VU User's Manual

Publication date: April 2002

Sony Computer Entertainment Inc.
1-1, Akasaka 7-chome, Minato-ku
Tokyo 107-0052 Japan

Sony Computer Entertainment America
919 East Hillsdale Blvd.
Foster City, CA 94404, U.S.A.

Sony Computer Entertainment Europe
30 Golden Square
London W1F 9LD, U.K.

About This Manual

The "VU User's Manual" describes the operational functions of the VPU (Vector Operation Unit) built into the Emotion Engine. For information on data and microprogram transfers to the VPU, refer to the "EE User's Manual".

- Chapter 1 "VU Overview" describes the configurations of the VPU and the VU (the core of the VPU), the differences between the two VPUs (VPU0/VPU1) that are embedded, and the operation modes.
- Chapter 2 "Data/Calculation Basic Specifications" describes the numerical data formats used by the VU, rounding-off operations in calculations, and exception specifications. Note that the description does not fully conform to the requirements of the IEEE 754 standard.
- Chapter 3 "Micro Mode" describes the architecture and operation of micro mode, in which the VU operates as a stand-alone processor.
- Chapter 4 "Micro Mode Instruction Reference" describes the individual micro mode instructions.
- Chapter 5 "Macro Mode" describes the architecture and operation of macro mode, in which VPU0 operates as a coprocessor of the EE Core. This chapter also explains how to control VPU1 from the EE Core.
- Chapter 6 "Macro Mode Instruction Reference" describes the individual macro mode instructions.
- Chapter 7 "Appendix" gives other information, including sample micro programs.

Changes Since Release of 5th Edition

Since release of the 5th Edition of the VU User's Manual, the following changes have been made.
Note that each of these changes is indicated by a revision bar in the margin of the affected page.

Ch. 1: VU Overview

- Information about the derivation of the M series polynomial was added to "RANDU" in section 1.1.2. Lower Execution Unit, on page 17.

Ch. 3: Micro Mode

- A correction was made to the description following the figure in Section 3.4.9. XGKICK Pipeline, on page 52.

Ch. 4: Micro Mode Instruction Reference

- A correction was made to the "Operation Code" figure in the EATANxz reference on page 134.
- A correction was made to "Example" in the IBGTZ reference on page 164.
- A correction was made to "Example" in the IBLEZ reference on page 165.
- A correction was made to "Example" in the IBLTZ reference on page 166.
- A correction was made to "Mnemonic" in the ISW reference on page 173.
- Information about the XGKIXK pipeline and XGKICK synchronization was added to "Remarks" in the XGKICK reference on page 196.
- Information was added to "Operation" in the XITOP reference on page 197.

Ch. 6: Macro Mode Instruction Reference

- A correction was made to "Operation" in the QMTC2 reference on page 238.

(This page is left blank intentionally)

Glossary

Term	Definition
EE	Emotion Engine. CPU of the PlayStation 2.
EE Core	Generalized computation and control unit of EE. Core of the CPU.
COP0	EE Core system control coprocessor.
COP1	EE Core floating-point operation coprocessor. Also referred to as FPU.
COP2	Vector operation unit coupled as a coprocessor of EE Core. VPU0.
GS	Graphics Synthesizer. Graphics processor connected to EE.
GIF	EE Interface unit to GS.
IOP	Processor connected to EE for controlling input/output devices.
SBUS	Bus connecting EE to IOP.
VPU (VPU0/VPU1)	Vector operation unit. EE contains 2 VPUs: VPU0 and VPU1.
VU (VU0/VU1)	VPU core operation unit.
VIF (VIF0/VIF1)	VPU data decompression unit.
VIFcode	Instruction code for VIF.
SPR	Quick-access data memory built into EE Core (Scratchpad memory).
IPU	EE Image processor unit.
word	Unit of data length: 32 bits
qword	Unit of data length: 128 bits
Slice	Physical unit of DMA transfer: 8 qwords or less
Packet	Data to be handled as a logical unit for transfer processing.
Transfer list	A group of packets transferred in serial DMA transfer processing.
Tag	Additional data indicating data size and other attributes of packets.
DMAtag	Tag positioned first in DMA packet to indicate address/size of data and address of the following packet.
GS primitive	Data to indicate image elements such as point and triangle.
Context	A set of drawing information (e.g. texture, distant fog color, and dither matrix) applied to two or more primitives uniformly. Also referred to as the drawing environment.
GIFtag	Additional data to indicate attributes of GS primitives.
Display list	A group of GS primitives to indicate batches of images.

(This page is left blank intentionally)

Contents

1. VU Overview 15
 1.1. VPU Structure 16
  1.1.1. Upper Execution Unit 16
  1.1.2. Lower Execution Unit 17
  1.1.3. Floating-Point Registers 18
  1.1.4. Integer Register 18
  1.1.5. VU Mem 18
  1.1.6. Micro Mem 19
 1.2. VU Execution Mode 20
 1.3. VU Operation Status 21
  1.3.1. Ready State 21
  1.3.2. Run State 21
  1.3.3. Stop State 21
 1.4. VU Usage 23
  1.4.1. VU1 Usage Outline 23
  1.4.2. VU0 Usage Outline 23
2. Data/Calculation Basic Specifications 25
 2.1. Data Format 26
  2.1.1. Floating-Point Values 26
  2.1.2. Fixed-Point Values 26
 2.2. Rounding Off Floating-Point Values 27
 2.3. Exception Processing 28
 2.4. Differences from IEEE 754 29
3. Micro Mode 31
 3.1. Micro Mode Register Set 32
  3.1.1. Floating-Point Registers 32
  3.1.2. Integer Registers 32
  3.1.3. ACC Registers 33
  3.1.4. I Register 33
  3.1.5. Q Register 34
  3.1.6. R Register 34
  3.1.7. P Register 34
 3.2. Micro Instruction Set Overview 35
  3.2.1. Upper Instructions 35
  3.2.2. Lower Instructions 37
 3.3. Flags 39
  3.3.1. MAC Flags 39
  3.3.2. Status Flags (SF) 39
  3.3.3. Clipping Flags (CF) 40
  3.3.4. Flag Set Instructions 40

3.3.5. Flag Changes for Each Instruction......41
3.3.6. Flag Changes for Exception Occurrences......41
3.4. Pipeline Operation......44
3.4.1. Hazards......44
3.4.2. Upper Instruction and Lower Instruction......44
3.4.3. Priority for Writing to a Register......44
3.4.4. FMAC Pipeline......45
3.4.5. FDIV Pipeline......46
3.4.6. EFU Pipeline......48
3.4.7. IALU Pipeline......49
3.4.8. Conditional Branching and Pipeline......50
3.4.9. XGKICK Pipeline......52
3.5. Micro Subroutine Execution......53
3.5.1. How to Execute a Micro Subroutine......53
3.5.2. How to Terminate a Micro Subroutine......53
3.5.3. Operation of Execution and Termination......53
3.6. Other Functions......55
3.6.1. Data Transfer with VU Mem/Micro Mem......55
3.6.2. Debug Support Function......55
4. Micro Mode Instruction Reference......57
4.1. Micro Mode Instruction Set......58
4.1.1. Types of Upper Instruction......58
4.1.2. Types of Lower Instructions......59
4.1.3. Operation Fields for Micro Instructions......61
4.2. Upper Instruction Reference......65
ABS : Absolute Value......66
ADD : Add......67
ADDi : Add to I Register......68
ADDq : Add to Q Register......69
ADDbc : Broadcast Add......70
ADDA : Add; to Accumulator......71
ADDAi : Add I Register; to Accumulator......72
ADDAq : Add Q Register; to Accumulator......73
ADDAbc : Broadcast Add; to Accumulator......74
CLIP : Clipping Judgment......75
FTOI0 : Convert to Fixed Point......77
FTOI4 : Convert to Fixed Point......78
FTOI12 : Convert to Fixed Point......79
FTOI15 : Convert to Fixed Point......80
ITOF0 : Convert to Floating-Point Number......81
ITOF4 : Convert to Floating-Point Number......82
ITOF12 : Convert to Floating-Point Number......83
ITOF15 : Convert to Floating-Point Number......84
MADD : Product Sum......85
MADDi : Product Sum; with I Register......86
MADDq : Product Sum; by Q Register......87

MADDbc : Broadcast Product Sum 88
MADDA : Product Sum; to Accumulator 89
MADDAi : Product Sum; by I register, to Accumulator 90
MADDAq : Product Sum; by Q Register, to Accumulator 91
MADDAbc : Broadcast Product Sum; to Accumulator 92
MAX : Maximum Value 93
MAXi : Maximum Value 94
MAXbc : Maximum Value 95
MINI : Minimum Value 96
MINIi : Minimum Value 97
MINIbc : Minimum Value 98
MSUB : Multiply and Subtract 99
MSUBi : Multiply and Subtract; with I Register 100
MSUBq : Multiply and Subtract; by Q Register 101
MSUBbc : Broadcast Multiply and Subtract 102
MSUBA : Multiply and Subtract; to Accumulator 103
MSUBAi : Multiply and Subtract; with I Register, to Accumulator 104
MSUBAq : Multiply and Subtract; by Q Register, to Accumulator 105
MSUBAbc : Broadcast Multiply and Subtract; to Accumulator 106
MUL : Multiply 107
MULi : Multiply by I Register 108
MULq : Multiply by Q Register 109
MULbc : Multiply by Broadcast 110
MULA : Multiply; to Accumulator 111
MULAi : Multiply by I Register, to Accumulator 112
MULAq : Multiply by Q Register, to Accumulator 113
MULAbc : Broadcast Multiply by broadcast, to Accumulator 114
NOP : No Operation 115
OPMULA : Vector Outer Product 116
OPMSUB : Vector Outer Product 117
SUB : Subtract 118
SUBi : Subtract I Register 119
SUBq : Subtract Q Register 120
SUBbc : Broadcast Subtract 121
SUBA : Substract; to Accumulator 122
SUBAi : Subtract I Register; to Accumulator 123
SUBAq : Subtract Q Register; to Accumulator 124
SUBAbc : Broadcast Subtract; to Accumulator 125
4.3. Lower Instruction Reference 126
B : Unconditional Branch 127
BAL : Unconditional Branch with Saving Address 128
DIV : Divide 129
EATAN : Arctangent 130
EATANxy : Arctangent 132
EATANxz : Arctangent 134
EEXP : Exponent 136

ELENG : Length 137
ERCPR : Reciprocal Number 138
ERLENG : Reciprocal Number of Length 139
ERSADD : Reciprocal Number 140
ERSQRT : Reciprocal Number of Square Root 141
ESADD : Sum of Square Numbers 142
ESIN : Sine 143
ESQRT : Square Root 144
ESUM : Sum of Each Field 145
FCAND : Test Clipping Flag 146
FCEQ : Test Clipping Flag 147
FCGET : Get Clipping Flag 148
FCOR : Test Clipping Flag 149
FCSET : Setting Clipping Flag 150
FMAND : Test MAC Flag Check 151
FMEQ : Test MAC Flag Check 152
FMOR : Test MAC Flag Check 153
FSAND : Test Status Flag Check 154
FSEQ : Test Status Flag Check 155
FSOR : Test Status Flag 156
FSSET : Set Sticky Flags 157
IADD : ADD Integer 158
IADDI : Add Immediate Value Integer 159
IADDIU : Add Immediate Integer 160
IAND : Logical Product 161
IBEQ : Conditional Branch 162
IBGEZ : Conditional Branch 163
IBGTZ : Conditional Branch 164
IBLEZ : Conditional Branch 165
IBLTZ : Conditional Branch 166
IBNE : Conditional Branch 167
ILW : Integer Load with Offset Specification 168
ILWR : Integer Load 169
IOR : Logical Sum 170
ISUB : Integer Subtract 171
ISUBIU : Immediate Value Integer Subtract 172
ISW : Integer Store with Offset 173
ISWR : Integer Store 174
JALR : Unconditional Jump with Address Saving 175
JR : Unconditional Jump 176
LQ : Load Qword 177
LQD : Load Qword with Pre-Decrement 178
LQI : Load with Post-Increment 179
MFIR : Move from Integer Register to Floating-Point Register 180
MFP : Move from P Register to Floating-Point Register 181
MOVE : Transfer between Floating-Point Registers 182

MR32 : Move with Rotate 183
MTIR : Move from Floating-Point Register to Integer Register 184
RGET : Get Random Number 185
RINIT : Random Number Intialize 186
RNEXT : Next Random Number 187
RSQRT : Square Root Division 188
RXOR : Random Number Set 189
SQ : Store Qword with Offset 190
SQD : Store Qword with Pre-Decrement 191
SQI : Store with Post-Increment 192
SQRT : Square Root 193
WAITP : P Register Syncronize 194
WAITQ : Q Register Syncronize 195
XGKICK : GIF Control 196
XITOP : VIF Control 196
XTOP : VIF Control 198
5. Macro Mode 199
5.1. Macro Mode Register Set 200
5.1.1. Floating-Point Registers 200
5.1.2. Integer Registers 200
5.1.3. Control Registers 200
5.1.4. Special Registers 205
5.2. Macro Instruction Set Overview 206
5.2.1. MIPS COP2 Instructions 206
5.2.2. Coprocessor Transfer Instructions 206
5.2.3. Coprocessor Branch Instructions 206
5.2.4. Coprocessor Calculation Instructions 206
5.2.5. Micro Subroutine Execution Instructions 208
5.3. Flags 209
5.4. Macro Mode Pipeline 210
5.4.1. Pipeline Structure of Macroinstructions 210
5.4.2. Hazards in Macro Mode 210
5.4.3. Macroinstruction Operation 211
5.4.4. Operation when Transferring Data with EE Core 211
5.4.5. Operation when Executing a Micro Subroutine 215
5.4.6. Micro Subroutine and Data Transfer Operations 216
5.4.7. Q Register Synchronization 218
5.4.8. Notes on Other Pipeline Operations 218
5.5. VU1 Control 220
5.5.1. MIPS COP2 Condition Signal 220
5.5.2. MIPS COP2 Control Register 220
5.5.3. Floating-Point Registers 221
5.5.4. Integer Registers 222
5.5.5. Control Registers 222
6. Macro Mode Instruction Reference 225
6.1. Macro Instruction Operation Code 226

6.1.1. Macro Instruction Operation Type 226
6.1.2. Macro Instruction Operation Field 227
6.2. Macro Instruction Set 229
BC2F : Branch on COP2 Conditional Signal 230
BC2FL : Branch on COP2 Conditional Signal 231
BC2T : Branch on COP2 Conditional Signal 232
BC2TL : Branch on COP2 Conditional signal 233
CFC2 : Transfer Integer Data from VU to EE Core 234
CTC2 : Transfer Integer Data from EE Core to VU 235
LQC2 : Floating-Point Data Transfer from EE Core to VU 236
QMFC2 : Floating-Point Data Transfer from VU to EE Core 237
QMTC2 : Floating-Point Data Transfer from EE Core to VU 238
SQC2 : Floating-Point Data Transfer from VU to EE Core 239
VABS : Absolute Value 240
VADD : Add 241
VADDi : Add to I Register 242
VADDq : Add to Q Register 243
VADDbc : Broadcast Add 244
VADDA : Add to Accumulator 245
VADDAi : Add I Register to Accumulator 246
VADDAq : Add Q Register to Accumulator 247
VADDAbc : Broadcast Add to Accumulator 248
VCALLMS : Start Micro Sub-Routine 249
VCALLMSR : Start Micro Sub-Routine by Register 250
VCLIP : Clipping Judgment 251
VDIV : Divide 252
VFTOI0 : Conversion to Fixed Point 253
VFTOI4 : Conversion to Fixed Point 254
VFTOI12 : Conversion to Fixed Point 255
VFTOI15 : Conversion to Fixed Point 256
VIADD : Add Integer 257
VIADDI : Add Immediate Value Integer 258
VIAND : Logical Product 259
VILWR : Integer Load 260
VIOR : Logical Sum 261
VISUB : Integer Subtract 262
VISWR : Integer Store 263
VITOF0 : Conversion to Floating-Point Number 264
VITOF4 : Conversion to Floating-Point Number 265
VITOF12 : Conversion to Floating-Point Number 266
VITOF15 : Conversion to Floating-Point Number 267
VLQD : Load with Pre-Decrement 268
VLQI : Load with Post-Increment 269
VMADD : Product Sum 270
VMADDi : Product Sum; with I Register 271
VMADDq : Product Sum; with Q Register 272

