduction to understanding the manual and basics of VU.

First lets remember that there are 32 x 128 bit general purpose registers, each of these 128 bits registers is split into 4 x 32 bits (floats). These 32 bit parts of the 128 bit register is accessed by X,Y,Z,W (like a vector).

Register:

Bit: 128	96	64	32	0
W	Z	Y	X	

Register are named VF00 - VF31, VF00 is static and has the values; X = 0.0f, Y = 0.0f, Z = 0.0f, W = 1.0f

Now lets look at some documentation.

Add broadcast

ADDx ADDy ADDz ADDw

31 30 29 28 27 26 25 24 23 22 21 20	16 15	11 10	06 05	02 01 00
I E M D T 0 0 x y z w	ft	fs	fd	ADDbc bc
	0 0 dest			0000

Format: ADDx.dest fd,fs,ft
ADDy.dest fd,fs,ft
ADDz.dest fd,fs,ft
ADDw.dest fd,fs,ft

Purpose:
To add up to 4 fields with the broadcast field

Description: if dest & .x == true : fd.x <- fs.x + ft.bc
if dest & .y == true : fd.y <- fs.y + ft.bc
if dest & .z == true : fd.z <- fs.z + ft.bc
if dest & .w == true : fd.w <- fs.w + ft.bc

First, lets look at description, "if dest & .x == true", this means that if we can do

addy.x \$vf3x, \$vf4x, \$vf5y

This performs **\$vf3.x** (x part of vf3) = **\$vf4.x** (x part of vf4) + **\$vf5.y** (y part of vf5)

But what the description also says, is that we can combine different dests

addy.xyzw \$vf3xyzw, \$vf4xyzw, \$vf5y

This performs

$\$vf3.x = \$vf4.x + \$vf5.y$
 $\$vf3.y = \$vf4.y + \$vf5.y$
 $\$vf3.z = \$vf4.z + \$vf5.y$
 $\$vf3.w = \$vf4.w + \$vf5.y$

Or ..

addz.xw \$vf3xw, \$vf4xw, \$vf5z

Which performs

$\$vf3.x = \$vf4.x + \$vf5.z$
 $\$vf3.w = \$vf4.w + \$vf5.z$

You can also just write this instruction like this and still make sense because of the last z on \$vf5

addz.xw \$vf3, \$vf4, \$vf5z

As you can see VU is very flexible and as you learn more and read others code you will learn quite a few tricks which will result in fast VU code.

VU0 INSTRUCTION NOTE

This manual covers VU1 instructions, but the VU1 instructions which are available on VU0 are noted on the following pages. Just remember that you need to add a **V** as the first letter of the instruction, like

VU1 Instruction: **addy.x \$vf3x, \$vf4x, \$vf5y**

VU0 Instruction: **vaddy.x \$vf3x, \$vf4x, \$vf5y**

UPPER INSTRUCTION AVAILABILITY IN VU UNITS

Instruction	VU0 coprocessor	VU0 microprocessor	VU1	Instruction	VU0 coprocessor	VU0 microprocessor	VU1
ADD.dest	Yes	Yes	Yes	MSUB.dest	Yes	Yes	Yes
I.dest	Yes	Yes	Yes	Q.dest	Yes	Yes	Yes
Q.dest	Yes	Yes	Yes	I.dest	Yes	Yes	Yes
e.dest	Yes	Yes	Yes	e.dest	Yes	Yes	Yes
ADDA.dest	Yes	Yes	Yes	MSUBA.dest	Yes	Yes	Yes
I.dest	Yes	Yes	Yes	I.dest	Yes	Yes	Yes
Q.dest	Yes	Yes	Yes	Q.dest	Yes	Yes	Yes
e.dest	Yes	Yes	Yes	e.dest	Yes	Yes	Yes
SUB.dest	Yes	Yes	Yes	OPMULA.xyz	Yes	Yes	Yes
I.dest	Yes	Yes	Yes	OPMSUB.xyz	Yes	Yes	Yes
Q.dest	Yes	Yes	Yes	ABS.dest	Yes	Yes	Yes
e.dest	Yes	Yes	Yes	MAX.dest	Yes	Yes	Yes
SUBA.dest	Yes	Yes	Yes	I.dest	Yes	Yes	Yes
I.dest	Yes	Yes	Yes	e.dest	Yes	Yes	Yes
Q.dest	Yes	Yes	Yes	MINI.dest	Yes	Yes	Yes
e.dest	Yes	Yes	Yes	I.dest	Yes	Yes	Yes
MUL.dest	Yes	Yes	Yes	e.dest	Yes	Yes	Yes
I.dest	Yes	Yes	Yes	CLIPw.dest	Yes	Yes	Yes
Q.dest	Yes	Yes	Yes	ITOF0.dest	Yes	Yes	Yes
e.dest	Yes	Yes	Yes	4.dest	Yes	Yes	Yes
MULA.dest	Yes	Yes	Yes	12.dest	Yes	Yes	Yes
I.dest	Yes	Yes	Yes	15.dest	Yes	Yes	Yes
Q.dest	Yes	Yes	Yes	FTOI0.dest	Yes	Yes	Yes
e.dest	Yes	Yes	Yes	4.dest	Yes	Yes	Yes
MADD.dest	Yes	Yes	Yes	12.dest	Yes	Yes	Yes
I.dest	Yes	Yes	Yes	15.dest	Yes	Yes	Yes
Q.dest	Yes	Yes	Yes	NOP	Yes	Yes	Yes
e.dest	Yes	Yes	Yes				
MADDA.dest	Yes	Yes	Yes				
I.dest	Yes	Yes	Yes				
Q.dest	Yes	Yes	Yes				
e.dest	Yes	Yes	Yes				

LOWER INSTRUCTION AVAILABILITY IN VU UNITS

Instruction	VU0 coprocessor	VU0 microprocessor	VU1	Instruction	VU0 coprocessor	VU0 microprocessor	VU1
IADD	Yes	Yes	Yes	FCEQ	Yes	Yes	Yes
IADDI	Yes	Yes	Yes	FCGET	Yes	Yes	Yes
IADDIU	Yes	Yes	Yes	FCSET	Yes	Yes	Yes
ISUB	Yes	Yes	Yes	FSAND	Yes	Yes	Yes
ISUBIU	Yes	Yes	Yes	FSOR	Yes	Yes	Yes
DIV	Yes	Yes	Yes	FSEQ	Yes	Yes	Yes
SQRT	Yes	Yes	Yes	FSSET	Yes	Yes	Yes
RQSRT	Yes	Yes	Yes	FMAND	Yes	Yes	Yes
IAND	Yes	Yes	Yes	FMOR	Yes	Yes	Yes
IOR	Yes	Yes	Yes	FMEQ	Yes	Yes	Yes
B	Yes	Yes	Yes	RINIT	Yes	Yes	Yes
BAL	Yes	Yes	Yes	RGET.dest	Yes	Yes	Yes
IBEQ	Yes	Yes	Yes	RNEXT.dest	Yes	Yes	Yes
IBNE	Yes	Yes	Yes	RXOR	Yes	Yes	Yes
IBGTZ	Yes	Yes	Yes	WAITP	Yes	Yes	Yes
IBLEZ	Yes	Yes	Yes	WAITQ	Yes	Yes	Yes
IBGEZ	Yes	Yes	Yes	ESUM	No	No	Yes
JR	Yes	Yes	Yes	ESQRT	No	No	Yes
JALR	Yes	Yes	Yes	ESADD	No	No	Yes
ILW.dest	Yes	Yes	Yes	ERSQRT	No	No	Yes
ILWR.dest	Yes	Yes	Yes	ERSADD	No	No	Yes
ISW.dest	Yes	Yes	Yes	EATAN	No	No	Yes
ISWR.dest	Yes	Yes	Yes	EATANxz	No	No	Yes
LQ.dest	Yes	Yes	Yes	EATANxy	No	No	Yes
LQI.dest	Yes	Yes	Yes	ELENG	No	No	Yes
LQD.dest	Yes	Yes	Yes	ESIN	No	No	Yes
SQ.dest	Yes	Yes	Yes	ERLENG	No	No	Yes
SQI.dest	Yes	Yes	Yes	ERCPR	No	No	Yes
SQD.dest	Yes	Yes	Yes	EEXP	No	No	Yes
MOVE.dest	Yes	Yes	Yes	MFP.dest	No	No	Yes
MFIR.dest	Yes	Yes	Yes	XTOP	No	No	Yes
MTIR	Yes	Yes	Yes	XITOP	No	No	Yes
MR32.dest	Yes	Yes	Yes	XGKICK	No	No	Yes
FCAND	Yes	Yes	Yes				
FCOR	Yes	Yes	Yes				

UPPER VU1 INSTRUCTION ENCODING

UPPER VU1 INSTRUCTION LAYOUT

31	30	29	28	27	26	25	24	23	22	21	20			16	15			11	10			06	05			02	01	00
I	E	M	D	T	0	0	x	y	z	w		ft			fs				fd				OPCODE		bc			
flags					0	0	dest																					

