**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 its 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 programs 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**

<table>
<thead>
<tr>
<th>Word 0</th>
<th>Word 1</th>
<th>Word 2</th>
<th>Word 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>0</td>
<td>63</td>
<td>32</td>
</tr>
<tr>
<td>x</td>
<td>y</td>
<td>z</td>
<td>w</td>
</tr>
</tbody>
</table>

**VU0F00 AND VU1F00 REGISTERS**

<table>
<thead>
<tr>
<th>Word 0</th>
<th>Word 1</th>
<th>Word 2</th>
<th>Word 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>0</td>
<td>63</td>
<td>32</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>
The Floating registers can be represented in 4 fixed point formats:

<table>
<thead>
<tr>
<th>Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed 0</td>
<td>fixed point 0 bits</td>
</tr>
<tr>
<td>Fixed 4</td>
<td>fixed point 4 bits</td>
</tr>
<tr>
<td>Fixed 12</td>
<td>fixed point 12 bits</td>
</tr>
<tr>
<td>Fixed 15</td>
<td>fixed point 15 bits</td>
</tr>
</tbody>
</table>

**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
- ADDAII Upper
- SUBAII Upper
- MULAI Upper
- MADDAI Upper
- MSUBAII 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 let's 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:

|   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 let's look at some documentation.

| 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, let's 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 saids, is that we can combine different dests

```
addy.xyzw $vf3xywz, $vf4xywz, $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

```
adz.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 manaul 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`
<table>
<thead>
<tr>
<th>Instruction</th>
<th>VU0 coprocessor</th>
<th>VU0 microprocessor</th>
<th>VU1 coprocessor</th>
<th>VU1 microprocessor</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDA.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>I.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Q.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>e.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SUBA.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>I.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Q.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>e.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MUL.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>I.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Q.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>e.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MUL.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>I.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MADD.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>I.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Q.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>e.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MADD.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>I.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Q.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>e.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>NOP</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Instruction</td>
<td>VU0 coprocessor</td>
<td>VU0 microprocessor</td>
<td>VU1</td>
<td>Instruction</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------</td>
<td>--------------------</td>
<td>-----</td>
<td>-------------</td>
</tr>
<tr>
<td>IADD</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>FCEQ</td>
</tr>
<tr>
<td>IADDI</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>FCGET</td>
</tr>
<tr>
<td>IADDIU</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>FCSET</td>
</tr>
<tr>
<td>ISUB</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>FSAND</td>
</tr>
<tr>
<td>ISUBIU</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>FSOR</td>
</tr>
<tr>
<td>DIV</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>FSEQ</td>
</tr>
<tr>
<td>SQRT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>FSSET</td>
</tr>
<tr>
<td>RQSRT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>FMAND</td>
</tr>
<tr>
<td>IAND</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>FMOR</td>
</tr>
<tr>
<td>IOR</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>FMEQ</td>
</tr>
<tr>
<td>B</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>RINIT</td>
</tr>
<tr>
<td>BAL</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>RGET.dest</td>
</tr>
<tr>
<td>IBEQ</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>RNEXT.dest</td>
</tr>
<tr>
<td>IBNE</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>RXOR</td>
</tr>
<tr>
<td>IBGTZ</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>WAITP</td>
</tr>
<tr>
<td>IBLEZ</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>WAITQ</td>
</tr>
<tr>
<td>IBGEZ</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>ESUM</td>
</tr>
<tr>
<td>JR</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>ESQRT</td>
</tr>
<tr>
<td>JALR</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>ESADD</td>
</tr>
<tr>
<td>ILW.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>ERSQRT</td>
</tr>
<tr>
<td>ILWR.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>ERSADD</td>
</tr>
<tr>
<td>ISW.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>EATAN</td>
</tr>
<tr>
<td>ISWR.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>EATANx</td>
</tr>
<tr>
<td>LQ.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>EATANxy</td>
</tr>
<tr>
<td>LQI.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>ELENG</td>
</tr>
<tr>
<td>LQD.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>ESIN</td>
</tr>
<tr>
<td>SQ.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>ERLENG</td>
</tr>
<tr>
<td>SQI.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>ERCPR</td>
</tr>
<tr>
<td>SQD.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>EEXP</td>
</tr>
<tr>
<td>MOVE.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>MFP.dest</td>
</tr>
<tr>
<td>MFIR.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>XTOP</td>
</tr>
<tr>
<td>MTIR</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>XITOP</td>
</tr>
<tr>
<td>MR32.dest</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>XGKICK</td>
</tr>
<tr>
<td>FCAND</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>FCOR</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
UPPER VU1 INSTRUCTION ENCODING