VMADDbc : Broadcast Product Sum....273
VMADDA : Product Sum; to Accumulator....274
VMADDAi : Product Sum; with I Register, to Accumulator....275
VMADDAq : Product Sum; with Q Register, to Accumulator....276
VMADDAbc : Broadcast Product Sum; to Accumulator....277
VMAX : Maximum Value....278
VMAXi : Maximum Value....279
VMAXbc : Maximum Value....280
VMFIR : Transfer from Integer Register to Floating-Point Register....281
VMINI : Minimum Value....282
VMINIi : Minimum Value....283
VMINIbc : Minimum Value....284
VMOVE : Transfer between Floating-Point Registers....285
VMR32 : Vector Rotate....286
VMSUB : Multiply and Subtract....287
VMSUBi : Multiply and Subtract with I Register....288
VMSUBq : Multiply and Subtract; Q Register....289
VMSUBbc : Broadcast Multiply and Subtract....290
VMSUBA : Multiply and Subtract; to Accumulator....291
VMSUBAi : Multiply and Subtract; with I Register, to Accumulator....292
VMSUBAq : Multiply and Subtract; with Q Register, to Accumulator....293
VMSUBAbc : Broadcast Multiply and Subtract; to Accumulator....294
VMTIR : Transfer from Floating-Point Register to Integer Register....295
VMUL : Multiply....296
VMULi : Multiply; by I Register....297
VMULq : Multiply; by Q Register....298
VMULbc : Broadcast Multiply....299
VMULA : Multiply; to Accumulator....300
VMULAi : Multiply by I Register; to Accumulator....301
VMULAq : Multiply by Q Register; to Accumulator....302
VMULAbc : Broadcast Multiply; to Accumulator....303
VNOP : No Operation....304
VOPMULA : Vector Outer Product....305
VOPMSUB : Vector Outer Product....306
VRGET : Get Random Numbers....307
VRINIT : Random Number Initial Set....308
VRNEXT : New Random Numbers....309
VRSQRT : Square Root Division....310
VRXOR : Random Number Set....311
VSQD : Store with Pre-Decrement....312
VSQI : Store with Post-Increment....313
VSQRT : Square Root....314
VSUB : Subtract....315
VSUBi : Subtract I Register....316
VSUBq : Subtract Q Register....317
VSUBbc : Broadcast Subtract....318

VSUBA : Subtract; to Accumulator 319
VSUBAi : Subtract I Register; to Accumulator 320
VSUBAq : Subtract Q Register; to Accumulator 321
VSUBAbc : Broadcast Subtract; to Accumulator........ 322
VWAITQ : Q Register Synchronize 323
7. Appendix........ 325
7.1. Sample Micro Programs 326
7.2. EFU Processing........ 353
7.3. Micro Subroutine Debugging 362
7.3.1. Debug Flow........ 362
7.3.2. Notes on Re-execution........ 363
7.4. Throughput / Latency List 365

1. VU Overview

The VU is a vector ALU that efficiently performs four-element floating-point vector calculations. It is part of the VPU, along with the VU Mem (VU Data Memory) and the VIF (compressed data expansion engine).
Two VPUs are mounted on the EE as shown in Figure 1-1.

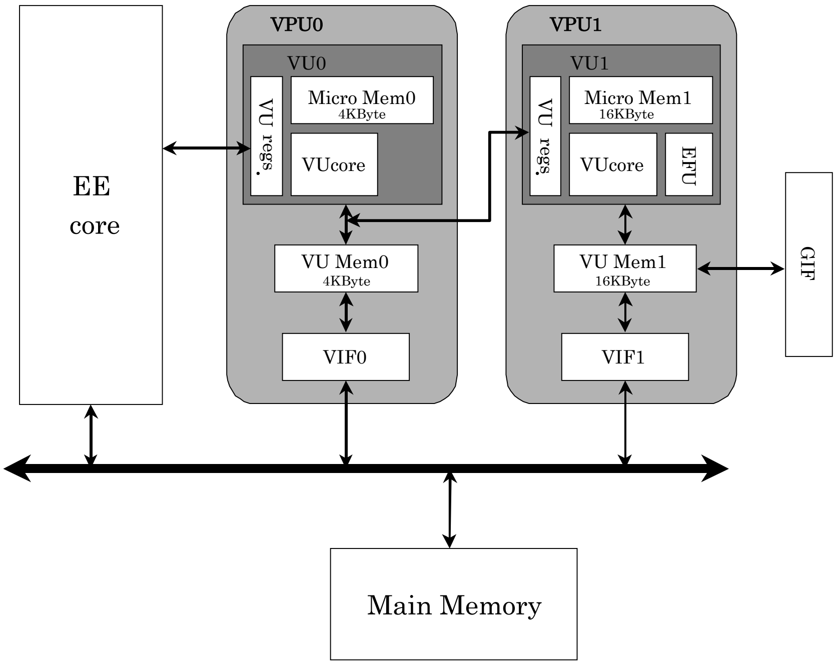


Figure 1-1 VPU System Outline

VU0 is joined to the EE Core as COP2 via a coprocessor connection. It assists the EE Core in non-stationary geometry processing. It has a 4-Kbyte instruction memory (MicroMem0) and a 4-Kbyte data memory (VU Mem0).
VU1 operates independently, and is chiefly in charge of background stationary geometry processing. VU1 has an Elementary Function Unit (EFU), as well as a 16-Kbyte instruction memory (MicroMem1) and a 16-Kbyte data memory (VU Mem1). VU1 is also connected to the GIF (the interface unit to the Graphics Synthesizer), and the GIF control instruction (XGKICK instruction) is mounted. VU1's floating-point and integer registers are mapped to VPU0's VU Mem0.

1.1. VPU Structure

Figure 1-2 is a block diagram of the VPU including the VU.

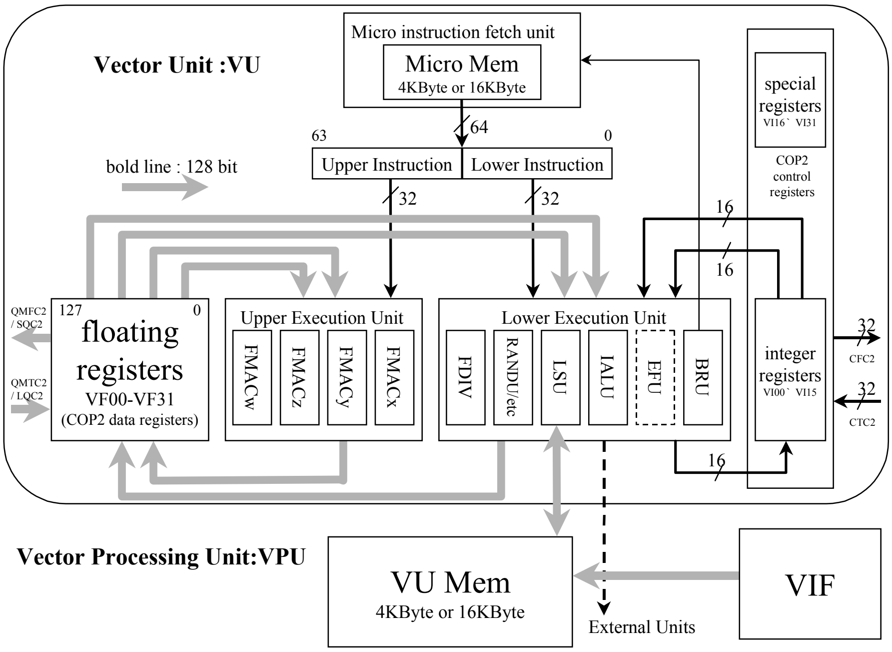


Figure 1-2 VU Outline Block Diagram

The VPU consists of the VU, VU Mem (VU Data Memory), and the VIF (compressed data expansion engine). The VU loads data in 128-bit units (single-precision floating-point number x 4) from the VU Mem, performs calculations according to micro programs in the VU's internal MicroMem, and stores the results in the VU Mem. The VU Mem may be used as a temporary area depending on the micro program.
Micro programs employ 64-bit length LIW (Long Instruction Word) instruction sets. They can concurrently execute a floating-point product-sum calculation in the upper 32 bits (the Upper instruction field) and a floating-point division or integer calculation in the lower 32 bits (the Lower instruction field).
There are 32 128-bit floating-point registers (single-precision floating-point x 4). There are 16 16-bit integer registers.

1.1.1. Upper Execution Unit

FMAC

This unit adds, subtracts, multiplies, and does product-sum operations on floating-point numbers. Four units are mounted in order to efficiently execute four-element vector calculations: FMACx, FMACy, FMACz, and FMACw. To increase the efficiency of pipeline processing, the latency of the instructions that use the FMAC has been unified at four cycles.

1.1.2. Lower Execution Unit

FDIV

This unit performs self-synchronous high-speed floating-point division/square root calculations. It uses a single-precision floating-point value as input, then stores the calculation result in the dedicated Q register. The next FDIV instruction cannot be executed while the FDIV is executing. The FDIV stalls if this is attempted.

LSU

This unit controls Load/Store to and from VU Mem.
Load/Store must be performed in units of 128 bits, but the x, y, z, and w field units can be masked.
There are two ways of specifying addresses. The first specifies a base (Integer) register and offsets in the operation code field. The second specifies the base register only, not the offsets, and performs post-incrementing or pre-decrementing. Incrementing/Decrementing is +1/-1 in 128-bit word address units.

IALU

This unit performs 16-bit integer calculations.
Loop counter calculation and Load/Store address calculation are performed using the Integer registers.

BRU

This unit controls jumping and conditional branching.
Most instructions specify PC-relative addresses for the jump target address. The offset is specified by an 11-bit immediate value, so it is possible to jump within a range of 8 Kbytes before and after the PC. The JR and JALR instructions are register indirect jump instructions, which use the data in a register as an absolute address.
Conditional branching is performed by a comparison with one or two Integer registers. When doing conditional branching based on the results of floating-point calculations, the results of the AND operation with the MAC flag, status flag, or clipping flag and the appropriate mask value are temporarily stored in an Integer register. Branching is performed by comparing with this register.

RANDU

This unit generates floating-point random numbers in the range $+1.0<r<+2.0$. Using the M series, the mantissa is created from the type that the user specified. Due to a feature of the M series, +1.0 does not appear in the random numbers. When 0 is specified as the type, only +1.0 is created, not a random number.
This M series is represented by the following polynomial:

$$p(x) = x^{23} + x^5 + 1$$

EFU

This is the Elementary Function Unit, which performs calculations such as exponential, logarithmic, and trigonometric functions. This unit is mounted only on VU1.
The EFU uses a scalar value (a single floating-point value) or vector value (four floating-point values) as input, then stores the scalar value from the calculation result in the dedicated P register. Calculation latency varies for each function. The next EFU instruction cannot be executed while the EFU is executing. The EFU stalls if this is attempted.

1.1.3. Floating-Point Registers

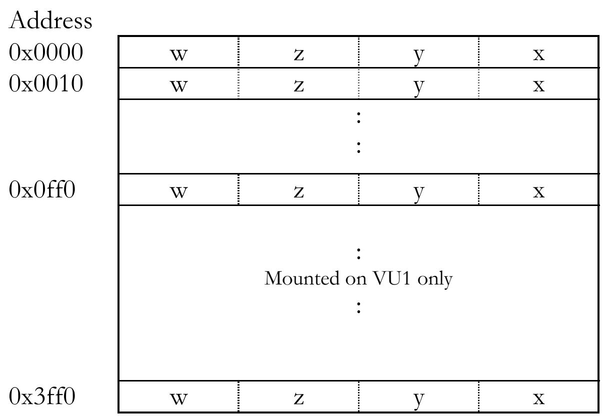
The VU has 32 128-bit floating-point registers (VF00 – VF31), which are equivalent to four single-precision floating-point values each. For a product-sum calculation, two 128-bit registers can be specified as source registers and one 128-bit register can be specified as the destination register.

1.1.4. Integer Register

The VU has sixteen 16-bit Integer registers (VI00 – VI15). These registers are used for loop counters and load/store address calculations.

1.1.5. VU Mem

The VU data memory capacity is 4 Kbytes for VU0, and 16 Kbytes for VU1. This memory is connected to the LSU (Load/Store Unit) at a width of 128 bits, and the address is qword (16 bytes) aligned. The effective data address must be divisible by 16: the address divided by 16 is specified in some instructions.



Furthermore, VU1 registers are mapped to addresses 0x4000 to 0x43ff in VU0.

1.1.6. Micro Mem

This on-chip memory stores 64-bit length LIW (Long Instruction Word) microinstructions. The capacity is 4 Kbytes for VU0, and 16 Kbytes for VU1.
Since the instruction length is 64 bits, instruction addresses must be divisible by 8. The address divided by 8 is specified in branches and other instructions.

Address		
0x0000	Upper	Lower
0x0008	Upper	Lower
	:	
	:	
0x0ff8	Upper	Lower
	:	
	Mounted on VU1 only	
	:	
0x3ff8	Upper	Lower

1.2. VU Execution Mode

There are two VU execution modes: micro mode and macro mode.
In micro mode, the VU functions as a stand-alone processor. It executes microinstruction programs stored in Micro Mem. VU1 operates in this mode.
In macro mode, the VU functions as COP2 (Coprocessor 2) of the EE Core. VU0 chiefly operates in this mode.
Macroinstructions lack some of the microinstruction-equivalent functions. Upper instructions and Lower instructions cannot be executed simultaneously in macroinstructions. However, it is possible to execute CALLMS instructions, which execute microinstruction programs in MicroMem as subroutines, and COP2 data transfer instructions, which transfer data to and from a VU register.

	Micro mode (VU1/VU0)	Macro mode (VU0)
Operation	Operates as a stand-alone processor	Operates as a coprocessor of EE Core.
Operation code	64-bit long LIW instructions	32-bit MIPS COP2 instructions
Instruction set	Upper instruction+ Lower instruction (Can be specified simultaneously) EFU instructions (VU1 only) External unit control instructions (VU1 only)	Upper instruction Lower instruction (partial) VCALLMS, VCALLMSR COP2 transfer instructions
Total instruction count	127 instructions	90 instructions
EFU	Is usable as an option (VU1 only)	Is not supported
Registers	Floating-point registers: 32 x 128 bits Integer registers: 16 Special registers: ACC, I, Q, R (,P)	Floating-point registers: 32 x 128 bits Integer registers: 16 Special registers: ACC, I, Q, R Control registers: 16

1.3. VU Operation Status

The VU has three operation states: Ready, Run, and Stop. The following sections explain these states and their transitions to other states. See also "**5.1.3. Control Registers**" for information regarding state transitions.

1.3.1. Ready State

The Ready state is a stand-by state. When power is turned on, the VU goes into the Ready state, and receives micro subroutine start-up, macroinstructions, and coprocessor transfer instructions from the EE Core. The VU can also receive micro program start-up from the VIF.
The VU shifts from the Ready state to the Run state when a micro subroutine is started or a macroinstruction is executed. On reset, the control register is initialized and the VU goes into the Ready state again.
The VU enters the Stop state if a ForceBreak occurs.

1.3.2. Run State

The Run state is an execution state. In this state, the VU cannot receive micro subroutine start-up nor macroinstructions from the EE Core. If it does, the EE Core stalls. A coprocessor transfer instruction may or may not stall, according to the user specification.
The VU shifts from the Run state to the Ready state at micro subroutine E bit termination or macroinstruction execution termination. The VU shifts to the Stop state when a D bit halt, T bit halt, or ForceBreak occurs during execution of a micro subroutine.

1.3.3. Stop State

The Stop state is used for debugging. In this state, the VU can receive micro subroutine start-up and coprocessor transfer instructions from the EE Core. Operation is indeterminate if a macroinstruction is executed. The VU cannot receive micro program start-up from the VIF.
The VU shifts from the Stop state to the Run state when the VCALLMS or VCALLMSR instruction is executed, or the CMSAR1 register is written to. The control register is initialized by resetting, then the VU enters the Ready state. There is no status shift with a ForceBreak.
From the point of view of the VIF, there is no difference between the Run state and the Stop state.

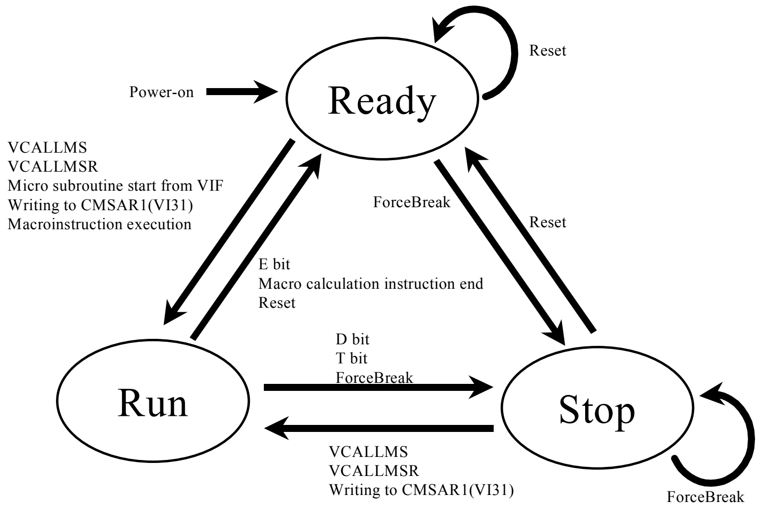


Figure 1-3 VU Status Shift

1.4. VU Usage

This section briefly explains VU usage from the point of view of the EE Core. For details, see the document "EE User's Manual".