UPPER VU1 INSTRUCTION FLAGS

FLAG	USAGE
i	Floating point value in lower microcode
e	Unknown
m	Unknown
d	Breakpoint. The code breaks after the current instruction executes
t	Unknown

Absolute

ABS

31 30 29 28 27 26 25 24 23 22 21 20	16 15	11 10	06 05	02 01 00
I E M D T 0 0 x y z w	ft	fs	ABS	SPECIAL bc
	0 0	dest		00111 1111 01

Format: ABS.dest fs,ft

Purpose:

To calculate the absolute values of up to 4 fields

Description: if dest & .x == true : fs.x <- abs(ft.x)
if dest & .y == true : fs.y <- abs(ft.y)
if dest & .z == true : fs.z <- abs(ft.z)
if dest & .w == true : fs.w <- abs(ft.w)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Add

ADD

31 30 29 28 27 26 25 24 23 22 21 20	16 15	11 10	06 05	00
I E M D T 0 0 x y z w	ft	fs	fd	ADD
				101000

Format: ADD.dest fd,fs,ft

Purpose:

To add up to 4 matching fields

Description: if dest & .x == true : fd.x <- fs.x + ft.x
if dest & .y == true : fd.y <- fs.y + ft.y
if dest & .z == true : fd.z <- fs.z + ft.z
if dest & .w == true : fd.w <- fs.w + ft.w

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Add I

ADDI

31	30	29	28	27	26	25	24	23	22	21	20	16		15	11		10	06		05	00	
I	E	M	D	T	0	0	x	y	z	w	0	fs			fd			ADDI				
					0	0	dest				00000							100010				

Format: ADDI.dest fd,fs,I

Purpose:
To add up to 4 fields with the I register

Description: if dest & .x == true : fd.x <- fs.x + I
if dest & .y == true : fd.y <- fs.y + I
if dest & .z == true : fd.z <- fs.z + I
if dest & .w == true : fd.w <- fs.w + I

Restrictions:
None

Operation:

Exceptions:

Programming notes:
This instruction is applicable to both VU0 and VU1

Add Q

ADDQ

31	30	29	28	27	26	25	24	23	22	21	20	16		15	11		10	06		05	00	
I	E	M	D	T	0	0	x	y	z	w	0	fs			fd			ADDQ				
					0	0	dest			00000								100000				

Format: ADDQ.dest fd,fs,Q

Purpose:
To add up to 4 fields with the Q register

Description: if dest & .x == true : fd.x <- fs.x + Q
if dest & .y == true : fd.y <- fs.y + Q
if dest & .z == true : fd.z <- fs.z + Q
if dest & .w == true : fd.w <- fs.w + Q

Restrictions:
None

Operation:

Exceptions:

Programming notes:
This instruction is applicable to both VU0 and VU1

Add broadcast

ADDx ADDy ADDz ADDw

31	30	29	28	27	26	25	24	23	22	21	20		16	15		11	10		06	05		02	01	00	
I	E	M	D	T	0	0	x	y	z	w		ft		fs			fd			ADDbc	bc				
					0	0			dest											0000					

Format: ADDx.dest fd,fs,ft
ADDy.dest fd,fs,ft
ADDz.dest fd,fs,ft
ADDw.dest fd,fs,ft

Purpose:
To add up to 4 fields with the broadcast field

Description: if dest & .x == true : fd.x <- fs.x + ft.bc
if dest & .y == true : fd.y <- fs.y + ft.bc
if dest & .z == true : fd.z <- fs.z + ft.bc
if dest & .w == true : fd.w <- fs.w + ft.bc

Restrictions:
None

Operation:

Exceptions:

Programming notes:
This instruction is applicable to both VU0 and VU1

Add accumulator

ADDA

31	30	29	28	27	26	25	24	23	22	21	20				16	15				11	10				06	05				02	01	00
I	E	M	D	T	0	0	x	y	z	w	ft					fs					ADDA					SPECIAL					bc	
					0	0	dest													01010					1111					00		

Format: ADDA.dest ACC,fs,ft

Purpose:
To add up to 4 matching fields

Description: if dest & .x == true : ACC.x <- fs.x + ft.x
 if dest & .y == true : ACC.y <- fs.y + ft.y
 if dest & .z == true : ACC.z <- fs.z + ft.z
 if dest & .w == true : ACC.w <- fs.w + ft.w

Restrictions:
None

Operation:

Exceptions:

Programming notes:
This instruction is applicable to both VU0 and VU1

Add accumulator I

ADDAI

31	30	29	28	27	26	25	24	23	22	21	20	16		15	11		10	06		05	02		01	00
I	E	M	D	T	0	0	x	y	z	w	0			fs			ADDAI			SPECIAL	bc			
					0	0	dest				00000						01000			1111	10			

Format: ADDAI.dest ACC,fs,I

Purpose:

To add up to 4 fields with the I register

Description: if dest & .x == true : ACC.x <- fs.x + I
if dest & .y == true : ACC.y <- fs.y + I
if dest & .z == true : ACC.z <- fs.z + I
if dest & .w == true : ACC.w <- fs.w + I

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Add accumulator Q

ADDAQ

31	30	29	28	27	26	25	24	23	22	21	20	16	15	11	10	06	05	02	01	00
I	E	M	D	T	0	0	x	y	z	w	0		fs		ADDAQ		SPECIAL	bc		
					0	0		dest			00000				01000		1111	00		

Format: ADDAQ.dest ACC,fs,Q

Purpose:

To add up to 4 fields with the Q register

Description: if dest & .x == true : ACC.x <- fs.x + Q
if dest & .y == true : ACC.y <- fs.y + Q
if dest & .z == true : ACC.z <- fs.z + Q
if dest & .w == true : ACC.w <- fs.w + Q

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Add accumulator broadcast

ADDx ADDy ADDz ADDw

31	30	29	28	27	26	25	24	23	22	21	20		16	15		11	10		06	05		02	01	00
I	E	M	D	T	0	0	x	y	z	w		ft			fs			ADDAbc		SPECIAL		bc		
					0	0			dest									00000		1111		00		

Format: ADDx.dest ACC,fs,ft
ADDy.dest ACC,fs,ft
ADDz.dest ACC,fs,ft
ADDw.dest ACC,fs,ft

Purpose:
To add up to 4 fields with the broadcast field

Description: if dest & .x == true : ACC.x <- fs.x + ft.bc
if dest & .y == true : ACC.y <- fs.y + ft.bc
if dest & .z == true : ACC.z <- fs.z + ft.bc
if dest & .w == true : ACC.w <- fs.w + ft.bc

Restrictions:
None

Operation:

Exceptions:

Programming notes:
This instruction is applicable to both VU0 and VU1

Clip

CLIPw

31 30 29 28 27 26 25 24 23 22 21 20	16 15	11 10	06 05	02 01 00
I E M D T 0 0 x y z w	ft		CLIPw	SPECIAL bc
	0 0 dest		00111	1111 11

Format: CLIPw.dest ft,fs

Purpose:

To update the status flag about the clipping condition of the destination values

Description:

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Floating to integer

FTOI0 FTOI4 FTOI12 FTOI15

31 30 29 28 27 26 25 24 23 22 21 20	16 15	11 10	06 05	02 01 00
I E M D T 0 0 x y z w	ft	fs	FTOI	SPECIAL bc
0 0 dest			00101	1111

Format: FTOI0.dest It,fs
 FTOI4.dest It,fs
 FTOI12.dest It,fs
 FTOI15.dest It,fs

Purpose:

To convert a floating point value to a fixed point (integer) number of 0, 4, 12 or 15 decimal digits

Description: if dest & .x == true : ft <- FixedPoint0/4/12/15(fs.x)
 elseif dest & .y == true : ft <- FixedPoint0/4/12/15(fs.y)
 elseif dest & .z == true : ft <- FixedPoint0/4/12/15(fs.z)
 elseif dest & .w == true : ft <- FixedPoint0/4/12/15(fs.w)

Restrictions:

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1. It is primarily used with the Random number generator

The broadcast field specifies the number of decimal digits

Broadcast field	Meaning
00	0 decimal digits
01	4 decimal digits
10	12 decimal digits
11	15 decimal digits

Integer to floating point

ITOF0 ITOF4 ITOF12 ITOF15

31	30	29	28	27	26	25	24	23	22	21	20		16	15		11	10		06	05		02	01	00
I	E	M	D	T	0	0	x	y	z	w		ft		fs			ITOF		SPECIAL		bc			
					0	0			dest								00100			1111				

Format: ITOF0.dest ft,fs
 ITOF4.dest ft,fs
 ITOF12.dest ft,fs
 ITOF15.dest ft,fs

Purpose:

To convert a fixed point (integer) value of 0, 4, 12 or 15 decimal digits to a floating point number

Description: if dest & .x == true : ft.x <- FixedPoint0/4/12/15(fs)
 elseif dest & .y == true : ft.y <- FixedPoint0/4/12/15(fs)
 elseif dest & .z == true : ft.z <- FixedPoint0/4/12/15(fs)
 elseif dest & .w == true : ft.w <- FixedPoint0/4/12/15(fs)

Restrictions:

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1. It is primarily used with the Random number generator