UPPER VU1 INSTRUCTION LAYOUT

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

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

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     |
|       | 0| 0|    dest   |        |     |      |     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

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>11</th>
<th>10</th>
<th>06</th>
<th>05</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>E</td>
<td>M</td>
<td>D</td>
<td>T</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>w</td>
<td>0</td>
<td>fs</td>
<td>fd</td>
<td>ADDI</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>dest</td>
<td>00000</td>
<td></td>
<td></td>
<td></td>
<td>100010</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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

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

|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

|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

| 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

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

| 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:  
ADD Ax. dest ACC, fs, ft
ADD Ay. dest ACC, fs, ft
ADD Az. dest ACC, fs, ft
ADD Aw. 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

| 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 | CLIPw | SPECIAL | bc |
| 0| 0| dest |

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

```
| 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 | PTOI | 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

<table>
<thead>
<tr>
<th>Broadcast field</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0 decimal digits</td>
</tr>
<tr>
<td>01</td>
<td>4 decimal digits</td>
</tr>
<tr>
<td>10</td>
<td>12 decimal digits</td>
</tr>
<tr>
<td>11</td>
<td>15 decimal digits</td>
</tr>
</tbody>
</table>
Integer to floating point

| 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 |  
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

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.

<table>
<thead>
<tr>
<th>Broadcast field</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0 decimal digits</td>
</tr>
<tr>
<td>01</td>
<td>4 decimal digits</td>
</tr>
<tr>
<td>10</td>
<td>12 decimal digits</td>
</tr>
<tr>
<td>11</td>
<td>15 decimal digits</td>
</tr>
</tbody>
</table>
Multiply and add

MADD

<table>
<thead>
<tr>
<th>31 30 29 28 27 26 25 24 23 22 21 20</th>
<th>16 15</th>
<th>11 10</th>
<th>06 05</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>E</td>
<td>M</td>
<td>D</td>
<td>T</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>dest</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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

| 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     |      |      | MADDI |
|    |    |    |    |    |    |    |    |    |    |    |       |      | 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**

| I | E | M | D | T | 0 | 0 | x | y | z | w |       0      |      fs      |      fd      | MADDQ     |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | dest | 00000 | | | | | | | | |

**MADDQ**

**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)

**Purpose:**

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

**Exceptions:**

None

**Operation:**

**Restrictions:**

None

**Programming notes:**

This instruction is applicable to both VU0 and VU1
Multiply and add broadcast

| I | E | M | D | T | 0 | 0 | x | y | z | w |      ft      |      fs      |      fd      | MADDbc | bc |
|   |   |   |   |   |   |   |   |   |   |   |      0010    |             |             |        |    |

| dest |
|      |

**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

<table>
<thead>
<tr>
<th>I</th>
<th>E</th>
<th>M</th>
<th>D</th>
<th>T</th>
<th>0</th>
<th>0</th>
<th>x</th>
<th>y</th>
<th>z</th>
<th>w</th>
<th>ft</th>
<th>fs</th>
<th>MADDA</th>
<th>SPECIAL</th>
<th>bc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>dest</td>
<td></td>
<td></td>
<td></td>
<td>01010</td>
<td>1111</td>
<td>01</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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  

| 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

| 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 MADDAx MADDAy MADDAz MADDAw

<table>
<thead>
<tr>
<th>I</th>
<th>E</th>
<th>M</th>
<th>D</th>
<th>T</th>
<th>0</th>
<th>0</th>
<th>x</th>
<th>y</th>
<th>z</th>
<th>w</th>
<th>ft</th>
<th>fs</th>
<th>MADDAbc</th>
<th>SPECIAL</th>
<th>bc</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>16</td>
<td>15</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>

Format: MADDAx.dest ACC,fs,ft
MADDAy.dest ACC,fs,ft
MADDAz.dest ACC,fs,ft
MADDAw.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

<table>
<thead>
<tr>
<th>I</th>
<th>E</th>
<th>M</th>
<th>D</th>
<th>T</th>
<th>0</th>
<th>0</th>
<th>x</th>
<th>y</th>
<th>z</th>
<th>w</th>
<th>ft</th>
<th>fs</th>
<th>fd</th>
<th>MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>dest</td>
<td></td>
<td></td>
<td></td>
<td>101011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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

|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|fd|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

|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|

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

<table>
<thead>
<tr>
<th>I</th>
<th>E</th>
<th>M</th>
<th>D</th>
<th>T</th>
<th>0</th>
<th>0</th>
<th>x</th>
<th>y</th>
<th>z</th>
<th>w</th>
<th>ft</th>
<th>fs</th>
<th>fd</th>
<th>MINI</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>dest</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>101111</td>
</tr>
</tbody>
</table>

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    |     |      |     01111    |

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

| I | E | M | D | T | 0 | 0 | x | y | z | w |      ft      |      fs      |      fd      |  MINIbc | bc  |
|---|---|---|---|---|---|---|---|---|---|---|      |      |      |       |     |
|  | 0 | 0 | dest |      |     |      |   0101  |     |      |      |      |      |      | 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 substract                MSUB

| I| E| M| D| T| 0| 0| x| y| z| w|      ft      |      fs      |      fd      |  MSUB     |
| | 0| 0| 0| dest |       |      |     101101    |

Format:  MSUB.dest fd,fs,ft

Purpose:  
To multiply up to 4 matching fields and substract 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 substract I

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>11</th>
<th>10</th>
<th>06</th>
<th>05</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>E</td>
<td>M</td>
<td>D</td>
<td>T</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>w</td>
<td></td>
<td>0</td>
<td>fs</td>
<td>fd</td>
<td>MSUBI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td>dest</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100111</td>
<td></td>
<td></td>
<td>100111</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Format: MSUBI.dest fd,fs,ft

Purpose:
To multiply up to 4 fields with the I register and substract 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 substract  Q

|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|fd|MSUBQ|100101|

**Format:** MSUBQ.dest fd,fs,ft

**Purpose:**
To multiply up to 4 fields with the Q register and substract 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

|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| 0| 0|      |     |     |      |   0011  |     |

Format:
- MSUBx.dest fd,fs,ft
- MSUBy.dest fd,fs,ft
- MSUBz.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

|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|

Format: MSUBA.dest fd,fs,ft

Purpose:
To multiply up to 4 fields and substract 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  

| 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 |
|       | 0| 0|    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

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