1.4.1. VU1 Usage Outline

VU1 performs independently of the EE Core, as a preprocessor of the GS. For this reason, it has a unique on-chip data memory (VU Mem1) and instruction memory (MicroMem1), and is directly connected to the GS via the GIF.
Microinstruction programs executed in VU1 are DMA-transferred from main memory to MicroMem1 via VIF1.
Data required by VU1 is transferred to VU Mem1 via VIF1.
Two methods are used to start the microinstruction programs transferred to VU1:

1) Write the execution address to the control register (CMSAR1).
2) Specify the execution address by the VIFcode (MSCAL/MSCALF).

1) is used when returning from the Stop state and 2) is used in ordinary cases.

1.4.2. VU0 Usage Outline

Since VU0 is joined to the EE Core via a coprocessor connection, the VU0 resources can be controlled directly by EE Core instructions (Macro mode).
By transferring a microinstruction program to VU0 on-chip instruction memory (MicroMem0) in the same way as VU1, VU0 can be activated as a micro subroutine (Micro Mode).
MicroMem0 and VUMem0 are accessible by EE Core instructions and via VIF0 in the same way as VU1.
Two methods are used to start microinstruction programs in VU0:

1) Execute VCALLMS/VCALLMSR instruction.
2) Specify the execution address by the VIFcode (MSCAL/MSCALF).

(This page is left blank intentionally)

2. Data/Calculation Basic Specifications

2.1. Data Format

Data used by the VU consists of single-precision floating-point values based on IEEE 754 and four types of 32-bit fixed-point values. The formats are explained below.

2.1.1. Floating-Point Values

Only single-precision (32-bit) floating-point values are supported. The bit fields comply with IEEE 754 as follows:

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

S	E	F
1	8 bits	23 bits

Sign	S: 1 bit	
Exponents	E: 8 bits	Biased exponent for bias value 127
Mantissa	F: 23 bits	Excluding hidden bits

The actual mantissa adds 1 hidden bit to the beginning, becoming 1.F. The normalized value is $(-1)^{S} \times 1.F \times 2^{(E-127)}$ and zero is E = (F =) 0. The VU does not support non-numeric, infinity, and non-normalized values based on IEEE 754. See the table below.

<table>
<tr><th>Exponent E</th><th>Mantissa F</th><th colspan="2">Value with IEEE 754</th><th colspan="2">Value with VU</th></tr>
<tr><td>255</td><td>Other than 0</td><td colspan="2">Non numeric</td><td colspan="2">$x\ 2^{(+128)}$</td></tr>
<tr><td>255</td><td>0</td><td colspan="2">+/- infinity</td><td colspan="2">$x\ 2^{(+128)}$</td></tr>
<tr><td>254
:::
128</td><td rowspan="2"></td><td rowspan="2">Normalization count
$(-1)^{S}$ x 1.F</td><td>$x\ 2^{(+127)}$
:::
$x\ 2^{(+1)}$</td><td rowspan="2">Normalization count
$(-1)^{S}$ x 1.F</td><td>$x\ 2^{(+127)}$
:::
$x\ 2^{(+1)}$</td></tr>
<tr><td>127
:::
1</td><td>$x\ 2^{(+0)}$
:::
$x\ 2^{(-126)}$</td><td>$x\ 2^{(+0)}$
:::
$x\ 2^{(-126)}$</td></tr>
<tr><td>0</td><td>Other than 0</td><td colspan="2">Non-normalized value</td><td colspan="2">0</td></tr>
<tr><td>0</td><td>0</td><td colspan="2">0</td><td colspan="2">0</td></tr>
</table>

2.1.2. Fixed-Point Values

The VU supports four formats of 32-bit fixed-point values. The formats specify the number of bits to the right of the decimal point: 0 bit, 4 bits, 12 bits, and 15 bits. 2's complement expressions are used for negative numbers.

Format	Range of values			Expression precision
0-bit fixed point	+2147483647	–	-2147483648	1
4-bit fixed point	+134217720	–	-134217720	0.0625
12-bit fixed point	+524287.96875	–	-524287.96875	0.000244140625
15-bit fixed point	+65535.99609375	–	-65535.99609375	0.000030517578125

2.2. Rounding Off Floating-Point Values

When calculating floating-point values and converting them to and from fixed-point values, rounding off is performed as follows:

- Calculation

  A 24-bit calculation including hidden bits is performed, and the result is truncated. The rounding-off operation in IEEE 754 is performed in the 0 direction, so the values for the least significant bit may vary.

- Conversion to fixed point

  When converting from floating point to fixed point, truncation is made in the 0 direction.

- Conversion from fixed point

  When converting from fixed point to floating point, truncation is made in the 0 direction. If the valid bit count of the fixed-point value exceeds 24 bits, the upper 24 valid bits of the absolute value become the mantissa, which includes the hidden bits. The remaining bits are truncated.

2.3. Exception Processing

An exception in the VU means that calculation results differ from normal results (such as division by 0 and overflow). The VU does not pause when an exception is generated. A flag is set, clamping of the calculation results is performed, then processing continues.

Exceptions in floating-point calculations are shown in the table below.

Exception	Calculation result	MAC / status flag	Sticky flag
0/0 (0 is valid sign)	+MAX/-MAX	I flag = 1	IS flag = 1
$\sqrt{x}$ (x < 0)	$\sqrt{\lvert x \rvert}$	I flag = 1	IS flag = 1
0 division	+MAX/-MAX	D flag = 1	DS flag = 1
Exponent overflow	+MAX/-MAX	Ox/Oy/Oz/Ow/O flag = 1	OS flag = 1
Exponent underflow	+0/-0	Ux/Uy/Uz/Uw/U flag = 1 Zx/Zy/Zz/Zw/Z flag = 1	US flag = 1 ZS flag = 1
Conversion overflow (Floating point count -> fixed point count)	+MAX/-MAX	None	None

Note: D flag is not set for 0/0.

2.4. Differences from IEEE 754

The following are differences between VU calculations and the IEEE 754 standard:

- Precision

  The VU supports only single-precision floating-point values.

- Rounding off

  Of the four IEEE 754 rounding-off modes, the VU performs rounding similar to a 0 direction truncation. Since the least significant bit may vary, this method is not exactly the same as IEEE 754.

- Non-numeral/infinity/non-normalized number

  The VU does not support non-numerals, infinity, and non-normalized numbers. Therefore, exceptions related to these items are not generated.

- Overflow/underflow

  Overflow and underflow are detected only by the overflow and underflow of the exponent. Underflow in particular is not considered to lose the precision prescribed by IEEE 754.

- Trap for exception

  IEEE 754 recommends that a trap be settable for a calculation result exception. In the VU, processing continues by just setting a flag. If a trap is necessary, use a flag check instruction after the calculation instruction.

(This page is left blank intentionally)

3. Micro Mode

In Micro mode, the VU executes microinstruction programs (micro subroutines) in MicroMem as a stand-alone processor. Using the VU in micro mode maximizes the degree of parallelism, and the highest performance can be achieved.
Microinstructions are 64-bit long LIW instructions. It is possible to specify both an Upper instruction and a Lower instruction at the same time. The Upper instruction controls four floating-point product-sum ALUs (FMAC), and the Lower instruction controls one floating-point division/square root ALU (FDIV), one load-store unit (LSU), and one integer operation unit (IALU), etc. A maximum of six units can execute concurrently.

3.1. Micro Mode Register Set

In micro mode, 32 floating-point registers, 16 integer registers, and other special registers can be used. These registers are explained below.

3.1.1. Floating-Point Registers

There are 32 floating-point registers, VF00 - VF31. These are 128-bit long vector registers, which consist of four single-precision floating-point fields.
The four 32-bit fields have a little-endian arrangement. From the least to the most significant bit, they are referred to as field x, field y, field z, and field w.

	32 bits (127–96)	32 bits (95–64)	32 bits (63–32)	32 bits (31–0)
VF00	VF00w	VF00z	VF00y	VF00x
VF01	VF01w	VF01z	VF01y	VF01x
VF02	VF02w	VF02z	VF02y	VF02x
VF03	VF03w	VF03z	VF03y	VF03x
		:		
		:		
VF31	VF31w	VF31z	VF31y	VF31x

VF00 is the constant register. Its fields are set to the following values:

- VF00x: 0.0 (Single-precision floating number)
- VF00y: 0.0 (Single-precision floating number)
- VF00z: 0.0 (Single-precision floating number)
- VF00w: 1.0 (Single-precision floating number)

3.1.2. Integer Registers

There are 16 16-bit long integer registers, VI00 - VI15. VI00 is the constant register and is set to 0.

	15 0
VI00	0 Register
VI01	
VI02	
VI03	
VI04	
VI05	
VI06	
VI07	
VI08	
VI09	
VI10	
VI11	
VI12	
VI13	
VI14	Stack pointer (recommended)
VI15	Link register (recommended)

3.1.3. ACC Registers

The ACC registers are accumulators for floating-point product-sum calculations. Four registers exist for four product-sum ALUs: ACCx, ACCy, ACCz, and ACCw. The ACC registers not only work as destinations of instructions such as ADDA and MULA but also store the intermediate results of the vector outer product calculated by OPMULA and OPMULB.
Stalls due to data dependency, described later in this document, do not occur to the ACC register.

3.1.4. I Register

The I register is a 32-bit single-precision floating-point register in which immediate values are stored. When the I bit (bit 63) of the Upper OP field is set, the content of the Lower OP field is written to the I register at the T stage of the instruction as a single-precision floating-point number. It is used by the next instruction to be executed, such as ADDi/MULi. This operation can be described with the LOI pseudo instruction.
No stalls due to data dependency are generated for the I register. Figure 3-1 illustrates a pipeline operation example in which the I register is used.

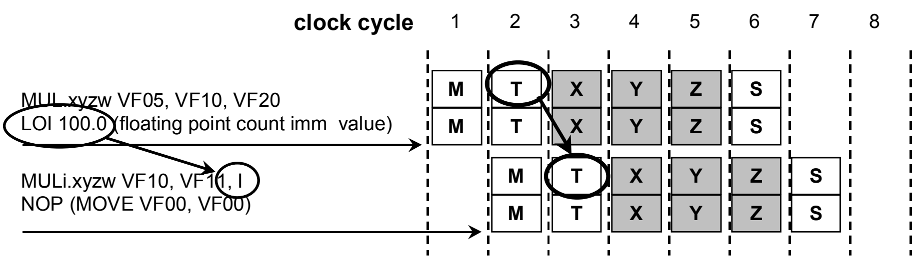


Figure 3-1 Usage Example of I Register

For more information on pipeline operation, see "3.4. Pipeline Operation".

3.1.5. Q Register

The Q register is a floating-point register in which results of division, square root operations, and square root division are stored. Because these calculations differ in latency from other floating-point calculations, a special register (the Q register) is needed.
No stalls due to data dependency are generated for the Q register, and it is necessary to use the WAITQ instruction for synchronization.

3.1.6. R Register

The R register is a 23-bit register in which random number values are stored.

3.1.7. P Register

The P register is a 32-bit register in which the EFU instruction results are stored. Because elementary function calculations differ in latency from other floating-point calculations, a special register (the P register) is needed.
No stalls due to data dependency are generated for the P register, and it is necessary to use the WAITP instruction for synchronization.

3.2. Micro Instruction Set Overview

A microinstruction is a 64-bit LIW (Long Instruction Word). It can specify instructions in the Upper instruction field (the upper 32 bits) and the Lower instruction field (the lower 32 bits) independently.
Instructions that use floating-point product-sum ALUs (FMAC) are usually specified in the Upper instruction field, and other instructions are specified in the Lower instruction field.

63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32

Upper Instruction Field
Upper Instruction

32 bits

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

Lower Instruction Field
Lower Instruction

32 bits

3.2.1. Upper Instructions

The Upper instructions are mainly related to floating-point calculations. There are 59 instructions.

Category	Instruction	Function
Floating- point calculation	ABS	absolute
	ADD	addition
	ADDi	ADD broadcast I register
	ADDq	ADD broadcast Q register
	ADDbc	ADD broadcast bc field
	ADDA	ADD output to ACC
	ADDAi	ADD output to ACC broadcast I register
	ADDAq	ADD output to ACC broadcast Q register
	ADDAbc	ADD output to ACC broadcast bc field
	SUB	subtraction
	SUBi	SUB broadcast I register
	SUBq	SUB broadcast Q register
	SUBbc	SUB broadcast bc field
	SUBA	SUB output to ACC
	SUBAi	SUB output to ACC broadcast I register
	SUBAq	SUB output to ACC broadcast Q register
	SUBAbc	SUB output to ACC broadcast bc field
	MUL	multiply
	MULi	MUL broadcast I register
	MULq	MUL broadcast Q register
	MULbc	MUL broadcast bc field
	MULA	MUL output to ACC
	MULAi	MUL output to ACC broadcast I register
	MULAq	MUL output to ACC broadcast Q register
	MULAbc	MUL output to ACC broadcast bc field
	MADD	MUL and ADD
	MADDi	MUL and ADD broadcast I register
	MADDq	MUL and ADD broadcast Q register
	MADDbc	MUL and ADD broadcast bc field

Category	Instruction	Function
	MADDA	MUL and ADD output to ACC
	MADDAi	MUL and ADD output to ACC broadcast I register
	MADDAq	MUL and ADD output to ACC broadcast Q register
	MADDAbc	MUL and ADD output to ACC broadcast bc field
	MSUB	MUL and SUB
	MSUBi	MUL and SUB broadcast I register
	MSUBq	MUL and SUB broadcast Q register
	MSUBbc	MUL and SUB broadcast bc field
	MSUBA	MUL and SUB output to ACC
	MSUBAi	MUL and SUB output to ACC broadcast I register
	MSUBAq	MUL and SUB output to ACC broadcast Q register
	MSUBAbc	MUL and SUB output to ACC broadcast bc field
	MAX	maximum
	MAXi	MAX broadcast I register
	MAXbc	MAX broadcast bc field
	MINI	minimum
	MINIi	MINI broadcast I register
	MINIbc	MINI broadcast bc field
	OPMULA	outer product MULA
	OPMSUB	outer product MSUB
	NOP	no operation
Floating-point/fixed-point conversion	FTOI0	float to integer, fixed point 0 bit
	FTOI4	float to integer, fixed point 4 bits
	FTOI12	float to integer, fixed point 12 bits
	FTOI15	float to integer, fixed point 15 bits
	ITOF0	integer to float, fixed point 0 bit
	ITOF4	integer to float, fixed point 4 bits
	ITOF12	integer to float, fixed point 12 bits
	ITOF15	integer to float, fixed point 15 bits
Clipping judgment	CLIP	clipping

3.2.2. Lower Instructions

The Lower instructions are floating-point division, integer calculation, transfer between registers, flag operation, branching, and elementary function calculation, and other control instructions. There are 69 instructions listed below.

The NOP instruction is not included in the Lower instructions. If necessary, use meaningless instructions such as MOVE VF00, VF00 in place of NOP.

Category	Instruction	Function
Floating-point division	DIV	floating divide
	SQRT	floating square-root
	RSQRT	floating reciprocal square-root
Integer calculation	IADD	integer ADD
	IADDI	integer ADD immediate
	IADDIU	integer ADD immediate unsigned
	IAND	integer AND
	IOR	integer OR
	ISUB	integer SUB
	ISUBIU	integer SUB immediate unsigned
Register-register transfer	MOVE	move floating register
	MFIR	move from integer register
	MTIR	move to integer register
	MR32	move rotate 32 bits
Load/Store	LQ	Load Quadword
	LQD	Load Quadword with pre-decrement
	LQI	Load Quadword with post-increment
	SQ	Store Quadword
	SQD	Store Quadword with pre-decrement
	SQI	Store Quadword with post-increment
	ILW	integer load word
	ISW	integer store word
	ILWR	integer load word register
	ISWR	integer store word register
	LOI	Load immediate value to I register (pseudo instruction)
Random numbers	RINIT	random-unit init R register
	RGET	random-unit get R register
	RNEXT	random-unit next M sequence
	RXOR	random-unit XOR R register
Synchroniza-tion	WAITQ	wait Q register
Flag operation	FSAND	flag-operation status flag AND
	FSEQ	flag-operation status flag EQ
	FSOR	flag-operation status flag OR
	FSSET	flag-operation set status flag
	FMAND	flag-operation MAC flag AND
	FMEQ	flag-operation MAC flag EQ
	FMOR	flag-operation MAC flag OR
	FCAND	flag-operation clipping flag AND
	FCEQ	flag-operation clipping flag EQ
	FCOR	flag-operation clipping flag OR
	FCSET	flag-operation clipping flag set
	FCGET	flag-operation clipping flag get
Branching	IBEQ	integer branch on equal

Category	Instruction	Function
	IBGEZ	integer branch on greater than or equal to zero
	IBGTZ	integer branch on greater than zero
	IBLEZ	integer branch on less than or equal to zero
	IBLTZ	integer branch on less than zero
	IBNE	integer branch on not equal
	B	branch (PC relative address)
	BAL	branch and link (PC relative address)
	JR	jump register (absolute address)
	JALR	jump and link register (absolute address)
EFU transfer	MFP	move from P register
EFU synchroniza-tion	WAITP	wait P register
Vector elementary function	ESADD	Elementary-function Square and ADD
	ERSADD	Elementary-function Reciprocal Square and ADD
	ELENG	Elementary-function Length
	ERLENG	Elementary-function Reciprocal Length
	EATANxy	Elementary-function ArcTAN y/x
	EATANxz	Elementary-function ArcTAN z/x
	ESUM	Elementary-function Sum
Scalar elementary function	ERCPR	Elementary-function Reciprocal
	ESQRT	Elementary-function Square-root
	ERSQRT	Elementary-function Reciprocal Square-root
	ESIN	Elementary-function SIN
	EATAN	Elementary-function ArcTAN
	EEXP	Elementary-function Exponential
External unit control	XGKICK	Kick external unit (GIF)
	XTOP	Read VIF data (TOP register)
	XITOP	Read VIF data (ITOP register)

3.3. Flags

VU flags are roughly divided into 3 types: MAC flags which show floating-point calculation results, status flags which show total calculation results, and clipping flags which show clipping judgment results.