The broadcast field specifies the number of decimal digits

Broadcast field	Meaning
00	0 decimal digits
01	4 decimal digits
10	12 decimal digits
11	15 decimal digits

Multiply and add

MADD

31	30	29	28	27	26	25	24	23	22	21	20				16	15				11	10				06	05				00
I	E	M	D	T	0	0	x	y	z	w	ft					fs					fd					MADD				
					0	0	dest																			101001				

Format: MADD.dest fd,fs,ft

Purpose:

To multiply up to 4 matching fields and add the products with matching fields of the accumulator

Description: if dest & .x == true : fd.x <- ACC.x + (fs.x * ft.x)
if dest & .y == true : fd.y <- ACC.y + (fs.y * ft.y)
if dest & .z == true : fd.z <- ACC.z + (fs.z * ft.z)
if dest & .w == true : fd.w <- ACC.w + (fs.w * ft.w)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Multiply and add I

MADDI

31	30	29	28	27	26	25	24	23	22	21	20	16	15	11	10	06	05	00
I	E	M	D	T	0	0	x	y	z	w	0	fs	fd	MADDI				
				0	0		dest				00000					100011		

Format: MADDI.dest fd,fs,I

Purpose:

To multiply up to 4 fields with the I register and add the products with matching fields of the accumulator

Description: if dest & .x == true : fd.x <- ACC.x + (fs.x * I)
if dest & .y == true : fd.y <- ACC.y + (fs.y * I)
if dest & .z == true : fd.z <- ACC.z + (fs.z * I)
if dest & .w == true : fd.w <- ACC.w + (fs.w * I)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Multiply and add Q

MADDQ

31	30	29	28	27	26	25	24	23	22	21	20	16	15	11	10	06	05	00
I	E	M	D	T	0	0	x	y	z	w	0	fs		fd		MADDQ		
					0	0		dest			00000					100001		

Format: MADDQ.dest fd,fs,Q

Purpose:

To multiply up to 4 fields with the Q register and add the products with matching fields of the accumulator

Description: if dest & .x == true : fd.x <- ACC.x + (fs.x * Q)
if dest & .y == true : fd.y <- ACC.y + (fs.y * Q)
if dest & .z == true : fd.z <- ACC.z + (fs.z * Q)
if dest & .w == true : fd.w <- ACC.w + (fs.w * Q)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Multiply and add broadcast

MADDx MADDy MADDz MADDw

31	30	29	28	27	26	25	24	23	22	21	20		16	15		11	10		06	05		02	01	00
I	E	M	D	T	0	0	x	y	z	w		ft		fs		fd				MADDbc		bc		
					0	0			dest											0010				

Format: MADDx.dest fd,fs,ft
MADDy.dest fd,fs,ft
MADDz.dest fd,fs,ft
MADDw.dest fd,fs,ft

Purpose:

To multiply up to 4 fields with the broadcast field and add the products with matching fields of the accumulator

Description: if dest & .x == true : fd.x <- ACC.x + (fs.x * ft.bc)
if dest & .y == true : fd.y <- ACC.y + (fs.y * ft.bc)
if dest & .z == true : fd.z <- ACC.z + (fs.z * ft.bc)
if dest & .w == true : fd.w <- ACC.w + (fs.w * ft.bc)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Multiply and add accumulator

MADDA

31	30	29	28	27	26	25	24	23	22	21	20		16	15		11	10		06	05		02	01	00
I	E	M	D	T	0	0	x	y	z	w		ft		fs			MADDA		SPECIAL		bc			
					0	0			dest								01010			1111		01		

Format: MADDA.dest ACC,fs,ft

Purpose:

To multiply up to 4 fields and add the products with matching fields in the accumulator

Description: if dest & .x == true : ACC.x <- ACC.x + (fs.x * ft.x)
if dest & .y == true : ACC.y <- ACC.y + (fs.y * ft.y)
if dest & .z == true : ACC.z <- ACC.z + (fs.z * ft.z)
if dest & .w == true : ACC.w <- ACC.w + (fs.w * ft.w)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Multiply and add accumulator I

MADDAI

31	30	29	28	27	26	25	24	23	22	21	20	16				15	11				10	06				05	02				01	00			
I	E	M	D	T	0	0	x	y	z	w	0				fs				MADDAI				SPECIAL				bc								
												dest				00000								01000				1111				11			

Format: MADDAI.dest ACC,fs,I

Purpose:

To multiply up to 4 fields with the I register and add the products with matching fields in the accumulator

Description: if dest & .x == true : ACC.x <- ACC.x + (fs.x * I)
if dest & .y == true : ACC.y <- ACC.y + (fs.y * I)
if dest & .z == true : ACC.z <- ACC.z + (fs.z * I)
if dest & .w == true : ACC.w <- ACC.w + (fs.w * I)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Multiply and add accumulator Q

MADDAQ

31	30	29	28	27	26	25	24	23	22	21	20	16	15	11	10	06	05	02	01	00
I	E	M	D	T	0	0	x	y	z	w	0		fs		MADDAQ		SPECIAL	bc		
					0	0		dest			00000				01000		1111	01		

Format: MADDAI.dest ACC,fs,Q

Purpose:

To multiply up to 4 fields with the Q register and add the products with matching fields in the accumulator

Description: if dest & .x == true : ACC.x <- ACC.x + (fs.x * Q)
if dest & .y == true : ACC.y <- ACC.y + (fs.y * Q)
if dest & .z == true : ACC.z <- ACC.z + (fs.z * Q)
if dest & .w == true : ACC.w <- ACC.w + (fs.w * Q)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Multiply and add accumulator broadcast MADDx MADDy MADDz MADDw

31	30	29	28	27	26	25	24	23	22	21	20		16	15		11	10		06	05		02	01	00
I	E	M	D	T	0	0	x	y	z	w		ft			fs			MADDAbc		SPECIAL		bc		
					0	0				dest								01000		1111				

Format: MADDx.dest ACC,fs,ft
MADDy.dest ACC,fs,ft
MADDz.dest ACC,fs,ft
MADDw.dest ACC,fs,ft

Purpose:
To multiply up to 4 fields with the broadcast field and add the products with matching fields in the accumulator

Description: if dest & .x == true : ACC.x <- ACC.x + (fs.x * ft.x)
if dest & .y == true : ACC.y <- ACC.y + (fs.y * ft.y)
if dest & .z == true : ACC.z <- ACC.z + (fs.z * ft.z)
if dest & .w == true : ACC.w <- ACC.w + (fs.w * ft.w)

Restrictions:
None

Operation:

Exceptions:

Programming notes:
This instruction is applicable to both VU0 and VU1

Maximum

MAX

31	30	29	28	27	26	25	24	23	22	21	20				16	15				11	10				06	05				00
I	E	M	D	T	0	0	x	y	z	w	ft					fs					fd					MAX				
					0	0	dest																			101011				

Format: MAX.dest fd,fs,ft

Purpose:

To find the maximum value between up to 4 matching fields

Description: if dest & .x == true : fd.x <- max(fs.x : ft.bc)
if dest & .y == true : fd.y <- max(fs.y : ft.bc)
if dest & .z == true : fd.z <- max(fs.z : ft.bc)
if dest & .w == true : fd.w <- max(fs.w : ft.bc)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Maximum I

MAXI

31	30	29	28	27	26	25	24	23	22	21	20		16	15		11	10		06	05		00	
I	E	M	D	T	0	0	x	y	z	w		0		fs			fd			MAXI			
					0	0	dest				00000									011101			

Format: MAXI.dest fd,fs,I

Purpose:

To find the maximum value between up to 4 fields and the I register

Description: if dest & .x == true : fd.x <- max(fs.x : I)
if dest & .y == true : fd.y <- max(fs.y : I)
if dest & .z == true : fd.z <- max(fs.z : I)
if dest & .w == true : fd.w <- max(fs.w : I)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Maximum broadcast

MAXx MAXy MAXz MAXw

31	30	29	28	27	26	25	24	23	22	21	20		16	15		11	10		06	05		02	01	00	
I	E	M	D	T	0	0	x	y	z	w		ft		fs			fd			MAXbc		bc			
					0	0			dest											0100					

Format: MAXx.dest fd,fs,ft
MAXy.dest fd,fs,ft
MAXz.dest fd,fs,ft
MAXw.dest fd,fs,ft

Purpose:

To find the maximum value between up to 4 fields and the I register

Description: if dest & .x == true : fd.x <- max(fs.x : ft.bc)
if dest & .y == true : fd.y <- max(fs.y : ft.bc)
if dest & .z == true : fd.z <- max(fs.z : ft.bc)
if dest & .w == true : fd.w <- max(fs.w : ft.bc)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Minimum

MINI

31	30	29	28	27	26	25	24	23	22	21	20				16	15				11	10				06	05				00
I	E	M	D	T	0	0	x	y	z	w	ft					fs					fd					MINI				
					0	0	dest																			101111				

Format: MINI.dest fd,fs,ft

Purpose:

To find the minimum value between up to 4 matching fields

Description: if dest & .x == true : fd.x <- mini(fs.x : ft.x)
if dest & .y == true : fd.y <- mini(fs.y : ft.y)
if dest & .z == true : fd.z <- mini(fs.z : ft.z)
if dest & .w == true : fd.w <- mini(fs.w : ft.w)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Minimum I