```
| 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 |      |      | 01110 |
```

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

<table>
<thead>
<tr>
<th>I</th>
<th>E</th>
<th>M</th>
<th>D</th>
<th>T</th>
<th>0</th>
<th>0</th>
<th>x</th>
<th>y</th>
<th>z</th>
<th>w</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>MULQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>dest</td>
<td>00000</td>
<td></td>
<td></td>
<td></td>
<td>011100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Format:** \( \text{MULQ}.\text{dest} \text{ fd,fs,ft} \)

**Purpose:**
To multiply up to 4 fields with the Q register

**Description:**
if dest & .x == true : \( \text{fd.x} \leftarrow \text{fs.x} \times Q \)
if dest & .y == true : \( \text{fd.y} \leftarrow \text{fs.y} \times Q \)
if dest & .z == true : \( \text{fd.z} \leftarrow \text{fs.z} \times Q \)
if dest & .w == true : \( \text{fd.w} \leftarrow \text{fs.w} \times Q \)

**Restrictions:**
None

**Operation:**

**Exceptions:**

**Programming notes:**
This instruction is applicable to both VU0 and VU1
## Multiply broadcast

### 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

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

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

| 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 |
| 0| 0| 1| 1| 1| 0|     |     |      |     011011 |

Format: OPMSUB.xyz fd,fs,ft

Purpose: To calculate the outer product post decrement

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

Restrictions: None

Operation:

Exceptions:

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

Format: SUB.dest fd,fs,ft

Purpose:
To substract 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

| 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| 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
### Substract Q

![Format](image)

**Purpose:**
To substract 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
Substract 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 |
| | | |

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

Purpose:  
To substract 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
Substract accumulator

| 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  |
| | 0| 0| dest  | |     |    01011   |   1111  | 00  |

Format:  SUBA.dest ACC,fs,ft

Purpose:
To substract 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**  

<table>
<thead>
<tr>
<th>Format:</th>
<th>SUBAI.dest ACC,fs,I</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Purpose:</strong></td>
<td>To subtract the I register from up to 4 matching fields to the accumulator</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>if dest &amp; .x == true : ACC.x &lt;- fs.x - I</td>
</tr>
<tr>
<td></td>
<td>if dest &amp; .y == true : ACC.y &lt;- fs.y - I</td>
</tr>
<tr>
<td></td>
<td>if dest &amp; .z == true : ACC.z &lt;- fs.z - I</td>
</tr>
<tr>
<td></td>
<td>if dest &amp; .w == true : ACC.w &lt;- fs.w - I</td>
</tr>
<tr>
<td><strong>Restrictions:</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Operation:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Exceptions:</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Programming notes:</strong></td>
<td>This instruction is applicable to both VU0 and VU1</td>
</tr>
</tbody>
</table>
Subtract accumulator Q

**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
Substract accumulator broadcast

SUBAx dest ACC,fs,ft
SUBAy dest ACC,fs,ft
SUBAz dest ACC,fs,ft
SUBAw dest ACC,ft,fs

Purpose:
To substract 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 | rt | rs | rd | SPECIAL | bc |

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