3.3.1. MAC Flags

The MAC flags show the FMAC calculation results of the Upper instruction (or the macroinstruction corresponding to it). There are four flag types, z, s, u and o, and each of them has 1 bit corresponding to the four floating-point calculation units, FMACx, FMACy, FMACz, and FMACw (16 bits in total).

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
O field				U field				S field				Z field			
Ox	Oy	Oz	Ow	Ux	Uy	Uz	Uw	Sx	Sy	Sz	Sw	Zx	Zy	Zz	Zw
4 bits				4 bits				4 bits				4 bits			

Z field: Zero flag

This flag is set to 1 when the calculation results are 0, and set to 0 when the calculation results are non-zero.

S field: Sign flag

This flag is set to 1 when the calculation results are negative, and set to 0 when the calculation results are positive or 0.

U field: Underflow flag

This flag is set to 1 when the calculation results cause an underflow.

O field: Overflow flag

This flag is set to 1 when the calculation results cause an overflow.

The flags of ALUs that do not operate are cleared to 0. In the ADD.xyz instruction, for example, the Zw, Sw, Uw, and Ow flags become 0 since FMACw does not operate.
Also, the MAC flags do not change when NOPs are executed.
When an overflow or underflow occurs, the Z and S flags are set according to the clamped results (+MAX/-MAX/+0/-0).

3.3.2. Status Flags (SF)

Status flags consist of the following 12 bits. There are flags to show MAC flag status, flags to show invalid calculation, and flags to show their accumulation (sticky flag).

11	10	09	08	07	06	05	04	03	02	01	00
DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z

Z: Zero flag

This flag is set to 1 when any of the Zx, Zy, Zz, and Zw bits of the MAC flags is set to 1.

S: Sign flag

This flag is set to 1 when any of the Sx, Sy, Sz, and Sw bits of the MAC flags is set to 1.

U: Underflow flag

This flag is set to 1 when any of the Ux, Uy, Uz, and Uw bits of the MAC flags is set to 1.

O: Overflow flag

This flag is set to 1 when any of the Ox, Oy, Oz, and Ow bits of the MAC flags is set to 1.

I: Invalid flag

This flag is set to 1 when a 0/0 calculation is executed by the DIV instruction or a root calculation of negative number is executed by the SQRT/RSQRT instruction.

D: Zero division flag

This flag is set to 1 when 0 division (except 0/0) is performed by the DIV/RSQRT instruction, and set to 0 when the SQRT instruction is executed regardless of the results.

ZS/SS/US/OS/IS/DS: Sticky flag

Six flags, ZS, SS, US, OS, IS, and DS, are called Sticky Flags, and indicate the accumulation values of the Z, S, U, O, I, and D flags, respectively. For example, the ZS flag value becomes the logical OR of the Z flag value and the ZS flag value from the results of the most-recently performed instruction.
In product-sum calculations, a single instruction, such as MADD/MSUB, performs addition and subtraction following multiplication. Addition and subtraction results are reflected in the Z/ZS/S/SS/U/US/O/OS flags, and multiplication results are reflected only in the ZS/SS/US/OS flags.

3.3.3. Clipping Flags (CF)

The clipping flags are set according to the results of the clipping judgment (CLIP) instruction.

-x flag Set to 1 when x < -|w|.
+x flag Set to 1 when x > +|w|.
-y flag Set to 1 when y < -|w|.
+y flag Set to 1 when y > +|w|.
-z flag Set to 1 when z < -|w|.
+z flag Set to 1 when z > +|w|.

There are four sets of clipping flags, as shown below. The clipping flag is shifted 6 bits to the left each time the CLIP instruction is executed, so clipping information for the four most recent vertices is always maintained.

23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
3rd previous judgment						2nd previous judgment						Previous judgment						Current judgment					
-z	+z	-y	+y	-x	+x	-z	+z	-y	+y	-x	+x	-z	+z	-y	+y	-x	+x	-z	+z	-y	+y	-x	+x
6 bits						6 bits						6 bits						6 bits					

3.3.4. Flag Set Instructions

The following instructions (Lower instructions) are used to set the flags to a specific value.

FSSET instruction: Sets the status flag.
FCSET instruction: Sets the clipping flag.

The effect on the flag of the simultaneous Upper instruction is ignored and; the flag reflects only the results of the flag setting instructions.

3.3.5. Flag Changes for Each Instruction

The flag changes for each micorinstruction are shown in the table below.
"-" means "no flag change" and "X" means "flag change according to calculation results".
For the MAC flags, x, y, z, and w are grouped in the same column. Instructions of the same kind are grouped with "*", for example ADD*.

Upper Instruction

Instruction	MAC Flag	Status Flag		Clipping Flag
	OUSZ	DIOUSZ SSSSSS	DIOUSZ	
ABS	----	------	------	-
ADD*	XXXX	--XXXX	--XXXX	-
ADDA*	XXXX	--XXXX	--XXXX	-
CLIP	----	------	------	X
FTOI*	----	------	------	-
ITOF*	----	------	------	-
MADD*	XXXX	--XXXX	--XXXX	-
MADDA*	XXXX	--XXXX	--XXXX	-
MAX*	----	------	------	-
MINI*	----	------	------	-
MSUB*	XXXX	--XXXX	--XXXX	-
MSUBA*	XXXX	--XXXX	--XXXX	-
MUL*	XXXX	--XXXX	--XXXX	-
MULA*	XXXX	--XXXX	--XXXX	-
NOP	----	------	------	-
OPMULA	XXXX	--XXXX	--XXXX	-
OPMSUB	XXXX	--XXXX	--XXXX	-
SUB*	XXXX	--XXXX	--XXXX	-
SUBA*	XXXX	--XXXX	--XXXX	-

Lower Instruction

Instruction	MAC Flag	Status Flag		Clipping Flag
	OUSZ	DIOUSZ SSSSSS	DIOUSZ	
DIV	----	XX----	XX----	-
RSQRT	----	XX----	XX----	-
SQRT	----	-X----	0X----	-
FCSET	----	------	------	X
FSSET	----	XXXXXX	------	-
Others	----	------	------	-

3.3.6. Flag Changes for Exception Occurrences

Flag changes in the basic instructions (except MADD/MSUB)

The following table shows the results and flag changes for instructions other than MADD or MSUB in which calculation exceptions are generated.

Exception	Calculation Results	MAC Flag				Status Flag				Sticky Flag			
		Z*	S*	U*	O*	Z	S	U	O	ZS	SS	US	OS
No exception	X	X	X	0	0	X	X	0	0	X	X	-	-
Overflow	MAX	0	X	0	1	0	X	0	1	-	X	-	1
Underflow	0	1	X	1	0	1	X	1	0	1	X	1	-

X: 1 or 0 according to the calculation results -: Same as past values

MAC Flag is set to 0 when the corresponding FMAC unit is not used.

Flag change in MADD instruction

In the MADD instruction, addition/subtraction with the accumulator is performed following the multiplication. There is a double exception generation factor in one instruction, and flags are set in compliance with both the multiplication and addition/subtraction. Therefore, flag settings in this instruction are complicated.

Exception Factor		Calculation Results	MAC Flag				Status Flag				Sticky Flag			
ACC	Multiplication		Z*	S*	U*	O*	Z	S	U	O	ZS	SS	US	OS
0/Normalized value	No exception	X	X	X	0	0	X	X	0	0	X	X	-	-
0/Normalized value	OVF	+/-MAX	0	X	0	1	0	X	0	1	-	X	-	1
0/Normalized value	UDF	-	X	X	0	0	X	X	0	0	X	X	1	-
+/-MAX	No exception	+/-MAX	0	X	0	1	0	X	0	1	b	c	-	1
+MAX	OVF(+MAX)	+MAX	0	0	0	1	0	0	0	1	-	b	-	1
+MAX	OVF(-MAX)	-MAX	0	1	0	1	0	1	0	1	-	1	-	1
-MAX	OVF(+MAX)	+MAX	0	0	0	1	0	0	0	1	-	b	-	1
-MAX	OVF(-MAX)	-MAX	0	1	0	1	0	1	0	1	-	1	-	1
+/-MAX	UDF	+/-MAX	0	X	0	1	0	X	0	1	1	c	1	1
Addition/Subtraction OFV/UDF		+/-MAX/0	X	X	X	X	X	X	X	X	c	c	a	a

X: 1 or 0 according to the calculation results of addition -: Same as past values

a: Logical OR of the flag value that shows calculation results of addition and the past flag values

b: Logical OR of the flag value that shows calculation results of multiplication and the past flag values

c: Logical OR of the flag values that show calculation results of addition and multiplication and the past flag values

MAC Flag is set to 0 when the corresponding FMAC unit is not used.

Flag changes in MSUB instruction

The flag changes in the MSUB instruction are almost the same as those in the MADD instruction, but positive and negative signs are changed when an overflow occurs to multiplication.

Exception Factor		Calculation Results	MAC Flag				Status Flag				Sticky Flag			
ACC	Multiplication		Z*	S*	U*	O*	Z	S	U	O	ZS	SS	US	OS
0/Normalized value	No exception	X	X	X	0	0	X	X	0	0	X	X	-	-
0/Normalized value	OVF	+/-MAX	0	X	0	1	0	X	0	1	-	X	-	1
0/Normalized value	UDF	-	X	X	0	0	X	X	0	0	X	X	1	-
+/-MAX	No exception	+/-MAX	0	X	0	1	0	X	0	1	b	c	-	1
+MAX	OVF(+MAX)	-MAX	0	1	0	1	0	1	0	1	-	1	-	1
+MAX	OVF(-MAX)	+MAX	0	0	0	1	0	0	0	1	-	b	-	1
-MAX	OVF(+MAX)	-MAX	0	1	0	1	0	1	0	1	-	1	-	1
-MAX	OVF(-MAX)	+MAX	0	0	0	1	0	0	0	1	-	b	-	1
+/-MAX	UDF	+/-MAX	0	X	0	1	0	X	0	1	1	c	1	1
Addition/Subtraction OFV/UDF		+/-MAX/0	X	X	X	X	X	X	X	X	c	c	a	a

X: 1 or 0 according to the calculation results of addition -: Same as past values

a: Logical add of the flag value that shows calculation results of addition and the past flag values

b: Logical add of the flag value that shows calculation results of multiplication and the past flag values

c: Logical add of the flag values that show calculation results of addition and multiplication and the past flag values

MAC Flag is set to 0 when the corresponding FMAC unit is not used.

3.4. Pipeline Operation

3.4.1. Hazards

VU calculations are performed concurrently via the pipeline, but some operations may stall occasionally under the conditions specified below. For concrete operations, see sections "3.4.4. FMAC Pipeline" and following.

DIV Resource Hazards

An instruction (DIV/SQRT/RSQRT) that uses the floating-point divider unit when another instruction of this type is being executed.

EFU Resource Hazards

An instruction (such as ESIN) that uses the EFU when another instruction of this type is being executed.

Floating-Point Register Data Hazards

An instruction that uses a floating-point register when another instruction that uses the register as the destination is being executed (until the value is fixed).
Data hazard checks are performed independently in each field of x/y/z/w. VF00 is a constant register, and is not subject to hazard checks.

Integer Register Data Hazards

An instruction that uses the values of an integer register when a load/store instruction to the integer register is being executed.
Since integer calculation latency is 1 clock, data hazards due to calculations are not generated. VI00 is not subject to hazard checks.

Data hazards are not generated for the special registers such as ACC, I, Q, P, and R. It is possible to make the Q and P registers synchronize with each other by using the WAITQ/WAITP instruction.
When transferring data to an external unit (GS) by the XGKICK instruction, stalls continue until the data can be transferred.

3.4.2. Upper Instruction and Lower Instruction

In micro mode, there is an Upper instruction pipeline and a Lower instruction pipeline. The Upper and Lower instructions are issued concurrently, so both pipelines stall if hazards occur to either of them.

3.4.3. Priority for Writing to a Register

When the Upper and Lower instructions write data to the same register at the same time, priority is given to the Upper instruction and the result of the Lower instruction is discarded. Moreover, when a coprocessor transfer instruction (COP2) writes data at the same time, it is given priority.

COP2 Transfer Instruction> Upper Instruction> Lower Instruction

The above is performed in register units, so note that the result of the Lower instruction is discarded even when the Upper and Lower instructions write data to different fields as shown in the following example.

ADD.xy VF01, VF01, VF23 MOVE.w VF01, VF09

3.4.4. FMAC Pipeline

Figure 3-2 illustrates the FMAC pipeline. All instructions except some of the Lower instructions (DIV, SQRT, RSQRT, integer calculation, and conditional branching) are executed according to this pipeline. Load/Store of integer registers also follows this pipeline.
The MAC flag, status flag and clipping flag, which show the results of the calculation, are set at the S stage.

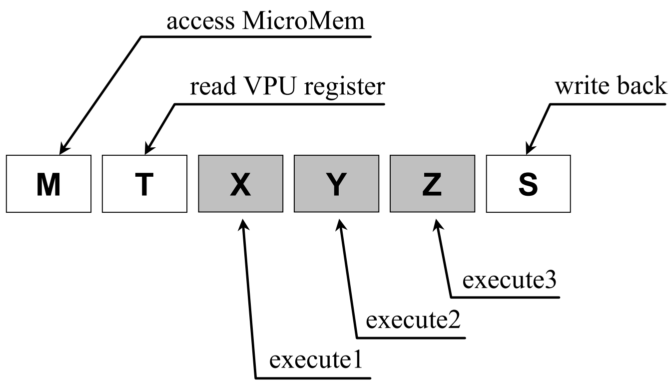


Figure 3-2 FMAC Pipeline

An example of the FMAC pipeline is illustrated below.

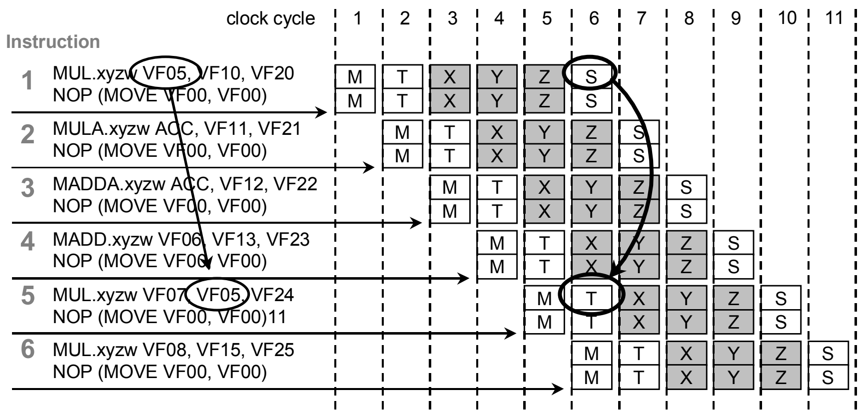


Figure 3-3 FMAC Pipeline Operation Example

Figure 3-3 illustrates a normal operation example of the FMAC pipeline. Calculation results of Instruction 1 are used for Instruction 5, but stalls do not occur because other instructions are being executed between them.