MINII

31	30	29	28	27	26	25	24	23	22	21	20	16	15	11	10	06	05	00
I	E	M	D	T	0	0	x	y	z	w	0	fs	fd	MINII				
				0	0		dest				00000				011111			

Format: MINII.dest fd,fs,ft

Purpose:

To find the minimum value between up to 4 fields and the I register

Description: if dest & .x == true : fd.x <- mini(fs.x : I)
if dest & .y == true : fd.y <- mini(fs.y : I)
if dest & .z == true : fd.z <- mini(fs.z : I)
if dest & .w == true : fd.w <- mini(fs.w : I)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Minimum broadcast

MINIx MINIy MINIz MINIw

31	30	29	28	27	26	25	24	23	22	21	20		16	15		11	10		06	05		02	01	00	
I	E	M	D	T	0	0	x	y	z	w		ft		fs			fd			MINIbc		bc			
					0	0			dest											0101					

Format: MINIx.dest fd,fs,ft
MINIy.dest fd,fs,ft
MINIz.dest fd,fs,ft
MINIw.dest fd,fs,ft

Purpose:

To find the minimum value between up to 4 fields and the I register

Description: if dest & .x == true : fd.x <- mini(fs.x : ft.bc)
if dest & .y == true : fd.y <- mini(fs.y : ft.bc)
if dest & .z == true : fd.z <- mini(fs.z : ft.bc)
if dest & .w == true : fd.w <- mini(fs.w : ft.bc)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Multiply and subtract

MSUB

[illegible]

Format: MSUB.dest fd,fs,ft

Purpose:

To multiply up to 4 matching fields and subtract the products from matching fields of the accumulator

```
Description: if dest & .x == true : fd.x <- ACC.x - (fs.x * ft.x)
               if dest & .y == true : fd.y <- ACC.y - (fs.y * ft.y)
               if dest & .z == true : fd.z <- ACC.z - (fs.z * ft.z)
               if dest & .w == true : fd.w <- ACC.w - (fs.w * ft.w)
```

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Multiply and subtract I

MSUBI

31	30	29	28	27	26	25	24	23	22	21	20	16	15	11	10	06	05	00
I	E	M	D	T	0	0	x	y	z	w	0	fs	fd	MSUBI				
				0	0		dest				00000					100111		

Format: MSUBI.dest fd,fs,ft

Purpose:

To multiply up to 4 fields with the I register and subtract the products from matching fields of the accumulator

Description: if dest & .x == true : fd.x <- ACC.x - (fs.x * I)
if dest & .y == true : fd.y <- ACC.y - (fs.y * I)
if dest & .z == true : fd.z <- ACC.z - (fs.z * I)
if dest & .w == true : fd.w <- ACC.w - (fs.w * I)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Multiply and subtract Q

MSUBQ

31	30	29	28	27	26	25	24	23	22	21	20		16	15		11	10		06	05		00	
I	E	M	D	T	0	0	x	y	z	w		0		fs			fd			MSUBQ			
					0	0	dest					00000								100101			

Format: MSUBQ.dest fd,fs,ft

Purpose:

To multiply up to 4 fields with the Q register and subtract the products from matching fields of the accumulator

Description: if dest & .x == true : fd.x <- ACC.x - (fs.x * Q)
if dest & .y == true : fd.y <- ACC.y - (fs.y * Q)
if dest & .z == true : fd.z <- ACC.z - (fs.z * Q)
if dest & .w == true : fd.w <- ACC.w - (fs.w * Q)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Multiply and subtract broadcast

MSUBx MSUBy MSUBz MSUBw

31	30	29	28	27	26	25	24	23	22	21	20		16	15		11	10		06	05		02	01	00	
I	E	M	D	T	0	0	x	y	z	w		ft		fs			fd			MSUBbc		bc			
					0	0			dest											0011					

Format: MSUBx.dest fd,fs,ft
MSUBy.dest fd,fs,ft
MSUBw.dest fd,fs,ft
MSUBw.dest fd,fs,ft

Purpose:

To multiply up to 4 fields with the broadcast field and subtract the products from matching fields of the accumulator

Description: if dest & .x == true : fd.x <- ACC.x - (fs.x * ft.bc)
if dest & .y == true : fd.y <- ACC.y - (fs.y * ft.bc)
if dest & .z == true : fd.z <- ACC.z - (fs.z * ft.bc)
if dest & .w == true : fd.w <- ACC.w - (fs.w * ft.bc)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Multiply and subtract accumulator

MSUBA

31	30	29	28	27	26	25	24	23	22	21	20		16	15		11	10		06	05		02	01	00
I	E	M	D	T	0	0	x	y	z	w		ft		fs			MSUBA			SPECIAL		bc		
					0	0			dest								01011			1111		01		

Format: MSUBA.dest fd,fs,ft

Purpose:

To multiply up to 4 fields and subtract the products from matching fields in the accumulator

Description: if dest & .x == true : ACC.x <- ACC.x - (fs.x * ft.bc)
if dest & .y == true : ACC.y <- ACC.y - (fs.y * ft.bc)
if dest & .z == true : ACC.z <- ACC.z - (fs.z * ft.bc)
if dest & .w == true : ACC.w <- ACC.w - (fs.w * ft.bc)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Multiply and subtract accumulator I

MSUBAI

31	30	29	28	27	26	25	24	23	22	21	20	16	15	11	10	06	05	02	01	00
I	E	M	D	T	0	0	x	y	z	w	0		fs		MSUBAI		SPECIAL	bc		
					0	0		dest			00000				01001		1111	11		

Format: MSUBAI.dest fd,fs,ft

Purpose:

To multiply up to 4 fields with the I register and subtract the products from matching fields in the accumulator

Description: if dest & .x == true : ACC.x <- ACC.x - (fs.x * I)
if dest & .y == true : ACC.y <- ACC.y - (fs.y * I)
if dest & .z == true : ACC.z <- ACC.z - (fs.z * I)
if dest & .w == true : ACC.w <- ACC.w - (fs.w * I)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Multiply and subtract accumulator Q

MSUBAQ

31	30	29	28	27	26	25	24	23	22	21	20	16	15	11	10	06	05	02	01	00
I	E	M	D	T	0	0	x	y	z	w	0		fs		MSUBAQ		SPECIAL	bc		
					0	0		dest			00000				01001		1111	01		

Format: MSUBAQ.dest fd,fs,ft

Purpose:

To multiply up to 4 fields with the Q register and subtract the products from matching fields in the accumulator

Description: if dest & .x == true : ACC.x <- ACC.x - (fs.x * Q)
if dest & .y == true : ACC.y <- ACC.y - (fs.y * Q)
if dest & .z == true : ACC.z <- ACC.z - (fs.z * Q)
if dest & .w == true : ACC.w <- ACC.w - (fs.w * Q)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Multiply and subtract accumulator broadcast MSUBAx MSUBAy MSUBAz MSUBAw

31	30	29	28	27	26	25	24	23	22	21	20	16				15	11				10	06				05	02				01	00			
I	E	M	D	T	0	0	x	y	z	w	0				fs				MSUBAbc				SPECIAL				bc								
												dest				00000								00011				1111							

Format: MSUBAx.dest fd,fs,ft
MSUBAy.dest fd,fs,ft
MSUBAz.dest fd,fs,ft
MSUBAw.dest fd,fs,ft

Purpose:

To multiply up to 4 fields with the broadcast field and subtract the products from matching fields in the accumulator

Description: if dest & .x == true : ACC.x <- ACC.x - (fs.x * ft.bc)
if dest & .y == true : ACC.y <- ACC.y - (fs.y * ft.bc)
if dest & .z == true : ACC.z <- ACC.z - (fs.z * ft.bc)
if dest & .w == true : ACC.w <- ACC.w - (fs.w * ft.bc)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Multiply

MUL

31	30	29	28	27	26	25	24	23	22	21	20					16	15					11	10					06	05					00
I	E	M	D	T	0	0	x	y	z	w	ft						fs						fd						MUL					
					0	0	dest																			101010								

Format: MUL.dest fd,fs,ft

Purpose:
To multiply up to 4 matching fields

Description: if dest & .x == true : fd.x <- fs.x * ft.x
if dest & .y == true : fd.y <- fs.y * ft.y
if dest & .z == true : fd.z <- fs.z * ft.z
if dest & .w == true : fd.w <- fs.w * ft.w

Restrictions:
None

Operation:

Exceptions:

Programming notes:
This instruction is applicable to both VU0 and VU1

Multiply I

MULI

31	30	29	28	27	26	25	24	23	22	21	20	16	15	11	10	06	05	00
I	E	M	D	T	0	0	x	y	z	w	0	fs	fd	MULI				
				0	0		dest				00000					011110		

Format: MULI.dest fd,fs,ft

Purpose:

To multiply up to 4 fields with the I register

Description: if dest & .x == true : fd.x <- fs.x * I)
if dest & .y == true : fd.y <- fs.y * I)
if dest & .z == true : fd.z <- fs.z * I)
if dest & .w == true : fd.w <- fs.w * I)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Multiply Q