```
| 31 | 25|24|23|22|21|20 | 15|11|10          | 00 |
|-----|---|---|---|---|---|----|----|----|----------------|
| B   | x | y | z | w | 0 | 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

### Format

<table>
<thead>
<tr>
<th>31</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>11</th>
<th>10</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BAL</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>w</td>
<td></td>
<td>It</td>
<td></td>
<td>0</td>
<td></td>
<td>offset</td>
</tr>
<tr>
<td>0100001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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

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 <- 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 |
| 1       | 0   | 0   | 0   | 0   | 0   | 0   |
| 1 | SPECIAL | x| y| z| w|      0       |      fs      |    EATAN   | SPECIAL | bc  |
| 1000000 | 0| 0| s.e | 00000 | 0| 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**: None

**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 |    |
| 0000000 | 1 | 1 | 0 | 0 | 00000 | 000000 | 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

<table>
<thead>
<tr>
<th>31</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>11</th>
<th>10</th>
<th>06</th>
<th>05</th>
<th>02</th>
<th>01</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SPECIAL</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>w</td>
<td>0</td>
<td></td>
<td>EATAN</td>
<td>SPECIAL</td>
<td>bc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000000</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>00000</td>
<td></td>
<td>11101</td>
<td>1111</td>
<td>01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>31</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>11</th>
<th>10</th>
<th>06</th>
<th>05</th>
<th>02</th>
<th>01</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPECIAL</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>w</td>
<td>0</td>
<td>fs</td>
<td>EEXP</td>
<td>SPECIAL</td>
<td>bc</td>
<td>1000000</td>
<td>0</td>
<td>0</td>
<td>s.e</td>
<td>00000</td>
<td></td>
</tr>
</tbody>
</table>

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: $P \leftarrow \exp(fs.e)$

Restrictions:
None

Operation:

Exceptions:

Programming notes:
This instruction is applicable only to VU1
Length

<table>
<thead>
<tr>
<th>31</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>11</th>
<th>10</th>
<th>06</th>
<th>05</th>
<th>02</th>
<th>01</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPECIAL</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>w</td>
<td>0</td>
<td>fs</td>
<td>ELENG</td>
<td>SPECIAL</td>
<td>bc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000000</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>00000</td>
<td></td>
<td>11100</td>
<td>1111</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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

| 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

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 <- rev.sq.root(fs.x + fs.y + fs.z)

Restrictions:
None

Operation:

Exceptions:

Programming notes:
This instruction is applicable only to VU1
Square root

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

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 <- sq.root(fs.x + fs.y + fs.z)

Restrictions:
None

Operation:

Exceptions:

Programming notes:
This instruction is applicable only to VU1
| 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**

|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 |
|1 | 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
### Format

```
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:

```
Description:  P <- fs.x + fs.y + fs.z + fs.w
```

### Restrictions:

None

### Operation:

- None

### Exceptions:

- None

### Programming notes:

This instruction is applicable only to VU1
**Clipping AND**

<table>
<thead>
<tr>
<th>31</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCAND</td>
<td>x</td>
<td>immediate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010010</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>31</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCEQ</td>
<td>x</td>
<td>immediate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010000</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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

**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

| 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

<table>
<thead>
<tr>
<th>31</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCSET</td>
<td>x</td>
<td>immediate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010001</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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
**FMAND**

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

| 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| 0 | Is |     | 00000000000 | |

**Format:** FMEQ It,Is

**Purpose:**
To compare the values in two VU IRs and only move the result to the MAC flag on eqaulity

**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|0|
|FMOR | x| y| z| w| It | Is | 0 | 0 |
|0011011 | 0| 0| 0| 0| 0 | 0 | 00000000000 |

**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 | 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 12th bit is in dest.w |
| FSAND | x| y| z| w| It | O | immediate |
| 0010110 | 0| 0| 0| *| 00000 |

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

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

<table>
<thead>
<tr>
<th>31</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>11</th>
<th>10</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSOR</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>w</td>
<td>It</td>
<td>0</td>
<td>immediate</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0010111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>*</td>
<td>00000</td>
<td>12th bit is in dest.w</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>00</th>
<th>12th bit is in dest.w</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,0</td>
<td>00000</td>
</tr>
<tr>
<td>0010101</td>
<td>0</td>
</tr>
</tbody>
</table>

**Format**: FSSET imm12

**Purpose**: To set the status flag to the value of the 12bit immediate

**Description**: STATUS == imm12

**Restrictions**: None

**Operation**: None

**Exceptions**: None

**Programming notes**: This instruction is applicable to both VU0 and VU1
## Integer add

| 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**  

<table>
<thead>
<tr>
<th>31</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>11</th>
<th>10</th>
<th>06</th>
<th>05</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPECIAL</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>w</td>
<td>It</td>
<td>Is</td>
<td>imm</td>
<td>IADDI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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

| 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**

| 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

<table>
<thead>
<tr>
<th>31</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>11</th>
<th>10</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBEQ</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>w</td>
<td>It</td>
<td>Is</td>
<td>offset</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0101000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Format: IBEQ It, Is, offset

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

Description: 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 greater or equal to zero

IBGEZ

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 <- 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  

<table>
<thead>
<tr>
<th>31</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>11</th>
<th>10</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBGTZ</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>w</td>
<td>0</td>
<td>Is</td>
<td>offset</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0101101</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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

|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 | It, Is, offset |
| 0101001 | 0 | 0 | 0 | 0 | 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  