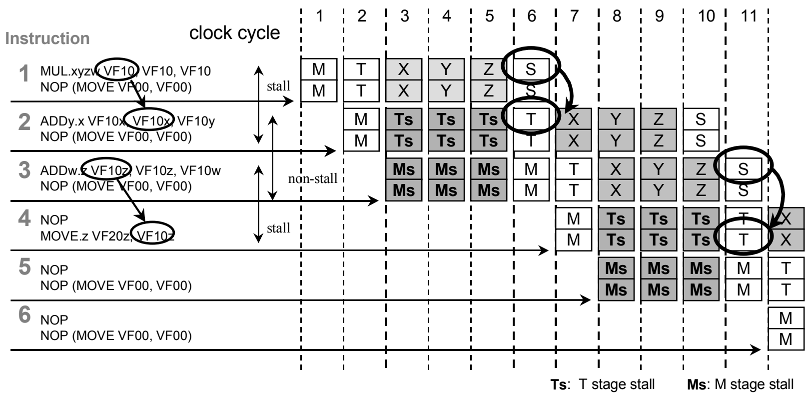


Figure 3-4 Stalls due to FMAC Pipeline Data Hazards

Figure 3-4 shows an example of FMAC pipeline stall. VF10x of Instruction 1 output is used for Instruction 2. Therefore, execution of Instruction 2 is delayed until Instruction 1 reaches the S stage.
Between Instruction 3 and Instruction 4, data hazards occur to VF10z.
Stalls do not occur between Instruction 2 and Instruction 3. The output VF10x of Instruction 2 and the input VF10z and VF10w of Instruction 3 are in the same register, but data hazards are not generated since they are in different fields. (Data hazards are generated in the macroinstruction.)
Stalls due to data dependency do not occur between the Upper instruction and the Lower instruction in the same instruction, though this is not shown in this example. For the relationship with COP2 instructions, see "5.4.4. Operation when Transferring Data with EE Core".

3.4.5. FDIV Pipeline

The following figures illustrate the pipelines related to the floating-point division unit.

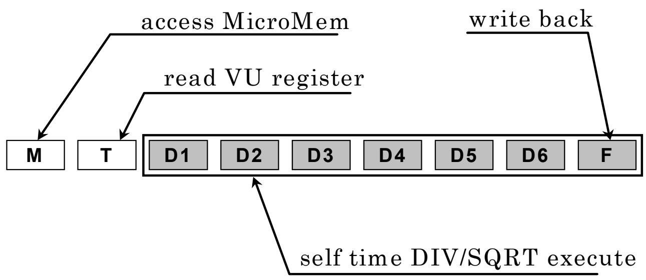


Figure 3-5 FDIV Pipeline (DIV / SQRT)

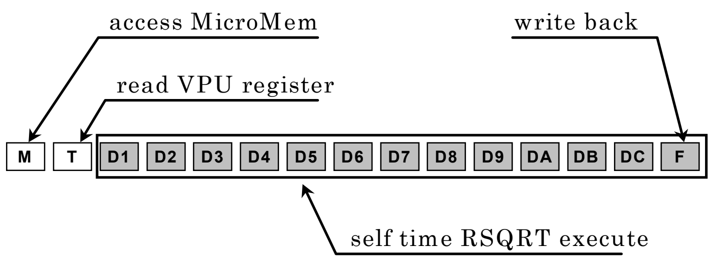


Figure 3-6 FDIV Pipeline (RSQRT)

The next DIV/SQRT/RSQRT instruction stalls with the generation of resource hazards during execution of D1- D6 stage of DIV/SQRT instruction and D1- DC stage of RSQRT instruction.

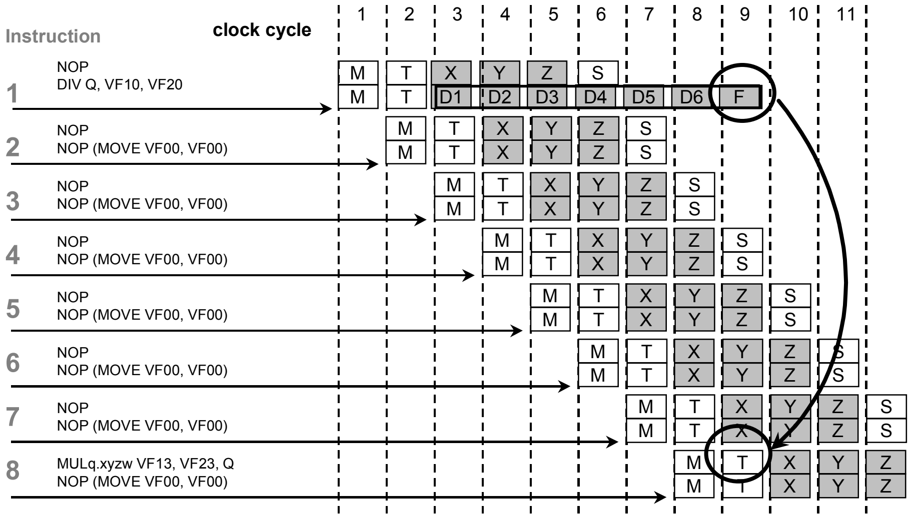


Figure 3-7 FDIV Pipeline Operation Example

Figure 3-7 illustrates a normal operation example of the FDIV pipeline. The DIV calculation results can be received and calculation can be performed by inserting six or more other instructions between the DIV instruction (Instruction 1) and the MULq instruction (Instruction 8) which uses the Q register, the results of the DIV instruction, after the DIV instruction (Instruction 1).

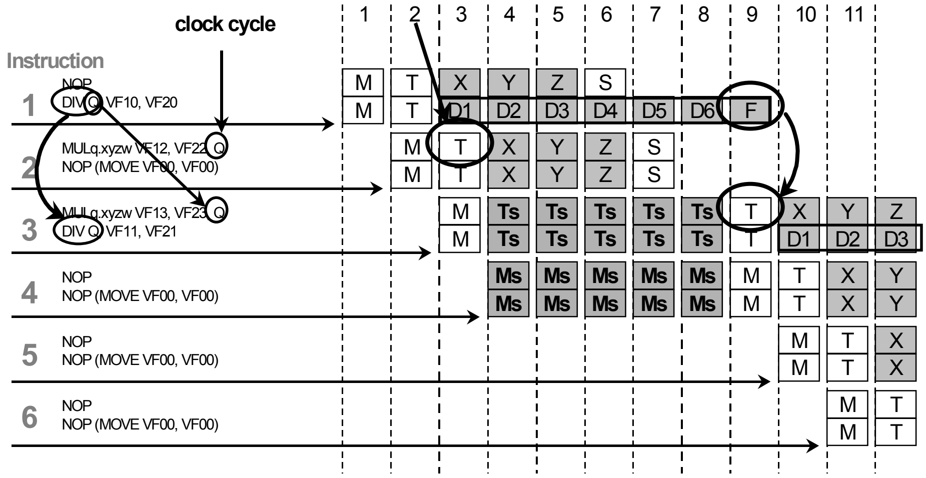


Figure 3-8 FDIV Pipeline Continuous Execution Example

Figure 3-8 shows an example of FDIV pipeline continuous execution. The next DIV instruction (Instruction 3) is started during the DIV instruction (Instruction 1) execution, and Instruction 3 stalls until the D6 stage of Instruction 1 ends.

Although Instruction 2 is a MULq instruction, data dependency is not checked regarding the Q register, so the Q register value previously obtained is used here, not the calculation results of Instruction 1.

The MULq of Instruction 3 uses the calculation results of Instruction 1, which was written to the Q register at the F stage, due to stalls.

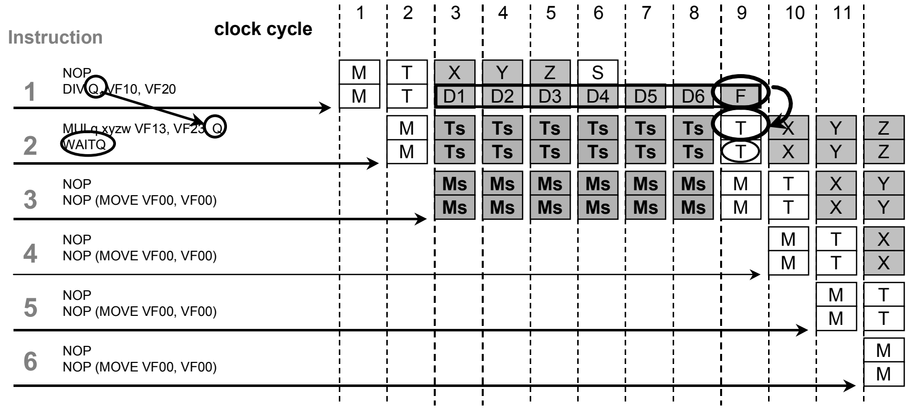


Figure 3-9 WAITQ Instruction Operation Example

Figure 3-9 illustrates an example of synchronization using the WAITQ instruction. Due to the WAITQ instruction of Instruction 2, subsequent instructions stall until the output of the DIV instruction (Instruction 1) is fixed. The results of Instruction 1 can be used from the Upper instruction of Instruction 2.

3.4.6. EFU Pipeline

Figure 3-10 illustrates the EFU pipeline. During execution of N1 - Nn-1 stages, the next elementary function calculation instruction stalls with the generation of resource hazards. Unlike the FDIV pipeline, the resource hazards are not generated during execution of the Nn stage, the final stage to be executed.

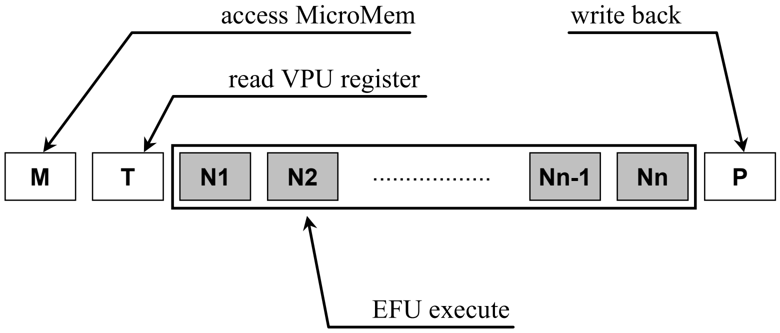


Figure 3-10 EFU Pipeline

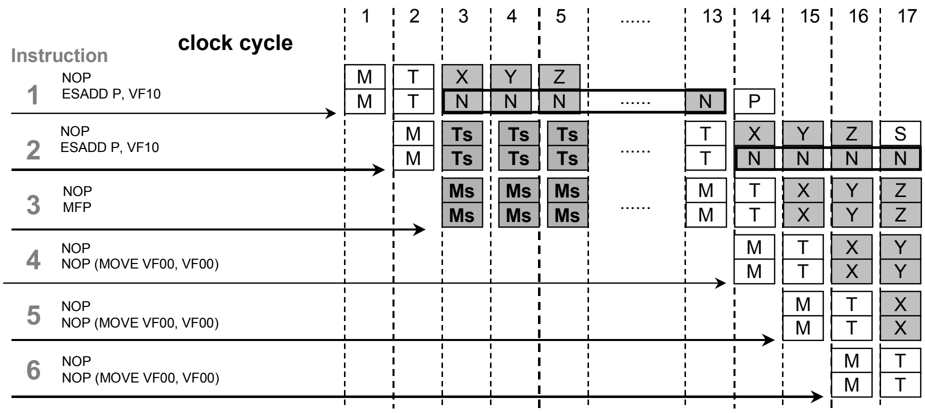


Figure 3-11 EFU Pipeline Continuous Execution Example

Figure 3-11 illustrates a pipeline operation example in which the EFU instruction is continuously executed. Since the ESADD instruction (Latency 11, Throughput 10) is executed by Instruction 1 and a new ESADD instruction (Instruction 2) is executed before the end of the current ESADD instruction, stalls are generated at the T stage of Instruction 2. Unlike the FDIV pipeline, the stall is cleared at the Nn stage (one stage before the P stage).

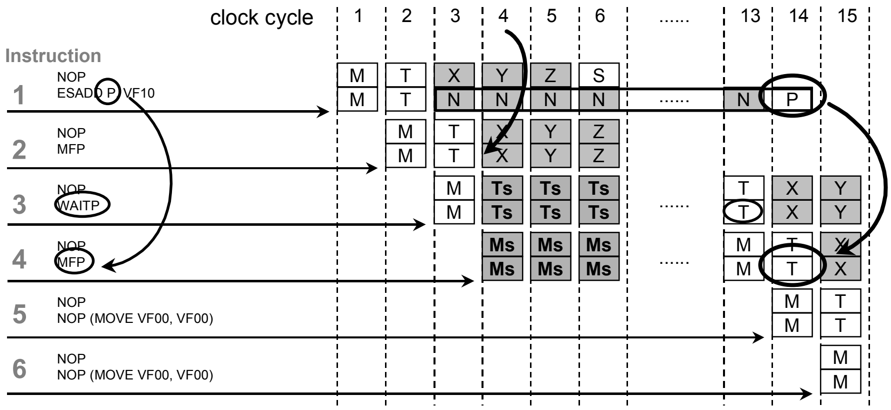


Figure 3-12 WAITP Instruction Operation Example

Figure 3-12 illustrates a synchronization example using the WAITP instruction. Due to the WAITP instruction of Instruction 3, a stall occurs and continues until the end of the ESADD instruction of Instruction 1 and is cleared at the Nn stage, then the succeeding instruction is executed.
Similar to the Q register, the data dependency is not checked in the P register. In the MFP instruction of Instruction 2, the P register values gained prior to Instruction 1 are used. In the MFP instruction of Instruction 4, the P register values gained from Instruction 1 are used as a result of the stall.

3.4.7. IALU Pipeline

Figure 3-13 illustrates the IALU pipeline, which performs Integer calculation.

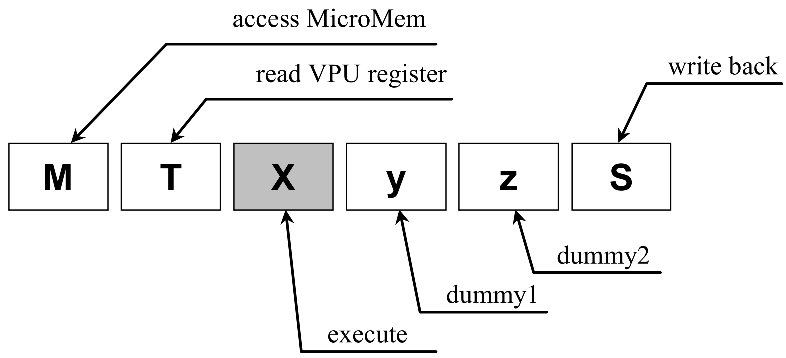


Figure 3-13 Actual IALU Pipeline

IALU execution ends in one cycle, but the y stage and z stage exist as dummy stages in order to adjust the timing with the FMAC pipeline. Although the results are actually stored in the integer register at the S stage, there is no latency in effect for the dummy stages since the results are bypassed from the X/y/z/S stage to the T stage.
The latency for the dummy stages appears only when the microinstruction calculation results are transferred after the Integer register has been written at the S stage by the CFC2 instruction (a coprocessor transfer macroinstruction).

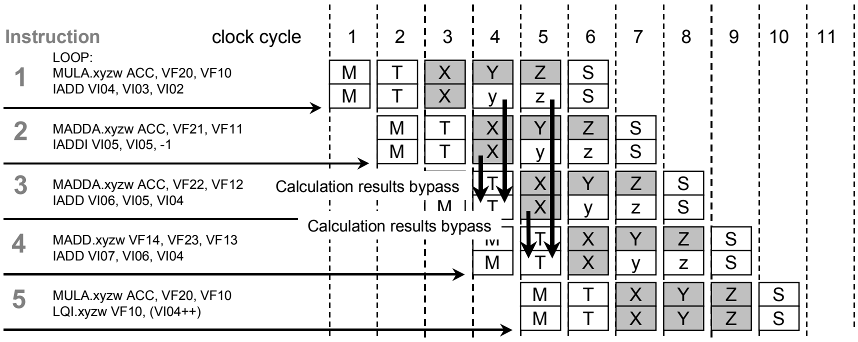


Figure 3-14 IALU Pipeline Bypass Example

Figure 3-14 illustrates an operation example of the IALU pipeline. The IADD instruction of Instruction 3 uses the results of Instruction 1 and Instruction 2, so bypasses are generated between the y stage of Instruction 1 and the T stage of Instruction 3, and between the X stage of Instruction 2 and the T stage of Instruction 3. Similarly, bypasses are generated between the z stage of Instruction 1 and the T stage of Instruction 4, and between the X stage of Instruction 3 and the T stage of Instruction 4.

3.4.8. Conditional Branching and Pipeline

Figure 3-15 illustrates an example of pipeline operation, which accompanies conditional branching.