MULQ

31	30	29	28	27	26	25	24	23	22	21	20	16	15	11	10	06	05	00
I	E	M	D	T	0	0	x	y	z	w	0	fs		fd		MULQ		
					0	0	dest				00000					011100		

Format: MULQ.dest fd,fs,ft

Purpose:

To multiply up to 4 fields with the Q register

Description: if dest & .x == true : fd.x <- fs.x * Q)
if dest & .y == true : fd.y <- fs.y * Q)
if dest & .z == true : fd.z <- fs.z * Q)
if dest & .w == true : fd.w <- fs.w * Q)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Multiply broadcast

MULx MULy MULz MULw

31	30	29	28	27	26	25	24	23	22	21	20		16	15		11	10		06	05		02	01	00	
I	E	M	D	T	0	0	x	y	z	w		ft		fs			fd			MULbc		bc			
					0	0			dest											0110					

Format: MULx.dest fd,fs,ft
MULy.dest fd,fs,ft
MULz.dest fd,fs,ft
MULw.dest fd,fs,ft

Purpose:
To multiply up to 4 fields with the broadcast field

Description: if dest & .x == true : fd.x <- fs.x * ft.bc)
if dest & .y == true : fd.y <- fs.y * ft.bc)
if dest & .z == true : fd.z <- fs.z * ft.bc)
if dest & .w == true : fd.w <- fs.w * ft.bc)

Restrictions:
None

Operation:

Exceptions:

Programming notes:
This instruction is applicable to both VU0 and VU1

No operation

NOP

31	30	29	28	27	26	25	24	23	22	21	20	16	15	11	10	06	05	02	01	00
I	E	M	D	T	0	0	x	y	z	w		0		0		NOP		SPECIAL	bc	
					0	0	0	0	0	0		00000		00000		01011		1111	11	

Format: NOP

Purpose:
To perform no operation

Description:

Restrictions:
None

Operation:

Exceptions:

Programming notes:
This instruction is applicable to both VU0 and VU1

Outer product pre increment

OPMULA

31	30	29	28	27	26	25	24	23	22	21	20		16	15		11	10		06	05		02	01	00
I	E	M	D	T	0	0	x	y	z	w		ft		fs			OPMULA		SPECIAL		bc			
					0	0	1	1	1	0							01011		1111		10			

Format: OPMULA.xyz ACC,fs,ft

Purpose:

To calculate the outer product pre increment

Description: ACC.x <- (fs.y * ft.z) + (fs.z * ft.y)
ACC.y <- (fs.z * ft.x) + (fs.x * ft.z)
ACC.z <- (fs.x * ft.y) + (fs.y * ft.x)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Outer product post decrement

OPMSUB

31 30 29 28 27 26 25 24 23 22 21 20	16 15	11 10	06 05	00
I E M D T 0 0 x y z w	ft	fs	fd	OPMSUB
				011011

Format: OPMSUB.xyz fd,fs,ft

Purpose:

To calculate the outer product post decrement

Description: $fd.x \leftarrow (fs.y * ft.z) - (fs.z * ft.y)$
 $fd.y \leftarrow (fs.z * ft.x) - (fs.x * ft.z)$
 $fd.z \leftarrow (fs.x * ft.y) - (fs.y * ft.x)$

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Subtract

SUB

31	30	29	28	27	26	25	24	23	22	21	20			16	15			11	10			06	05			00
I	E	M	D	T	0	0	x	y	z	w		ft			fs				fd				SUB			
					0	0			dest														101100			

Format: SUB.dest fd,fs,ft

Purpose:

To subtract up to 4 matching fields

Description: if dest & .x == true : fd.x <- fs.x - ft.x
if dest & .y == true : fd.y <- fs.y - ft.y
if dest & .z == true : fd.z <- fs.z - ft.z
if dest & .w == true : fd.w <- fs.w - ft.w

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Substract I

SUBI

31	30	29	28	27	26	25	24	23	22	21	20	16	15	11	10	06	05	00
I	E	M	D	T	0	0	x	y	z	w	0		fs		fd		SUBI	
				0	0		dest				00000						100110	

Format: SUBI.dest fd,fs,Q

Purpose:

To substract the I register from up to 4 fields

Description: if dest & .x == true : fd.x <- fs.x - I
if dest & .y == true : fd.y <- fs.y - I
if dest & .z == true : fd.z <- fs.z - I
if dest & .w == true : fd.w <- fs.w - I

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Subtract Q

SUBQ

31 30 29 28 27 26 25 24 23 22 21 20	16 15	11 10	06 05	00
I E M D T 0 0 x y z w	0	fs	fd	SUBQ
	00000			100100

Format: SUBQ.dest fd,fs,Q

Purpose:

To subtract the Q register from up to 4 fields

Description: if dest & .x == true : fd.x <- fs.x - Q
if dest & .y == true : fd.y <- fs.y - Q
if dest & .z == true : fd.z <- fs.z - Q
if dest & .w == true : fd.w <- fs.w - Q

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Subtract broadcast

SUBx SUBy SUBz SUBw

31	30	29	28	27	26	25	24	23	22	21	20		16	15		11	10		06	05		02	01	00	
I	E	M	D	T	0	0	x	y	z	w		ft		fs			fd			SUBbc		bc			
					0	0			dest											0001					

Format: SUBx.dest fd,fs,ft
SUBy.dest fd,fs,ft
SUBz.dest fd,fs,ft
SUBw.dest fd,ft,fs

Purpose:
To subtract the broadcast field from up to 4 fields

Description: if dest & .x == true : fd.x <- fs.x - ft.bc
if dest & .y == true : fd.y <- fs.y - ft.bc
if dest & .z == true : fd.z <- fs.z - ft.bc
if dest & .w == true : fd.w <- fs.w - ft.bc

Restrictions:
None

Operation:

Exceptions:

Programming notes:
This instruction is applicable to both VU0 and VU1

Subtract accumulator

SUBA

31	30	29	28	27	26	25	24	23	22	21	20					16	15					11	10					06	05					02	01	00
I	E	M	D	T	0	0	x	y	z	w	ft					fs					SUBA				SPECIAL				bc							
												dest								01011				1111				00								

Format: SUBA.dest ACC,fs,ft

Purpose:

To subtract up to 4 matching fields to the accumulator

Description: if dest & .x == true : ACC.x <- fs.x - ft.x
if dest & .y == true : ACC.y <- fs.y - ft.y
if dest & .z == true : ACC.z <- fs.z - ft.z
if dest & .w == true : ACC.w <- fs.w - ft.w

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Subtract accumulator I

SUBAI

31	30	29	28	27	26	25	24	23	22	21	20	16				15	11				10	06				05	02				01	00
I	E	M	D	T	0	0	x	y	z	w	0				fs				SUBAI				SPECIAL				bc					
												dest				00000				01001				1111				10				

Format: SUBAI.dest ACC,fs,I

Purpose:

To subtract the I register from up to 4 matching fields to the accumulator

Description: if dest & .x == true : ACC.x <- fs.x - I
if dest & .y == true : ACC.y <- fs.y - I
if dest & .z == true : ACC.z <- fs.z - I
if dest & .w == true : ACC.w <- fs.w - I

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Subtract accumulator Q

SUBAQ

31	30	29	28	27	26	25	24	23	22	21	20		16	15		11	10		06	05		02	01	00
I	E	M	D	T	0	0	x	y	z	w		0		fs			SUBAQ		SPECIAL		bc			
					0	0	dest				00000						01001		1111		00			

Format: SUBAQ.dest ACC,fs,Q

Purpose:

To subtract the Q register from up to 4 matching fields to the accumulator

Description: if dest & .x == true : ACC.x <- fs.x - Q
if dest & .y == true : ACC.y <- fs.y - Q
if dest & .z == true : ACC.z <- fs.z - Q
if dest & .w == true : ACC.w <- fs.w - Q

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Subtract accumulator broadcast

SUBAx SUBAy SUBAz SUBAw

31	30	29	28	27	26	25	24	23	22	21	20		16	15		11	10		06	05		02	01	00	
I	E	M	D	T	0	0	x	y	z	w		ft			fs			SUBAbc		SPECIAL		bc			
					0	0			dest									00001		1111					

Format: SUBAx.dest ACC,fs,ft
SUBAy.dest ACC,fs,ft
SUBAz.dest ACC,fs,ft
SUBAw.dest ACC,ft,fs

Purpose:
To subtract up to matching 4 fields to the accumulator

Description: if dest & .x == true : ACC.x <- fs.x - ft.bc
if dest & .y == true : ACC.y <- fs.y - ft.bc
if dest & .z == true : ACC.z <- fs.z - ft.bc
if dest & .w == true : ACC.w <- fs.w - ft.bc

Restrictions:
None

Operation:

Exceptions:

Programming notes:
This instruction is applicable to both VU0 and VU1

LOWER VU1 INSTRUCTION ENCODING

LOWER VU1 INSTRUCTION LAYOUT

31	25 24 23 22 21 20					16 15		11 10		06 05		02 01 00			
	OPCODE					x y z w	rt		rs		rd		SPECIAL	bc	
						t.e s.e									

ft.e and fs.e : 00 x
 01 y
 11 w

Branch

B

31		25 24 23 22 21 20		16 15		11 10		00
	B	x y z w	0		0		offset	
	0100000	0 0 0 0	00000		00000			

Format: B offset

Purpose:

To unconditionally branch to a VU instruction address

Description: branch -> PC + offset + 8

Restrictions:

None

Operation:

Exceptions:

Programming notes:

The offset refers to doublewords in the VU instruction memory.
This instruction is applicable to both VU0 and VU1

Branch and link

BAL

31		25 24 23 22 21 20		16 15		11 10		00
	BAL	x y z w	It		0		offset	
	0100001	0 0 0 0			00000			

Format: BAL It, offset

Purpose:

To unconditionally branch to a VU instruction address and link to an IR

Description: branch -> PC + offset + 8
It -> PC + 16

Restrictions:

None

Operation:

Exceptions:

Programming notes:

The offset refers to doublewords in the VU instruction memory.
This instruction is applicable to both VU0 and VU1

Divide

DIV

31		25 24 23 22 21 20		16 15		11 10		06 05		02 01 00
	SPECIAL	x y z w	ft		fs		DIV	SPECIAL	bc	
	1000000	t.e s.e					01110	1111	00	

Format: DIV Q,ft.e,fs.e

Purpose:

To divide a field of a VU FPR with a field of a VU FPR and store the result in the Q register

Description: $Q \leftarrow ft.e/fs.e$

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Archtangent

EATAN

31				25 24 23 22 21 20			16 15			11 10			06 05			02 01 00
	SPECIAL		x	y z w		0		fs			EATAN		SPECIAL	bc		
	1000000		0	0 s.e		00000					11111		1111	01		

Format: EATAN P, fs.e

Purpose:

To calculate the archtangent of a single field in a VU FPR and store it in the P register

Description: P <- arch(fs.e)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable only to VU1

Archtangent xy

EATANxy

31		25 24 23 22 21 20		16 15		11 10		06 05		02 01 00
	SPECIAL	x y z w	0		fs		EATAN	SPECIAL	bc	
	1000000	1 1 0 0	00000				11101	1111	00	

Format: EATANxy P, fs

Purpose:

To calculate the archtangent of two fields in a VU FPR and store it in the P register

Description: P <- arch(fs.x, fs.y)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable only to VU1

Archtangent xz

EATANxz

31		25 24 23 22 21 20		16 15		11 10		06 05		02 01 00
	SPECIAL	x y z w	0		fs		EATAN		SPECIAL	bc
	1000000	1 0 1 0	00000				11101		1111	01

Format: EATANxy P, fs

Purpose:

To calculate the archtangent of two fields in a VU FPR and store it in the P register

Description: P <- arch(fs.x, fs.z)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable only to VU1

Exponent

EEXP

31		25 24 23 22 21 20		16 15		11 10		06 05		02 01 00
	SPECIAL	x y z w	0		fs		EEXP	SPECIAL	bc	
	1000000	0 0 s.e	00000				11111	1111	10	

Format: EEXP P, fs.e

Purpose:

To calculate the exponent of a single field of a VU FPR and store it in the P register

Description: P <- exp(fs.e)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable only to VU1

Length

ELENG

31		25 24 23 22 21 20		16 15		11 10		06 05		02 01 00
	SPECIAL	x y z w	0		fs		ELENG		SPECIAL	bc
	1000000	1 1 1 0	00000				11100		1111	10

Format: ELENG P, fs

Purpose:

To calculate the length for the values in a VU FPR

Description: P <- len(fs)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable only to VU1

Reciprocal root

ERCPR

31		25 24 23 22 21 20		16 15		11 10		06 05		02 01 00
	SPECIAL	x y z w	0		fs		ERCPR		SPECIAL	bc
	1000000	0 0 s.e	00000				11110		1111	10

Format: ERCPR P, fs.e

Purpose:

To calculate the reciprocal root for a single field of a VU FPR and store it in the P register

Description: `P <- rec.root(fs.e)`

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable only to VU1

Reverse length

ERLENG

31		25 24 23 22 21 20		16 15		11 10		06 05		02 01 00
	SPECIAL	x y z w	0		fs		ERLENG	SPECIAL	bc	
	1000000	1 1 1 0	00000				11100	1111	11	

Format: ERLENG P, fs.xyz

Purpose:

To calculate the reverse length for the xyz fields of a VU FPR and store it in the P register

Description: P <- rev.len(fs.xyz)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable only to VU1

Reverse square root add

ERSADD

31		25 24 23 22 21 20		16 15		11 10		06 05		02 01 00
	SPECIAL	x y z w	0		fs		ERSADD		SPECIAL	bc
	1000000	1 1 1 0	00000				11100		1111	01

Format: ERSADD P,fs

Purpose:

To calculate the reverse square root of the addition of the xyz fields of a VU FPR and store it in the P register

Description: $P \leftarrow \text{rev.sq.root}(fs.x + fs.y + fs.z)$

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable only to VU1

Square root

ESQRT

31		25 24 23 22 21 20		16 15		11 10		06 05		02 01 00
	SPECIAL	x y z w	0		fs		ESQRT		SPECIAL	bc
	1000000	0 0 s.e	00000				11110		1111	01

Format: ESQRT P, fs.e

Purpose:

To calculate the square root of a single field of a VU FPR and store it in the P register

Description: P <- sq.root(fs.e)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable only to VU1

Square root add

ESADD

31	25 24 23 22 21 20					16 15	11 10		06 05		02 01 00					
	SPECIAL		x	y	z	w	0		fs		ESADD		SPECIAL		bc	
	1000000		1	1	1	0	00000				11100		1111		00	

Format: ESADD P, fs.xyz

Purpose:

To calculate the square root of the sum of the xyz fields of a VU FPR and store it in the P register

Description: $P \leftarrow \text{sq.root}(fs.x + fs.y + fs.z)$

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable only to VU1

Sin

SIN

31		25 24 23 22 21 20		16 15		11 10		06 05		02 01 00
	SPECIAL	x y z w	0		fs		ESIN	SPECIAL	bc	
	1000000	0 0 s.e	00000				11111	1111	00	

Format: ESIN P, fs.e

Purpose:

To calculate the sin of a single field of VU FPR and store it in the P register

Description: P <- sin(fs.e)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable only to VU1

Square root

ESQRT

31		25 24 23 22 21 20		16 15		11 10		06 05		02 01 00
	SPECIAL	x y z w	0		fs		ESQRT		SPECIAL	bc
	1000000	0 0 s.e	00000				11110		1111	00

Format: ESQRT P, fs.e

Purpose:

To calculate the square root of a single field of VU FPR and store it in the P register

Description: P <- sq.root(fs.e)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable only to VU1

Sum

ESUM

31		25 24 23 22 21 20		16 15		11 10		06 05		02 01 00
	SPECIAL	x y z w	0		fs		ESUM		SPECIAL	bc
	1000000	1 1 1 1	00000				11101		1111	10

Format: ESUM P,fs

Purpose:

To calculate the sum of all the fields of a VU FPR and store it in the P register

Description: $P \leftarrow fs.x + fs.y + fs.z + fs.w$

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable only to VU1

Clipping AND

FCAND

31	25	24	23	00
FCAND	x	immediate		
0010010	0			

Format: FCAND VI01,imm24

Purpose:

To perform a logical AND between the VI01 register and the 24 bit immediate and update the clipping flag

Description: clipping flag(VI01 AND imm24)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Clipping equality

FCEQ

31	25	24	23		00	
	FCEQ	x		immediate		
	0010000	0				

Format: FCEQ VI01,imm24

Purpose:

To perform an equality check between the VI01 register and the 24 bit immediate and only update the clipping flag on equality

Description: clipping flag(VI01 EQ imm24)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Clipping transfer

FCGET

31		25	24	23	22	21	20		16	15		11	10				00	
	FCGET		x	y	z	w		It		0				0				
	0011100		0	0	0	0				00000				000000000000				

Format: FCGET It

Purpose:

To move the clipping flag to a VU IR

Description: It <- clipping flag

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Clipping OR

FCOR

31	25	24	23		00	
	FCOR	x		immediate		
	0010011	0				

Format: FCOR VI01,imm24

Purpose:

To perform an OR between the VI01 register and the 24 bit immediate and update the clipping flag

Description: clipping flag(VI01 OR imm24)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Clipping set

FCSET

31	25	24	23		00	
	FCSET	x		immediate		
	0010001	0				

Format: FCSET imm24

Purpose:

To set the clipping flag to the value of the 24 bit immediate

Description: clipping flag = imm24

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

MAC AND

FMAND

31		25	24	23	22	21	20		16	15		11	10				00	
	FMAND		x	y	z	w		It		Is					0			
	0011010		0	0	0	0									000000000000			

Format: FMAND It,Is

Purpose:

To AND the values in two VU IRs and move the result to the MAC flag

Description: MAC <- It AND Is

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

MAC equality

FMEQ

31	25	24	23	22	21	20	16	15	11	10	00
FMEQ	x	y	z	w	It	Is				0	
0011000	0	0	0	0						000000000000	