```
| 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

**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**

```
| 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 substract

Format: ISUB Id, Is, It

Purpose:
To substract 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

| 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**

<table>
<thead>
<tr>
<th>31</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>11</th>
<th>10</th>
<th>06</th>
<th>05</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPECIAL</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>w</td>
<td>ft</td>
<td>base</td>
<td>ISWR</td>
<td>SPECIAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000000</td>
<td>dest</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>01111</td>
<td>11111</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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  

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 <- It
             Is <- PC + 8

Restrictions:
None

Operation:

Exceptions:

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

| 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| 0000 |     | 00000000000 |

**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

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 quadraword pre decrement  

| 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 quadraword post increment  

<table>
<thead>
<tr>
<th>31    25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>11</th>
<th>10</th>
<th>06</th>
<th>05</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPECIAL</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>w</td>
<td>ft</td>
<td>base</td>
<td>LQI</td>
<td>SPECIAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000000</td>
<td>dest</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>01101</td>
<td>11100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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

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

```
| 31 | 25|24|23|22|21|20 | 16|15 | 11|10 | 06|05 | 00 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
|   | SPECIAL | x| y| z| w|     ft      |  fs      |    MOVE | SPECIAL |   |
|   | 10000000 | 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

<table>
<thead>
<tr>
<th>31</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>11</th>
<th>10</th>
<th>06</th>
<th>05</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPECIAL</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>w</td>
<td>ft</td>
<td>fs</td>
<td></td>
<td>MR32</td>
<td>SPECIAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000000</td>
<td>dest</td>
<td></td>
<td></td>
<td>01100</td>
<td>111101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

| 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

|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**

<table>
<thead>
<tr>
<th>00</th>
<th>06</th>
<th>10</th>
<th>11</th>
<th>15</th>
<th>16</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPECIAL</td>
<td>w</td>
<td>z</td>
<td>y</td>
<td>x</td>
<td>ft</td>
<td>0</td>
<td>RNEXT</td>
<td>SPECIAL</td>
<td>1000000</td>
<td>dest</td>
<td>00000</td>
<td>111100</td>
</tr>
</tbody>
</table>

**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

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

Purpose:
To calculate ft.e/sqrt(fs.e)

Description: Q <- ft.e/sqrt(fs.e)

Restrictions:
None

Operation:

Exceptions:

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

| 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 quadraword

\[
\begin{array}{c|cccc|c|c|c|c}
| 31 | 25|24|23|22|21|20 | 16|15 | 11|10 |
\hline
| SQ | x| y| z| w| ft | base | offset |
\hline
| 000001 | dest | |
\end{array}
\]

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 quadraword pre decrement

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 quadraword post increment

| 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
Square root

Format: SQRT Q, ft.e

Purpose:
The square root of the value in a single field of a VU FPR

Description: Q <- sqrt(ft.e)

Restrictions:
None

Operation:

Exceptions:

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

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

Format: WAITQ

Purpose:
To postpone VU operation until 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

| 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 | 0000 | | 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

<table>
<thead>
<tr>
<th>31</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>16</th>
<th>15</th>
<th>11</th>
<th>10</th>
<th>06</th>
<th>05</th>
<th>00</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SPECIAL</td>
<td>x</td>
<td>y</td>
<td>z</td>
<td>w</td>
<td>It</td>
<td>0</td>
<td>XITOP</td>
<td>SPECIAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>00000</td>
<td>11010</td>
<td>111101</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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

| 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 | 1010 | 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