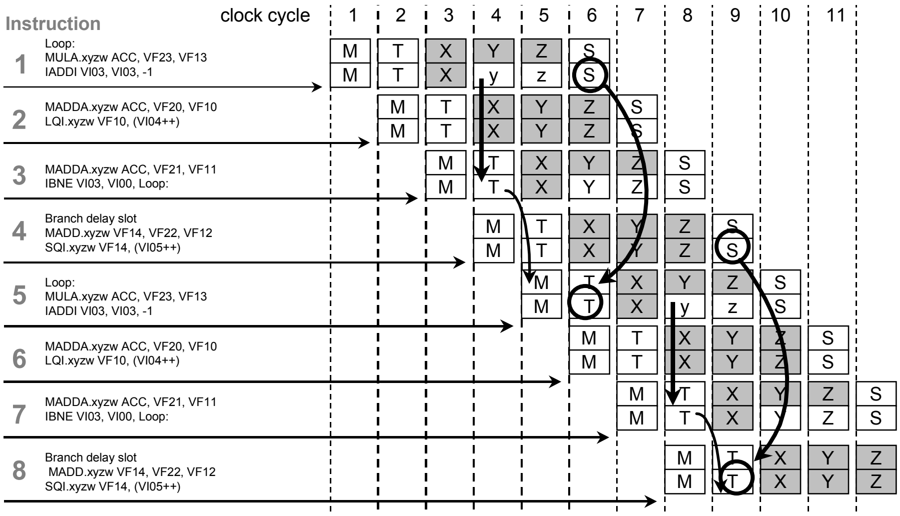


Figure 3-15 Operation Example of Integer Calculation Branching Instruction

In Figure 3-15, a loop has been created between Instruction 1 and Instruction 4. A conditional branch is performed in Instruction 3 according to the results of the Lower instruction of Instruction 1, and it branches to the beginning of the loop (Instruction 5) after the one-instruction branch delay slot (Instruction 4).

As mentioned above, a one-instruction slot is necessary between a branch condition setting instruction and a condition branching instruction. Flag check instructions (FCAND, FCEQ, FCGET, FCOR, FMAND, FMEQ, FMOR, FSAND, FSEQ, and FSOR) are an exception, and a conditional branch instruction can be placed immediately after them.

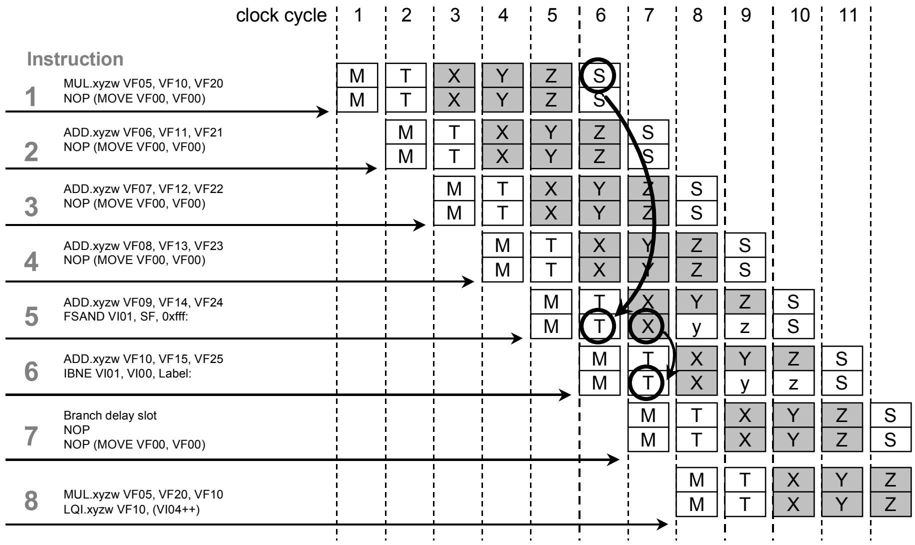


Figure 3-16 Example of Floating-Point Calculation Branch Instruction (1)

Figure 3-16 shows an example of branching according to floating-point calculation results. A conditional branch is performed according to the results of Instruction 1, and the calculation results are written to the status flag at the S stage. The logical AND of the status flag and the immediate value is written to VI01 in Instruction 5, after inserting Instructions 2 to 4. Then, the conditional branch is performed according to the value of VI01 in Instruction 6.

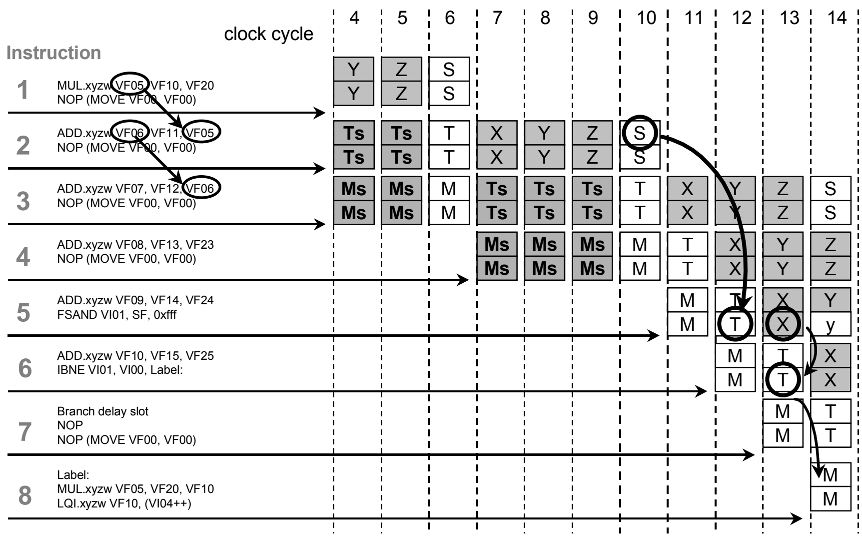


Figure 3-17 Example of Floating-Point Calculation Branch Instruction (2)

Figure 3-17 shows an example in which stalls are generated due to data hazards. As a result, the status flag referred to by Instruction 5 shows the calculation results of Instruction 2.

As mentioned above, when reading the status flag, it is necessary to pay attention to the timing.

3.4.9. XGKICK Pipeline

The stage structure of the XGKICK pipeline that activates GIF transfer via PATH1 is basically the same as that of the FMAC pipeline, but the XGKICK instruction performs a special pipeline operation that makes the subsequent instructions stall on the T stage when continuously executed.

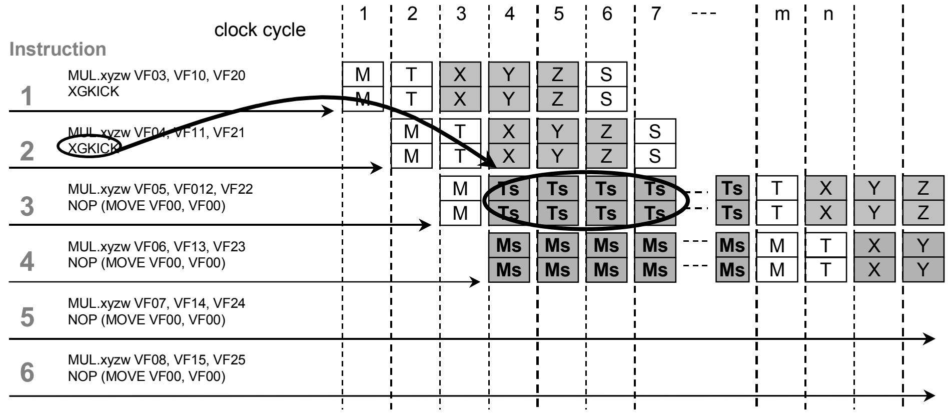


Figure 3-18 XGKICK Pipeline Operation

Figure 3-18 shows an XGKICK pipeline operation example. The XGKICK in Instruction 1 executes without stalling. However, at the XGKICK in Instruction 2, since the transfer via PATH1 activated in Instruction 1 is in process, the pipeline stalls until the PATH1 transfer caused by the preceding XGKICK instruction ends. At this time, not Instruction 2, but the following Instruction 3 is delayed. Note that the Upper instruction of Instruction 2 is executed without stalling.

3.5. Micro Subroutine Execution

3.5.1. How to Execute a Micro Subroutine

There are three ways to execute a micro subroutine:

Macroinstruction VCALLMS / VCALLMSR instruction	Executable in VU0
Write the execution address to the control register	Executable in VU1
MSCAL/MSCALF of VIFcode	Executable in VU0/VU1

Operation is indeterminate if VCALLMS/VCALLMSR instruction from the EE Core and start-up from the VIF are specified concurrently.

3.5.2. How to Terminate a Micro Subroutine

There are three ways to terminate a micro subroutine:

By a microinstruction that sets the E bit to 1.
By a microinstruction that sets the T bit or D bit to 1.
By a Force Break from an external source.

A micro subroutine is normally terminated by a microinstruction that sets the E bit to 1. Other termination methods are only for debugging purposes. See "**7.3. Micro Subroutine Debugging**".

3.5.3. Operation of Execution and Termination

Figure 3-19 shows an example of executing a micro subroutine.

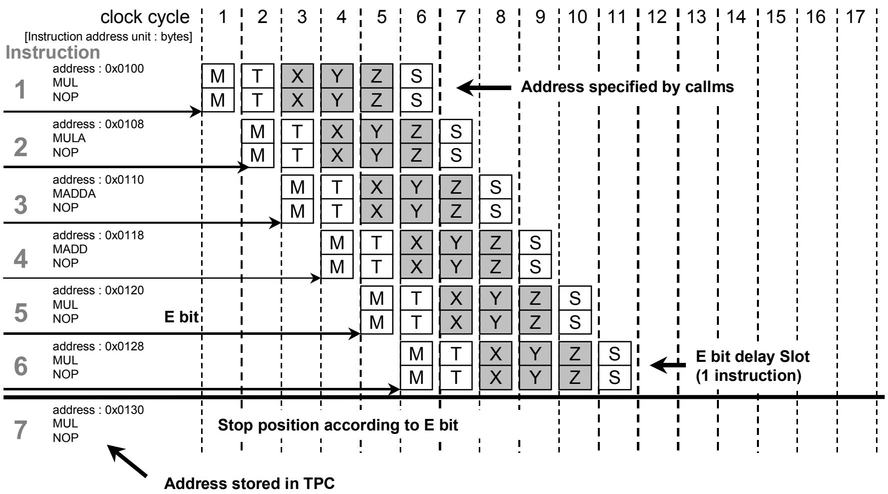


Figure 3-19 Execution by callms and Termination by the E bit

In Figure 3-19, micro subroutine execution starts from Instruction 1. The E bit, which indicates the end of the micro subroutine, is set in Instruction 5. There is a one-instruction E-bit delay slot, in which Instruction 6 is executed; then the micro subroutine stops execution and returns to macro mode. The address of Instruction 7 is stored in the termination position program counter (TPC).

The following kinds of instructions cannot be placed in the E bit delay slot:

- Branch instructions
- Instructions that synchronize to external units, such as XTOP and XITOP
- XGKICK
- VU Mem load/store instruction
- Microinstructions that set the E bit to 1

If a micro subroutine is terminated during the execution of the FDIV or EFU instruction, the FDIV or EFU process is continued, and the results are stored in the P or Q register at a given latency.

3.6. Other Functions

3.6.1. Data Transfer with VU Mem/Micro Mem

VU Mem and Micro Mem are I/O-mapped to the main memory of the EE Core. When the VPU is not operating, these memory locations are accessible directly from the EE Core.
The address map is shown in the following table.

Memory	Address
MicroMem0	0x1100_0000 - 0x1100_0ff0
VUMem0	0x1100_4000 - 0x1100_4ff0
MicroMem1	0x1100_8000 - 0x1100_bff0
VUMem1	0x1100_c000 - 0x1100_fff0

3.6.2. Debug Support Function

Execution of a micro subroutine can be suspended for debugging by setting the D bit in the operation code field to 1. For further information, see "**7.3. Micro Subroutine Debugging**".

(This page is left blank intentionally)

4. Micro Mode Instruction Reference

4.1. Micro Mode Instruction Set

4.1.1. Types of Upper Instruction

There are four types of Upper instructions that primarily execute floating-point calculations:

UpperOP field type 0

Specifies three registers (VF[fs],VF[ft],VF[fd]) and a broadcast field (e.g. ADD$_{bc}$ instruction), and performs scalar calculations as follows:

Example: `ADDx.xyzw VF10xyzw, VF20xyzw, VF30x`

Operation:
VF10x = VF20x + VF30x
VF10y = VF20y + VF30x
VF10z = VF20z + VF30x
VF10w = VF20w + VF30x

Upper 32-bit word: UpperOP field type 0

63 62 61 60 59 58 57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38	37 36 35 34	33 32
I E M D T - -	dest	ft reg	fs reg	fd reg	OPCODE	bc
- - - - - 0 0	----	-----	-----	-----	----	--
1 1 1 1 1 1 1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

UpperOP field type 1

Specifies three registers (VF[fs],VF[ft],VF[fd]), e.g. ADD instruction, and performs vector calculations as follows:

Example: `ADD.xyzw VF10xyzw, VF20xyzw, VF30xyzw`

Operation:
VF10x = VF20x + VF30x
VF10y = VF20y + VF30y
VF10z = VF20z + VF30z
VF10w = VF20w + VF30w

Upper 32-bit word: UpperOP field type 1

63 62 61 60 59 58 57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38	37 36 35 34 33 32
I E M D T - -	dest	ft reg	fs reg	fd reg	OPCODE
- - - - - 0 0	----	-----	-----	-----	------
1 1 1 1 1 1 1	4 bits	5 bits	5 bits	5 bits	6 bits

UpperOP field type 2

Specifies two registers (VF[fs],VF[ft]) and a broadcast field, e.g. ADDA$_{bc}$ instruction, and performs scalar calculations as follows:

Example: `ADDAx.xyzw ACCxyzw, VF20xyzw, VF30x`

Operation:
ACCx = VF20x + VF30x
ACCy = VF20y + VF30x
ACCz = VF20z + VF30x
ACCw = VF20w + VF30x

Upper 32-bit word: UpperOP field type 2

63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32

I	E	M	D	T	-	-	dest	ft reg	fs reg	OPCODE	bc
-	-	-	-	-	0	0	----	-----	-----	----- 1111	--
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	9 bits	2 bits

UpperOP field type 3

Specifies two registers (VF[fs],VF[ft]), e.g. ADDA instruction. Performs vector calculations as follows:

Example: `ADDA.xyzw ACCxyzw, VF20xyzw, VF30xyzw`

Operation:
ACCx = VF20x + VF30x
ACCy = VF20y + VF30y
ACCz = VF20z + VF30z
ACCw = VF20w + VF30w

Upper 32-bit word: UpperOP field type 3

63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32

I	E	M	D	T	-	-	dest	ft reg	fs reg	OPCODE
-	-	-	-	-	0	0	----	-----	-----	----- 1111 --
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits

4.1.2. Types of Lower Instructions

There are 7 types of Lower instructions:

LowerOP field type 1

Specifies 3 registers, e.g. IADD instruction.

Lower 32-bit word: LowerOP field type 1

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

Lower OP.	dest	ft reg	fs reg	fd reg	OPCODE
1000000	----	-----	-----	-----	------
7 bits	4 bits	5 bits	5 bits	5 bits	6 bits

LowerOP field type 3

Specifies up to two registers and a dest field, e.g. MOVE instruction.

Example: `MOVE.xyzw VF10xyzw, VF20xyzw`

Operation:
VF10x = VF20x
VF10y = VF20y
VF10z = VF20z
VF10w = VF20w

Lower 32-bit word: LowerOP field type 3

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

Lower OP.	dest	ft reg	fs reg	OPCODE
1000000	----	-----	-----	----- 1111 --
7 bits	4 bits	5 bits	5 bits	11 bits

LowerOP field type 4

Specifies 2 floating-point registers with a specific field for each, e.g. DIV instruction.

```
Example: DIV Q, VF10x, VF20y
```

Operation: Q = VF10x ÷ VF20y

Lower 32-bit word: LowerOP field type 4

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

Lower OP.	ftf	fsf	ft reg	fs reg	OPCODE
1000000	--	--	-----	-----	----- 1111 --
7 bits	2 bits	2 bits	5 bits	5 bits	11 bits

LowerOP field type 5

Specifies 2 registers and a 5-bit immediate value, e.g. IADDI instruction.

Lower 32-bit word: LowerOP field type 5

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

Lower OP.	dest	it reg	is reg	Imm5	OPCODE
1000000	0000	-----	-----	-----	------
7 bits	4 bits	5 bits	5 bits	5 bits	6 bits

LowerOP field type 7

Specifies 2 registers and an 11-bit immediate value, e.g. ILW instruction.

Lower 32-bit word: LowerOP field type 7

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

Lower OP.	dest	it reg	fs reg	Imm11
0------	----	-----	-----	-----------
7 bits	4 bits	5 bits	5 bits	11 bits

LowerOP field type 8

Specifies 2 registers and a 15-bit immediate value.

Lower 32-bit word: LowerOP field type 8

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

Lower OP.	Imm15	it reg	fs reg	Imm15
0------	----	-----	-----	-----------
7 bits	4 bits	5 bits	5 bits	11 bits

LowerOP field type 9

Specifies a 24-bit immediate value, e.g. FCAND instruction.