Format: FMEQ It,Is

Purpose:

To compare the values in two VU IRs and only move the result to the MAC flag on equality

Description: MAC eq(It,Is)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

MAC OR

FMOR

31				25 24 23 22 21 20				16 15				11 10				00
	FMOR		x	y	z	w	It		Is				0			
	0011011		0	0	0	0							000000000000			

Format: FMOR It,Is

Purpose:

To OR the values two VU IRs and move the result to the MAC flag

Description: MAC == It OR Is

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Status AND

FSAND

31	25	24	23	22	21	20	16	15	11	10	00
FSAND	x	y	z	w	It	0	immediate				
0010110	0	0	0	*		00000	12th bit is in dest.w				

Format: FSAND It,imm12

Purpose:

To AND the value in a VU IR with a 12 bit immediate and move the result to the status flag

Description: STATUS == It OR imm12

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Status equality

FSEQ

31	25	24	23	22	21	20	16	15	11	10	00
FSEQ	x	y	z	w	It	0	immediate				
0010100	0	0	0	*		00000	12th bit is in dest.w				

Format: FSEQ It, imm12

Purpose:

To compare the value in a VU IR with a 12 bit immediate and only move the result to the status flag on equality

Description: STATUS eq(It, imm12)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Status OR

FSOR

31	25	24	23	22	21	20	16	15	11	10	00
	FSOR	x	y	z	w	It	0		immediate		
	0010111	0	0	0	*		00000		12th bit is in dest.w		

Format: FSOR It, imm12

Purpose:

To OR the value in a VU IR with a 12 bit immediate and move the result to the status flag

Description: STATUS == It OR imm12

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Status set

FSSET

31	25	24	23	22	21	20	16	15	11	10	00	
	FSSET	x	y	z	w	0		0		immediate		
	0010101	0	0	0	*	00000		00000		12th bit is in dest.w		

Format: FSSET imm12

Purpose:

To set the status flag to the value of the 12bit immediate

Description: STATUS == imm12

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Integer add

IADD

31		25	24	23	22	21	20		16	15		11	10		06	05		00	
	SPECIAL		x	y	z	w		It			Is			Id			IADD		
	1000000		0	0	0	0											110000		

Format: IADD Id, Is, It

Purpose:

To add two VU IRs and store the result in a VU IR

Description: $Id = Is + It$

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Integer add immediate

IADDI

31		25	24	23	22	21	20		16	15		11	10		06	05		00	
	SPECIAL		x	y	z	w		It			Is			imm			IADDI		
	1000000		0	0	0	0											110010		

Format: IADDI It, Is, imm5

Purpose:

To add a VU IRs with a 5 bit immediate and store the result in a VU IR

Description: $It = Is + imm5$

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Integer add immediate unsigned

IADDIU

31	25	24	23	22	21	20	16	15	11	10	00
IADDIU	x	y	z	w	It	Is	immediate				
0001000	*	*	*	*			the upper 4 bits are in dest				

Format: IADDIU It, Is, imm12

Purpose:

To add a VU IRs with an unsigned 12 bit immediate and store the result in a VU IR

Description: $It = Is + imm12$

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Integer AND

IAND

31		25	24	23	22	21	20		16	15		11	10		06	05		00	
	SPECIAL		x	y	z	w		It		Is			Id			IADD			
	1000000		0	0	0	0										110100			

Format: IAND Id, Is, It

Purpose:

To AND two VU IRs and store the result in a VU IR

Description: Id = Is AND It

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Integer branch on equal

IBEQ

31	25	24	23	22	21	20	16	15	11	10	00
IBEQ						It	Is		offset		
0101000						0	0	0	0		

Format: IBEQ It, Is, offset

Purpose:

To branch to a VU instruction memory offset if two VU IRs are equal

Description: $PC \leftarrow PC + \text{offset} + 8$

Restrictions:

None

Operation:

Exceptions:

Programming notes:

The offset refers to doublewords in the VU instruction memory.
This instruction is applicable to both VU0 and VU1

Integer branch on greater or equal to zero

IBGEZ

31	25	24	23	22	21	20	16	15	11	10	00
IBGEZ						0	Is		offset		
0101111						00000					

Format: IBGEZ Is, offset

Purpose:

To branch to a VU instruction memory offset if a VU IR is greater or equal to zero

Description: if Is ≥ 0

PC \leftarrow PC + offset + 8

Restrictions:

None

Operation:

Exceptions:

Programming notes:

The offset refers to doublewords in the VU instruction memory.
This instruction is applicable to both VU0 and VU1

Integer branch on greater than zero

IBGTZ

31	25	24	23	22	21	20	16	15	11	10	00
IBGTZ						0	Is		offset		
0101101						00000					

Format: IBGEZ Is, offset

Purpose:

To branch to a VU instruction memory offset if a VU IR is greater than zero

Description: if Is >0

PC <- PC + offset + 8

Restrictions:

None

Operation:

Exceptions:

Programming notes:

The offset refers to doublewords in the VU instruction memory.
This instruction is applicable to both VU0 and VU1

Integer branch on less or equal to zero

IBLEZ

31	25	24	23	22	21	20	16	15	11	10	00
IBLEZ				x	y	z	w	0	Is		offset
0101110				0	0	0	0	00000			

Format: IBLEZ Is, offset

Purpose:

To branch to a VU instruction memory offset if a VU IR is less or equal to zero

Description: if Is <=0

PC <- PC + offset + 8

Restrictions:

None

Operation:

Exceptions:

Programming notes:

The offset refers to doublewords in the VU instruction memory.
This instruction is applicable to both VU0 and VU1

Integer branch on not equal

IBNE

31	25				24	23	22	21	20	16		15	11		10	00	
IBNE					x	y	z	w	It			Is			offset		
0101001					0	0	0	0									

Format: IBNE It, Is, offset

Purpose:

To branch to a VU instruction memory offset if two VU IR are not equal

Description: if Is =| It

PC <- PC + offset + 8

Restrictions:

None

Operation:

Exceptions:

Programming notes:

The offset refers to doublewords in the VU instruction memory.

This instruction is applicable to both VU0 and VU1

Integer load word

ILW

31	25	24	23	22	21	20	16	15	11	10	00
ILW	x	y	z	w	ft		base		offset		
0000100			dest								

Format: ILW.dest ft, offset(base)

Purpose:

To load up to 4 fields of a VU FPR with a word from the VU data cache

Description: fs.dest <- offset(base)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

The offset refers to doublewords in the VU data memory.
This instruction is applicable to both VU0 and VU1

Integer load word register

ILWR

31	25	24	23	22	21	20	16	15	11	10	06	05	00		
	SPECIAL		x	y	z	w		ft		base		ILWR		SPECIAL	
	1000000				dest							01111		111110	

Format: ILWR.dest ft, base

Purpose:

To load up to 4 fields of a VU FPR with a word from the VU data cache

Description: fs.dest <- [base]

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Integer OR

IOR

31		25 24 23 22 21 20		16 15		11 10		06 05		00
	SPECIAL	x y z w	It		Is		Id		IOR	
	1000000	0 0 0 0							110011	

Format: IOR Id, Is, It

Purpose:

To OR two VU IRs and store the result in a VU IR

Description: Id = Is OR It

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Integer subtract

ISUB

31		25 24 23 22 21 20		16 15		11 10		06 05		00
	SPECIAL	x y z w	It		Is		Id		ISUB	
	1000000	0 0 0 0							110001	

Format: ISUB Id, Is, It

Purpose:

To subtract two VU IRs and store the result in a VU IR

Description: $Id = Is - It$

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Integer subtract immediate unsigned

ISUBIU

31	25	24	23	22	21	20	16	15	11	10	00
ISUBIU				x	y	z	w	It	Is	immediate	
0001001				*	*	*	*			the upper 4 bits are in dest	

Format: IADDIU It, Is, imm12

Purpose:

To subtract an unsigned 12 bit immediate from a VU IR and store the result in a VU IR

Description: $It = Is + imm12$

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Integer store word

ISW

31	25	24	23	22	21	20	16	15	11	10	00
ISW	x	y	z	w	ft		base		offset		
0000101			dest								

Format: ISW.dest ft, offset(base)

Purpose:

To store up to 4 fields of a VU FPR in the VU data cache

Description: offset(base) <- fs.dest

Restrictions:

None

Operation:

Exceptions:

Programming notes:

The offset refers to doublewords in the VU data memory.
This instruction is applicable to both VU0 and VU1

Integer store word register

ISWR

31	25	24	23	22	21	20	16	15	11	10	06	05	00		
	SPECIAL		x	y	z	w		ft		base		ISWR		SPECIAL	
	1000000				dest							01111		111111	

Format: ISWR.dest ft, base

Purpose:

To store up to 4 fields of a VU FPR in the VU data cache

Description: [base] <- fs.dest

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Jump and link register

JALR

31	25 24 23 22 21 20					16 15	11 10					00
	JALR		x	y	z	w	It		Is		0	
	0100101		0	0	0	0					000000000000	

Format: JALR It, Is

Purpose:

To branch to the VU instruction memory offset of a VU IR and link to a VU IR

Description: PC \leftarrow It
Is \leftarrow PC + 8

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Jump register

JR

31	25	24	23	22	21	20	16	15	11	10	00
	JR	x	y	z	w	0		Is		0	
	0100100	0	0	0	0	00000				000000000000	

Format: JALR Is

Purpose:

To branch to the VU instruction memory offset of a VU IR

Description: PC <- Is

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Load quadraword

LQ

31	25	24	23	22	21	20	16	15	11	10	00
LQ	x	y	z	w	ft	base	offset				
00000000	dest										

Format: LQ.dest ft, offset(base)

Purpose:

To load up to 4 fields of a VU FPR with adjacent words in the VU data cache

Description: fs.dest <- offset(base)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

The offset refers to doublewords in the VU data memory.
This instruction is applicable to both VU0 and VU1

Load quadword pre decrement

LQD

31	25 24 23 22 21 20					16 15	11 10		06 05	00				
	SPECIAL		x	y	z	w	ft		base		LQD		SPECIAL	
	1000000		dest								01101		111110	

Format: LQD.dest ft, --base

Purpose:

To load up to 4 fields of a VU FPR with VU data pre decrement

Description: base == base - 8
ft.dest <- [base]

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Load quadword post increment

LQI

31	25 24 23 22 21 20					16 15	11 10		06 05	00				
	SPECIAL		x	y	z	w	ft		base		LQI		SPECIAL	
	1000000		dest								01101		111100	

Format: LQI.dest ft, base++

Purpose:

To load up to 4 fields of a VU FPR with VU data post increment

Description: ft.dest <- [base]
base == base + 8

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Move from integer register

MFIR

31	25 24 23 22 21 20					16 15	11 10		06 05		00			
	SPECIAL		x	y	z	w	ft		Is		MFIR		SPECIAL	
	1000000		dest								01111		111101	

Format: MFIR.dest ft, Is

Purpose:

To load up to 4 fields of a VU FPR with the value of an IR

Description: ft.dest <- Is

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Move floating point registers

MOVE

31	25	24	23	22	21	20	16	15	11	10	06	05	00		
	SPECIAL		x	y	z	w		ft		fs		MOVE		SPECIAL	
	1000000				dest							01100		111100	

Format: MOVE.dest ft, fs

Purpose:

To move up to 4 fields of a VU FPR to the corresponding fields of a VU FPR

Description: if dest & .x == true : ft.x <- fs.x
if dest & .y == true : ft.y <- fs.y
if dest & .z == true : ft.z <- fs.z
if dest & .w == true : ft.w <- fs.w

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Move and rotate per word

MR32

31	25 24 23 22 21 20					16 15	11 10			06 05	00			
	SPECIAL		x	y	z	w	ft		fs		MR32		SPECIAL	
	1000000		dest								01100		111101	

Format: MOVE.dest ft, fs

Purpose:

To move up to 4 fields of a VU FPR to rotated fields of a VU FPR

Description: if dest & .x == true : fs.x -> ft.y
if dest & .y == true : fs.y -> ft.z
if dest & .z == true : fs.z -> fs.w
if dest & .w == true : fs.w -> fs.x

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Move to integer register

MTIR

31		25 24 23 22 21 20		16 15		11 10		06 05		00
	SPECIAL	x y z w	It		fs		MTIR		SPECIAL	
	1000000	0 0 fs.e					01111		111100	

Format: MTIR.e It, fs.e

Purpose:

To move a field of a VU FPR to an IR

Description: It <- fs.dest

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

R move

RGET

31	25 24 23 22 21 20	16 15	11 10	06 05	00
SPECIAL	x y z w ft	0	RGET	SPECIAL	
1000000	dest	00000	10000	111101	

Format: RGET.dest ft, R

Purpose:

To load up to 4 fields of a VU FPR with the R register

Description: fs.dest <- R

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

R initialise

RINIT

31	25	24	23	22	21	20	16	15	11	10	06	05	00
SPECIAL	x	y	z	w	0	fs	RINIT	SPECIAL					
1000000	0	0	fs.e	00000			10000	111110					

Format: RINIT.e R, fs.e

Purpose:

To initialise the random generator using a field of a VU FPR as matrix

Description: R <- Random(fs.e)

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

R next

RNEXT

31		25 24 23 22 21 20		16 15		11 10		06 05		00
	SPECIAL	x y z w	ft		0		RNEXT		SPECIAL	
	1000000	dest			00000		10000		111100	

Format: RNEXT.dest ft, R

Purpose:

To load up to 4 fields of a VU FPR with the next random number from the R register

Description: ft.dest <- R

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Reverse square root

RSQRT

31	25	24	23	22	21	20	16	15	11	10	06	05	00		
SPECIAL							x	y	z	w	ft		fs	RSQRT	SPECIAL
1000000							ft.e		fs.e				01110	11110	

Format: RSQRT Q, ft.e, fs.e

Purpose:

To calculate $ft.e/\sqrt{fs.e}$

Description: $Q \leftarrow ft.e/\sqrt{fs.e}$

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

R XOR

RXOR

31	25	24	23	22	21	20	16	15	11	10	06	05	00		
	SPECIAL		x	y	z	w		0		fs		RXOR		SPECIAL	
	1000000		0	0	fs.e		00000					10000		111111	

Format: RXOR.e R, fs.e

Purpose:

To XOR the value in the R register and a field of a VU FPR and store the result in the R register

Description: R <- R XOR fs.e

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Store quadword

SQ

31	25	24	23	22	21	20	16	15	11	10	00	
	SQ	x	y	z	w		ft		base		offset	
	0000001			dest								

Format: SQ.dest ft, offset(base)

Purpose:

To store up to 4 fields of a VU FPR to adjacent words in the VU data cache

Description: offset(base) <- fs.dest

Restrictions:

None

Operation:

Exceptions:

Programming notes:

The offset refers to doublewords in the VU data memory.
This instruction is applicable to both VU0 and VU1

Store quadword pre decrement

SQD

31	25 24 23 22 21 20					16 15	11 10		06 05		00								
	SPECIAL			x	y z	w	base			ft			SQD			SPECIAL			
	1000000			dest										01101			111111		

Format: SQD.dest ft, --base

Purpose:

To store up to 4 fields of a VU FPR to the VU data cache pre decrement

Description: base == base - 8
[base] <- ft.dest

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Store quadword post increment

SQI

31	25 24 23 22 21 20					16 15	11 10			06 05	00			
	SPECIAL		x	y	z	w	base		ft		SQI		SPECIAL	
	1000000		dest								01101		111101	

Format: SQI.dest ft, base++

Purpose:

To store up to 4 fields of a VU FPR to the VU data cache post increment

Description: [base] <- ft.dest
base == base + 8

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

SQRT

This instruction is applicable to both VU0 and VU1

Wait P operation

WAITP

31	25 24 23 22 21 20					16 15	11 10	06 05	00					
	SPECIAL		x	y	z	w	0		0		WAITP		SPECIAL	
	1000000		0	0	0	0	00000		00000		11110		111111	

Format: WAITP

Purpose:

To postpone VU operation until the operation of the instruction that writes to the P register is concluded

Description:

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Wait Q operation

WAITQ

31	25 24 23 22 21 20					16 15	11 10	06 05	00					
	SPECIAL		x	y	z	w	0		0		WAITQ		SPECIAL	
	1000000		0	0	0	0	00000		00000		01110		11111	

Format: WAITQ

Purpose:

To postpone VU operation untill the operation of the instruction that writes to the Q register is concluded

Description:

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is applicable to both VU0 and VU1

Initialise GIF path

XGKICK

31		25	24	23	22	21	20		16	15		11	10		06	05		00	
	SPECIAL		x	y	z	w		0			Is			XGKICK			SPECIAL		
	1000000		0	0	0	0		00000						11011			111100		

Format: XGKICK Is

Purpose:

To initialise the VU1 path to the GIF

Description:

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is only applicable to VU1

GIF path handling

XITOP

31		25	24	23	22	21	20		16	15		11	10		06	05		00	
	SPECIAL		x	y	z	w		It		0			XITOP			SPECIAL			
	1000000		0	0	0	0				00000			11010			111101			

Format: XITOP It

Purpose:

To control the VU1 path to the GIF

Description:

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is only applicable to VU1

GIF path handling

XTOP

31		25	24	23	22	21	20		16	15		11	10		06	05		00	
	SPECIAL		x	y	z	w		It		0			XTOP			SPECIAL			
	1000000		0	0	0	0				00000			11010			111100			

Format: XTOP It

Purpose:

To control the VU1 path to the GIF

Description:

Restrictions:

None

Operation:

Exceptions:

Programming notes:

This instruction is only applicable to VU1

LEGEND

VU	Vector Unit
VU0	Vector Unit 0
VU1	Vector Unit 1
FPR	Floating point register
IR	Integer register
dest	destination fields
fs	floating point source register
ft	floating point target register
fd	floating point destination register
Is	integer source register
It	integer target register
Id	integer destination register
imm	immediate
bc	broadcast field
.e	field selector