Lower 32-bit word: LowerOP field type 9

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

Lower OP.	-	Imm24
0------	0	------------------------
7 bits	1	24 bits

4.1.3. Operation Fields for Micro Instructions

Various operation fields in operation codes are explained in this section.

dest field (Upper/Lower)

Upper 32-bit word: UpperOP field type 0

63	62	61	60	59	58	57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38	37 36 35 34	33 32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	OPCODE	bc
-	-	-	-	-	0	0	----	-----	-----	-----	----	--
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Lower 32-bit word: LowerOP field type 1

31 30 29 28 27 26 25	24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 09 08 07 06	05 04 03 02 01 00
Lower OP.	dest	ft reg	fs reg	fd reg	OPCODE
1000000	----	-----	-----	-----	------
7 bits	4 bits	5 bits	5 bits	5 bits	6 bits

The dest field specifies the FMAC units to be operated in parallel; that is, either the x, y, z or w field of the 128-bit data to be operated on.
The dest field is 4 bits: bits 56 through 53 for Upper instructions and bits 24 through 21 for Lower instructions. Each of the 4 bits can be specified independently; when the bit is set to 1, the corresponding FMAC unit /field becomes effective.

Bit		Corresponding FMAC /Field
Upper	Lower	
56	24	x
55	23	y
54	22	z
53	21	w

bc field (Upper)

Upper 32-bit word: UpperOP field type 0

63	62	61	60	59	58	57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38	37 36 35 34	33 32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	OPCODE	bc
-	-	-	-	-	0	0	----	-----	-----	-----	----	--
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Lower 32-bit word: LowerOP field type 1

31 30 29 28 27 26 25	24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 09 08 07 06	05 04 03 02 01 00
Lower OP.	dest	ft reg	fs reg	fd reg	OPCODE
1000000	----	-----	-----	-----	------
7 bits	4 bits	5 bits	5 bits	5 bits	6 bits

The bc field is bits 33 and 32, and specifies the broadcast field as below.

Specified value of bc field	Broadcast field
00	x
01	y
10	z
11	w

fsf/ftf field

Upper 32-bit word: UpperOP field type 0

63 62 61 60 59 58 57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38	37 36 35 34	33 32
I E M D T - -	dest	ft reg	fs reg	fd reg	OPCODE	bc
- - - - - 0 0	----	-----	-----	-----	----	--
1 1 1 1 1 1 1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Lower 32-bit word: LowerOP field type 4

31 30 29 28 27 26 25	24 23	22 21	20 19 18 17 16	15 14 13 12 11	10 09 08 07 06 05 04 03 02 01 00
Lower OP.	ftf	fsf	ft reg	fs reg	OPCODE
1000000	--	--	-----	-----	----- 1111 --
7 bits	2 bits	2 bits	5 bits	5 bits	11 bits

The combinations of the fsf field with the fs reg field and the ftf field with the ft reg field specify the field to be calculated by the instruction. Bits 22 and 21 of the Lower instruction are used for the fsf field, and bits 24 and 23 are used for the ftf field.

Specified value for fsf/ftf field	Field to be operated
00	x
01	y
10	z
11	w

I bit (Upper)

Upper 32-bit word: UpperOP field type 0

63 62 61 60 59 58 57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38	37 36 35 34	33 32
I E M D T - -	dest	ft reg	fs reg	fd reg	OPCODE	bc
- - - - - 0 0	----	-----	-----	-----	----	--
1 1 1 1 1 1 1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Lower 32-bit word: LowerOP field type 1

31 30 29 28 27 26 25	24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 09 08 07 06	05 04 03 02 01 00
Lower OP.	dest	ft reg	fs reg	fd reg	OPCODE
1000000	----	-----	-----	-----	------
7 bits	4 bits	5 bits	5 bits	5 bits	6 bits

The I bit is specified when loading an immediate value into the I register. When bit 63 of the Upper instruction field is set to 1, the contents of the Lower instruction field are loaded into the I register as a single-precision floating-point immediate value.

E bit (Upper)

Upper 32-bit word: UpperOP field type 0

63	62	61	60	59	58	57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38	37 36 35 34	33 32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	OPCODE	bc
-	1	-	-	-	0	0	----	-----	-----	-----	----	--
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Lower 32-bit word: LowerOP field type 1

31 30 29 28 27 26 25	24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 09 08 07 06	05 04 03 02 01 00
Lower OP.	dest	ft reg	fs reg	fd reg	OPCODE
1000000	----	-----	-----	-----	------
7 bits	4 bits	5 bits	5 bits	5 bits	6 bits

The E bit is bit 62 of the Upper instruction field; it is used when designating termination of a micro subroutine. When the E bit is set to 1, the VU terminates execution of the micro subroutine after the next instruction and returns to macro mode.

M bit (Upper)

Upper 32-bit word: UpperOP field type 0

63	62	61	60	59	58	57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38	37 36 35 34	33 32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	OPCODE	bc
-	-	-	-	-	0	0	----	-----	-----	-----	----	--
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Lower 32-bit word: LowerOP field type 1

31 30 29 28 27 26 25	24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 09 08 07 06	05 04 03 02 01 00
Lower OP.	dest	ft reg	fs reg	fd reg	OPCODE
1000000	----	-----	-----	-----	------
7 bits	4 bits	5 bits	5 bits	5 bits	6 bits

The M bit is bit 61 of the Upper instruction field; it specifies QMTC2 / CTC2 instruction interlock. The QMTC2 / CTC2 instruction is executed without interlocking when the M bit is set to 1. Refer to "5.4. Macro Mode Pipeline".

D bit (Upper)

Upper 32-bit word: UpperOP field type 0

63	62	61	60	59	58	57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38	37 36 35 34	33 32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	OPCODE	bc
-	-	-	-	-	0	0	----	-----	-----	-----	----	--
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Lower 32-bit word: LowerOP field type 1

31 30 29 28 27 26 25	24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 09 08 07 06	05 04 03 02 01 00
Lower OP.	dest	ft reg	fs reg	fd reg	OPCODE
1000000	----	-----	-----	-----	------
7 bits	4 bits	5 bits	5 bits	5 bits	6 bits

The D bit is bit 60 of the Upper instruction; it specifies a debug break instruction. When the D bit is set to 1 and the instruction is executed, the VU is halted and an interrupt signal is sent to the host processor. The interrupt can be enabled/disabled by the DE bit (D bit Enable) of the control register FBRST.

T bit (Upper)

Upper 32-bit word: UpperOP field type 0

63	62	61	60	59	58	57	56-53	52-48	47-43	42-38	37-34	33-32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	OPCODE	bc
-	-	-	-	-	0	0	----	-----	-----	-----	----	--
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Lower 32-bit word: LowerOP field type 1

31-25	24-21	20-16	15-11	10-06	05-00
Lower OP.	dest	ft reg	fs reg	fd reg	OPCODE
1000000	----	-----	-----	-----	------
7 bits	4 bits	5 bits	5 bits	5 bits	6 bits

The T bit is bit 59 of the Upper instruction; it specifies debug halt instruction. In the same manner as the D bit, when the T bit is set to 1 and the instruction is executed, the VU is halted, and an interrupt signal is sent to the host processor. The interrupt can be enabled/disabled by the TE bit (T bit Enable) of the control register FBRST.

4.2. Upper Instruction Reference

This section describes the function, operation code, mnemonic, operation, flag changes, and throughput/latency of Upper instructions. They are listed in alphabetical order in mnemonic form. The descriptions also include examples, programming notes, and reference information.

ABS : Absolute Value

Calculates the absolute value of VF[fs] and stores the result in VF[ft].

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38	37 36 35 34	33 32
I	E	M	D	T	-	-	dest	ft reg	fs reg	ABS		
-	-	-	-	-	0	0	----	-----	-----	00111	1111	01
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

ABS.dest **VF[**ft**]**dest, **VF[**fs**]**dest

Operation

if (x ⊆ dest) then VF[ft]x = |VF[fs]x|
if (y ⊆ dest) then VF[ft]y = |VF[fs]y|
if (z ⊆ dest) then VF[ft]z = |VF[fs]z|
if (w ⊆ dest) then VF[ft]w = |VF[fs]w|

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/latency

1 / 4

Example

```
ABS.xyzw VF10xyzw, VF20xyzw
```

VF10x = |VF20x|
VF10y = |VF20y|
VF10z = |VF20z|
VF10w = |VF20w|

ADD : Add

Calculates the sum of VF[fs] and VF[ft], and stores the result in VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 1

63	62	61	60	59	58	57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38	37 36 35 34 33 32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	ADD
-	-	-	-	-	0	0	----	-----	-----	-----	101000
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

ADD.dest **VF[**fd**]**dest, **VF[**fs**]**dest, **VF[**ft**]**dest

Operation

if (x ⊆ dest) then VF[fd]x = VF[fs]x + VF[ft]x
if (y ⊆ dest) then VF[fd]y = VF[fs]y + VF[ft]y
if (z ⊆ dest) then VF[fd]z = VF[fs]z + VF[ft]z
if (w ⊆ dest) then VF[fd]w = VF[fs]w + VF[ft]w

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
ADD.xyzw VF10xyzw, VF20xyzw, VF30xyzw
```

VF10x = VF20x + VF30x
VF10y = VF20y + VF30y
VF10z = VF20z + VF30z
VF10w = VF20w + VF30w

ADDi : Add to I Register

Adds each field of VF[fs] and the I register, and stores the sum in the corresponding field of VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 1

63	62	61	60	59	58	57	56–53	52–48	47–43	42–38	37–32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	ADDi
-	-	-	-	-	0	0	----	00000	-----	-----	100010
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

ADDi.dest **VF[**fd**]**dest, **VF[**fs**]**dest, **I**

Operation

if (x ⊆ dest) then VF[fd]x = VF[fs]x + I
if (y ⊆ dest) then VF[fd]y = VF[fs]y + I
if (z ⊆ dest) then VF[fd]z = VF[fs]z + I
if (w ⊆ dest) then VF[fd]w = VF[fs]w + I

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
ADDi.xyzw VF10xyzw, VF20xyzw, I
```

VF10x = VF20x + I
VF10y = VF20y + I
VF10z = VF20z + I
VF10w = VF20w + I

ADDq : Add to Q Register

Adds each field of VF[fs] and the Q register, and stores the sum in the corresponding field of VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 1

63	62	61	60	59	58	57	56-53	52-48	47-43	42-38	37-32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	ADDq
-	-	-	-	-	0	0	----	00000	-----	-----	100000
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

ADDq.dest **VF[**fd**]**dest, **VF[**fs**]**dest, **Q**

Operation

if (x ⊆ dest) then VF[fd]x = VF[fs]x + Q
if (y ⊆ dest) then VF[fd]y = VF[fs]y + Q
if (z ⊆ dest) then VF[fd]z = VF[fs]z + Q
if (w ⊆ dest) then VF[fd]w = VF[fs]w + Q

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
ADDq.xyzw VF10xyzw, VF20xyzw, Q
```

VF10x = VF20x + Q
VF10y = VF20y + Q
VF10z = VF20z + Q
VF10w = VF20w + Q

ADDbc : Broadcast Add

Calculates the sum of each field of VF[fs] and the specified field of VF[ft], and stores the sum in the corresponding field of VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 0

63	62	61	60	59	58	57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38	37 36 35 34	33 32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	ADD?	bc
-	-	-	-	-	0	0	----	-----	-----	-----	0000	--
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Mnemonic

ADDbc.dest **VF[**fd**]**dest, **VF[**fs**]**dest, **VF[**ft**]**bc

Operation

if (x ⊆ dest) then VF[fd]x = VF[fs]x + VF[ft]$_{bc}$
if (y ⊆ dest) then VF[fd]y = VF[fs]y + VF[ft]$_{bc}$
if (z ⊆ dest) then VF[fd]z = VF[fs]z + VF[ft]$_{bc}$
if (w ⊆ dest) then VF[fd]w = VF[fs]w + VF[ft]$_{bc}$

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
ADDx.xyzw VF10xyzw, VF20xyzw, VF30x
```

VF10x = VF20x + VF30x
VF10y = VF20y + VF30x
VF10z = VF20z + VF30x
VF10w = VF20w + VF30x

ADDA : Add; to Accumulator

Adds VF[fs] and VF[ft], and stores the sum in ACC.

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56–53	52–48	47–43	42–32		
I	E	M	D	T	-	-	dest	ft reg	fs reg	ADDA		
-	-	-	-	-	0	0	----	-----	-----	01010	1111	00
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

ADDA.dest **ACC**dest, **VF[**fs**]**dest, **VF[**ft**]**dest

Operation

if (x ⊆ dest) then ACCx = VF[fs]x + VF[ft]x
if (y ⊆ dest) then ACCy = VF[fs]y + VF[ft]y
if (z ⊆ dest) then ACCz = VF[fs]z + VF[ft]z
if (w ⊆ dest) then ACCw = VF[fs]w + VF[ft]w

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
ADDA.xyzw ACCxyzw, VF20xyzw, VF30xyzw
```

ACCx = VF20x + VF30x
ACCy = VF20y + VF30y
ACCz = VF20z + VF30z
ACCw = VF20w + VF30w

ADDAi : Add I Register; to Accumulator

Adds each field of VF[fs] and the I register, and stores the sum in the corresponding field of ACC.

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56–53	52–48	47–43	42–32		
I	E	M	D	T	-	-	dest	ft reg	fs reg	ADDAi		
-	-	-	-	-	0	0	----	00000	-----	01000	1111	10
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

ADDAi.dest **ACC**dest, **VF[**fs**]**dest, **I**

Operation

if (x ⊆ dest) then ACCx = VF[fs]x + I
if (y ⊆ dest) then ACCy = VF[fs]y + I
if (z ⊆ dest) then ACCz = VF[fs]z + I
if (w ⊆ dest) then ACCw = VF[fs]w + I

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
ADDAi.xyzw ACCxyzw, VF20xyzw, I
```

ACCx = VF20x + I
ACCy = VF20y + I
ACCz = VF20z + I
ACCw = VF20w + I

ADDAq : Add Q Register; to Accumulator

Adds each field of VF[fs] and the Q register, and stores the sum in the corresponding field of ACC.

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56-53	52-48	47-43	42-32		
I	E	M	D	T	-	-	dest	ft reg	fs reg	ADDAq		
-	-	-	-	-	0	0	----	00000	-----	01000	1111	00
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

ADDAq.dest **ACC**dest, **VF[**fs**]**dest, **Q**

Operation

if (x ⊆ dest) then ACCx = VF[fs]x + Q
if (y ⊆ dest) then ACCy = VF[fs]y + Q
if (z ⊆ dest) then ACCz = VF[fs]z + Q
if (w ⊆ dest) then ACCw = VF[fs]w + Q

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
ADDAq.xyzw ACCxyzw, VF20xyzw, Q
```

ACCx = VF20x + Q
ACCy = VF20y + Q
ACCz = VF20z + Q
ACCw = VF20w + Q

ADDAbc : Broadcast Add; to Accumulator

Adds each field of VF[fs] and the specified field of VF[ft], and stores the sum in ACC.

Operation Code

Upper 32-bit word: UpperOP field type 2

63	62	61	60	59	58	57	56-53	52-48	47-43	42-34		33-32
I	E	M	D	T	-	-	dest	ft reg	fs reg	ADDA?		bc
-	-	-	-	-	0	0	----	-----	-----	00000	1111	--
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	9 bits		2 bits

Mnemonic

ADDA$_{bc}$.$_{dest}$ **ACC**$_{dest}$, **VF[**fs**]**$_{dest}$, **VF[**ft**]**$_{bc}$

Operation

if (x ⊆ dest) then ACCx = VF[fs]x + VF[ft]$_{bc}$
if (y ⊆ dest) then ACCy = VF[fs]y + VF[ft]$_{bc}$
if (z ⊆ dest) then ACCz = VF[fs]z + VF[ft]$_{bc}$
if (w ⊆ dest) then ACCw = VF[fs]w + VF[ft]$_{bc}$

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
ADDAx.xyzw ACCxyzw, VF20xyzw, VF30x
```

ACCx = VF20x + VF30x
ACCy = VF20y + VF30x
ACCz = VF20z + VF30x
ACCw = VF20w + VF30x

CLIP : Clipping Judgment

Performs clipping judgment by the x,y,z field of VF[fs] and the w field of VF[ft] and sets the clipping flag (CF) to the result.

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56-53	52-48	47-43	42-32		
I	E	M	D	T	-	-	dest	ft reg	fs reg	CLIPw		
-	-	-	-	-	0	0	1110	-----	-----	00111	1111	11
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

CLIPw.xyz VF[fs**]xyz, VF[**ft**]w**

Operation

CF = CF << 6
if (VF[fs]x > +|VF[ft]w|) then {set +x flag}
if (VF[fs]x < -|VF[ft]w|) then {set -x flag}
if (VF[fs]y > +|VF[ft]w|) then {set +y flag}
if (VF[fs]y < -|VF[ft]w|) then {set -y flag}
if (VF[fs]z > +|VF[ft]w|) then {set +z flag}
if (VF[fs]z < -|VF[ft]w|) then {set -z flag}

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	X

Throughput/Latency

1 / 4

Example

```
CLIPw.xyz VF10xyz, VF10w
```

Under the following condition:

VF10x > +|VF10w|,
-|VF10w| < VF10y < +|VF10w|,
VF10z < -|VF10w|

Before Execution

23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
3rd previous judgment						2nd previous judgment						Previous judgment						Current judgment					
- z	+ z	- y	+ y	- x	+ x	- z	+ z	- y	+ y	- x	+ x	- z	+ z	- y	+ y	- x	+ x	- z	+ z	- y	+ y	- x	+ x
0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0
6 bits						6 bits						6 bits						6 bits					

↓

After Execution

23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
3rd previous judgment						2nd previous judgment						Previous judgment						Current judgment					
- z	+ z	- y	+ y	- x	+ x	- z	+ z	- y	+ y	- x	+ x	- z	+ z	- y	+ y	- x	+ x	- z	+ z	- y	+ y	- x	+ x
0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	0	0	0	1
6 bits						6 bits						6 bits						6 bits					

Remarks

In order to branch according to the results of clipping judgment, the Lower instructions FCAND/FCEQ/FCGET/FCOR are available.

FTOI0 : Convert to Fixed Point

Converts the value of VF[fs] into a fixed-point number whose fractional part is 0 bit, and stores the result in VF[ft].

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56-53	52-48	47-43	42-32		
I	E	M	D	T	-	-	dest	ft reg	fs reg	FTOI0		
-	-	-	-	-	0	0	----	-----	-----	00101	1111	00
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

FTOI0.dest **VF[**ft**]**dest, **VF[**fs**]**dest

Operation

if (x ⊆ dest) then
  VF[ft]x = float_to_integer0(VF[fs]x)
if (y ⊆ dest) then
  VF[ft]y = float_to_integer0(VF[fs]y)
if (z ⊆ dest) then
  VF[ft]z = float_to_integer0(VF[fs]z)
if (w ⊆ dest) then
  VF[ft]w = float_to_integer0(VF[fs]w)

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
FTOI0.xyzw VF10xyzw, VF20xyzw
```

VF10x = float_to_integer0(VF20x)
VF10y = float_to_integer0(VF20y)
VF10z = float_to_integer0(VF20z)
VF10w = float_to_integer0(VF20w)

Remarks

A few examples are shown in the following table:

x	float_to_integer0(x)
-0.45	0
0.45	0
0.55	0
123.45	123

FTOI4 : Convert to Fixed Point

Converts the value of VF[fs] into a fixed-point number whose fractional part is 4 bits, and stores the result in VF[ft].

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38 37 36 35 34 33 32
I	E	M	D	T	-	-	dest	ft reg	fs reg	FTOI4
-	-	-	-	-	0	0	----	-----	-----	00101 1111 01
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

FTOI4.dest **VF[**ft**]**dest, **VF[**fs**]**dest

Operation

if (x ⊆ dest) then
VF[ft]x = float_to_integer4(VF[fs]x)
if (y ⊆ dest) then
VF[ft]y = float_to_integer4(VF[fs]y)
if (z ⊆ dest) then
VF[ft]z = float_to_integer4(VF[fs]z)
if (w ⊆ dest) then
VF[ft]w = float_to_integer4(VF[fs]w)

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
FTOI4.xyzw VF10xyzw, VF20xyzw
```

VF10x = float_to_integer4(VF20x)
VF10y = float_to_integer4(VF20y)
VF10z = float_to_integer4(VF20z)
VF10w = float_to_integer4(VF20w)

Remarks

A few examples are shown in the following table:

x	float_to_integer4(x)
-0.45	-7
0.45	7
0.55	8
123.45	1975

FTOI12 : Convert to Fixed Point

Converts the value of VF[fs] into a fixed-point number whose fractional part is 12 bits, and stores the result in VF[ft].

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56–53	52–48	47–43	42–38	37–34	33–32
I	E	M	D	T	-	-	dest	ft reg	fs reg	FTOI12		
-	-	-	-	-	0	0	----	-----	-----	00101	1111	10
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

FTOI12.dest **VF[**ft**]**dest, **VF[**fs**]**dest

Operation

if (x ⊆ dest) then
VF[ft]x = float_to_integer12(VF[fs]x)
if (y ⊆ dest) then
VF[ft]y = float_to_integer12(VF[fs]y)
if (z ⊆ dest) then
VF[ft]z = float_to_integer12(VF[fs]z)
if (w ⊆ dest) then
VF[ft]w = float_to_integer12(VF[fs]w)

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
FTOI12.xyzw VF10xyzw, VF20xyzw
```

VF10x = float_to_integer12(VF20x)
VF10y = float_to_integer12(VF20y)
VF10z = float_to_integer12(VF20z)
VF10w = float_to_integer12(VF20w)

Remarks

A few examples are shown in the following table:

x	float_to_integer12(x)
-0.45	-1843
0.45	1843
0.55	2252
123.45	505651

FTOI15 : Convert to Fixed Point

Converts the value of VF[fs] into a fixed-point number whose fractional part is 15 bits, and stores the result in VF[ft].

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56-53	52-48	47-43	42-32		
I	E	M	D	T	-	-	dest	ft reg	fs reg	FTOI15		
-	-	-	-	-	0	0	----	-----	-----	00101	1111	11
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

FTOI15.dest **VF[**ft**]**dest, **VF[**fs**]**dest

Operation

if (x ⊆ dest) then
VF[ft]x = float_to_integer15(VF[fs]x)
if (y ⊆ dest) then
VF[ft]y = float_to_integer15(VF[fs]y)
if (z ⊆ dest) then
VF[ft]z = float_to_integer15(VF[fs]z)
if (w ⊆ dest) then
VF[ft]w = float_to_integer15(VF[fs]w)

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
FTOI15.xyzw VF10xyzw, VF20xyzw
```

VF10x = float_to_integer15(VF20x)
VF10y = float_to_integer15(VF20y)
VF10z = float_to_integer15(VF20z)
VF10w = float_to_integer15(VF20w)

Remarks

A few examples are shown in the following table:

x	float_to_integer15(x)
-0.45	-14745
0.45	14745
0.55	18022
123.45	4045209

ITOF0 : Convert to Floating-Point Number

Considers the value of VF[fs] as a fixed-point number whose fractional part is 0 bit, and converts it into floating point and stores the result in VF[ft].

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56-53	52-48	47-43	42-32
I	E	M	D	T	-	-	dest	ft reg	fs reg	ITOF0
-	-	-	-	-	0	0	----	-----	-----	00100 1111 00
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

ITOF0.dest **VF[**ft**]**dest, **VF[**fs**]**dest

Operation

if (x ⊆ dest) then
VF[ft]x = integer_to_float0(VF[fs]x)
if (y ⊆ dest) then
VF[ft]y = integer_to_float0(VF[fs]y)
if (z ⊆ dest) then
VF[ft]z = integer_to_float0(VF[fs]z)
if (w ⊆ dest) then
VF[ft]w = integer_to_float0(VF[fs]w)

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
ITOF0.xyzw VF10xyzw, VF20xyzw
```

VF10x = integer_to_float0(VF20x)
VF10y = integer_to_float0(VF20y)
VF10z = integer_to_float0(VF20z)
VF10w = integer_to_float0(VF20w)

Remarks

A few examples are shown in the following table:

x	integer_to_float0 (x)
-12	-12.0
1	1.0
123	123.0
1843	1843.0

ITOF4 : Convert to Floating-Point Number

Considers the value of VF[fs] as a fixed-point number whose fractional part is 4 bits, and converts it into floating point and stores the result in VF[ft].

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56-53	52-48	47-43	42-32		
I	E	M	D	T	-	-	dest	ft reg	fs reg	ITOF4		
-	-	-	-	-	0	0	----	-----	-----	00100	1111	01
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

ITOF4.dest **VF[**ft**]**dest, **VF[**fs**]**dest

Operation

if (x ⊆ dest) then
  VF[ft]x = integer_to_float4(VF[fs]x)
if (y ⊆ dest) then
  VF[ft]y = integer_to_float4(VF[fs]y)
if (z ⊆ dest) then
  VF[ft]z = integer_to_float4(VF[fs]z)
if (w ⊆ dest) then
  VF[ft]w = integer_to_float4(VF[fs]w)

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
ITOF4.xyzw VF10xyzw, VF20xyzw
```

VF10x = integer_to_float4(VF20x)
VF10y = integer_to_float4(VF20y)
VF10z = integer_to_float4(VF20z)
VF10w = integer_to_float4(VF20w)

Remarks

A few examples are shown in the following table:

x	integer_to_float4(x)
-12	-0.750000
1	0.062500
123	7.687500
1843	115.187500

ITOF12 : Convert to Floating-Point Number

Considers the value of VF[fs] as a fixed-point number whose fractional part is 12 bits, and converts it into floating point and stores the result in VF[ft].

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56-53	52-48	47-43	42-32
I	E	M	D	T	-	-	dest	ft reg	fs reg	ITOF12
-	-	-	-	-	0	0	----	-----	-----	00100 1111 10
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

ITOF12.dest **VF[**ft**]**dest, **VF[**fs**]**dest

Operation

if (x ⊆ dest) then
VF[ft]x = integer_to_float12(VF[fs]x)
if (y ⊆ dest) then
VF[ft]y = integer_to_float12(VF[fs]y)
if (z ⊆ dest) then
VF[ft]z = integer_to_float12(VF[fs]z)
if (w ⊆ dest) then
VF[ft]w = integer_to_float12(VF[fs]w)

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
ITOF12.xyzw VF10xyzw, VF20xyzw
```

VF10x = integer_to_float12(VF20x)
VF10y = integer_to_float12(VF20y)
VF10z = integer_to_float12(VF20z)
VF10w = integer_to_float12(VF20w)

Remarks

A few examples are shown in the following table:

x	integer_to_float12(x)
-12	-0.002930
1	0.000244
123	0.030029
1843	0.449951

ITOF15 : Convert to Floating-Point Number

Considers the value of VF[fs] as a fixed-point number whose fractional part is 15 bits, and converts it into floating point and stores the result in VF[ft].

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38 37 36 35 34 33 32		
I	E	M	D	T	-	-	dest	ft reg	fs reg	ITOF15		
-	-	-	-	-	0	0	----	-----	-----	00100	1111	11
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

ITOF15.dest **VF[**ft**]**dest**, VF[**fs**]**dest

Operation

if (x ⊆dest) then
  VF[ft]x = integer_to_float15(VF[fs]x)
if (y ⊆ dest) then
  VF[ft]y = integer_to_float15(VF[fs]y)
if (z ⊆ dest) then
  VF[ft]z = integer_to_float15(VF[fs]z)
if (w ⊆ dest) then
  VF[ft]w = integer_to_float15(VF[fs]w)

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
ITOF15.xyzw VF10xyzw, VF20xyzw
```

VF10x = integer_to_float15(VF20x)
VF10y = integer_to_float15(VF20y)
VF10z = integer_to_float15(VF20z)
VF10w = integer_to_float15(VF20w)

Remarks

A few examples are shown in the following table:

x	integer_to_float15(x)
-12	-0.000366
1	0.000031
123	0.003754
1843	0.056244

MADD : Product Sum

Adds the value of ACC to the product of VF[fs] and VF[ft], and stores the result in VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 1

63	62	61	60	59	58	57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38	37 36 35 34 33 32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	MADD
-	-	-	-	-	0	0	----	-----	-----	-----	101001
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

MADD.dest **VF[**fd**]**dest, **VF[**fs**]**dest, **VF[**ft**]**dest

Operation

if (x ⊆ dest) then VF[fd]x = ACCx + VF[fs]x × VF[ft]x
if (y ⊆ dest) then VF[fd]y = ACCy + VF[fs]y × VF[ft]y
if (z ⊆ dest) then VF[fd]z = ACCz + VF[fs]z × VF[ft]z
if (w ⊆ dest) then VF[fd]w = ACCw + VF[fs]w × VF[ft]w

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MADD.xyzw VF10xyzw, VF20xyzw, VF30xyzw
```

VF10x = ACCx + VF20x × VF30x
VF10y = ACCy + VF20y × VF30y
VF10z = ACCz + VF20z × VF30z
VF10w = ACCw + VF20w × VF30w

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MADDi : Product Sum; with I Register

Multiplies each field of VF[fs] by the I register, then adds the product to the corresponding field of ACC, and stores the result in VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 1

63	62	61	60	59	58	57	56-53	52-48	47-43	42-38	37-32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	MADDi
-	-	-	-	-	0	0	----	00000	-----	-----	100011
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

MADDi.dest **VF[**fd**]**dest, **VF[**fs**]**dest, **I**

Operation

if (x ⊆ dest) then VF[fd]x = ACCx + VF[fs]x × I
if (y ⊆ dest) then VF[fd]y = ACCy + VF[fs]y × I
if (z ⊆ dest) then VF[fd]z = ACCz + VF[fs]z × I
if (w ⊆ dest) then VF[fd]w = ACCw + VF[fs]w × I

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MADDi.xyzw VF10xyzw, VF20xyzw, I
```

VF10x = ACCx + VF20x × I
VF10y = ACCy + VF20y × I
VF10z = ACCz + VF20z × I
VF10w = ACCw + VF20w × I

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MADDq : Product Sum; by Q Register

Multiplies each field of VF[fs] by the Q register, then adds the product to the corresponding field of ACC, and stores the result in VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 1

63	62	61	60	59	58	57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38	37 36 35 34 33 32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	MADDq
-	-	-	-	-	0	0	----	00000	-----	-----	100001
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

MADDq.dest **VF[**fd**]**dest**, VF[**fs**]**dest**, Q**

Operation

if (x ⊆ dest) then VF[fd]x = ACCx + VF[fs]x × Q
if (y ⊆ dest) then VF[fd]y = ACCy + VF[fs]y × Q
if (z ⊆ dest) then VF[fd]z = ACCz + VF[fs]z × Q
if (w ⊆ dest) then VF[fd]w = ACCw + VF[fs]w × Q

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MADDq.xyzw VF10xyzw, VF20xyzw, Q
```

VF10x = ACCx + VF20x × Q
VF10y = ACCy + VF20y × Q
VF10z = ACCz + VF20z × Q
VF10w = ACCw + VF20w × Q

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MADDbc : Broadcast Product Sum

Multiplies each field of VF[fs] by the specified field of VF[ft], then adds the product to the corresponding field of ACC and stores the result in VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 0

63	62	61	60	59	58	57	56–53	52–48	47–43	42–38	37–34	33–32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	MADD?	bc
-	-	-	-	-	0	0	----	-----	-----	-----	0010	--
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Mnemonic

MADDbc.dest **VF[**fd**]**dest, **VF[**fs**]**dest, **VF[**ft**]**bc

Operation

if (x ⊆ dest) then VF[fd]x = ACCx + VF[fs]x × VF[ft]bc
if (y ⊆ dest) then VF[fd]y = ACCy + VF[fs]y × VF[ft]bc
if (z ⊆ dest) then VF[fd]z = ACCz + VF[fs]z × VF[ft]bc
if (w ⊆ dest) then VF[fd]w = ACCw + VF[fs]w × VF[ft]bc

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MADDx.xyzw VF10xyzw, VF20xyzw, VF30x
```

VF10x = ACCx + VF20x × VF30x
VF10y = ACCy + VF20y × VF30x
VF10z = ACCz + VF20z × VF30x
VF10w = ACCw + VF20w × VF30x

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MADDA : Product Sum; to Accumulator

Multiplies VF[fs] by VF[ft], then adds the product to ACC and stores the result in ACC.

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56-53	52-48	47-43	42-32		
I	E	M	D	T	-	-	dest	ft reg	fs reg	MADDA		
-	-	-	-	-	0	0	----	-----	-----	01010	1111	01
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

MADDA.dest **ACC**dest, **VF[**fs**]**dest, **VF[**ft**]**dest

Operation

if (x ⊆ dest) then ACCx = ACCx + VF[fs]x × VF[ft]x
if (y ⊆ dest) then ACCy = ACCy + VF[fs]y × VF[ft]y
if (z ⊆ dest) then ACCz = ACCz + VF[fs]z × VF[ft]z
if (w ⊆ dest) then ACCw = ACCw + VF[fs]w × VF[ft]w

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MADDA.xyzw ACCxyzw, VF20xyzw, VF30xyzw
```

ACCx = ACCx + VF20x × VF30x
ACCy = ACCy + VF20y × VF30y
ACCz = ACCz + VF20z × VF30z
ACCw = ACCw + VF20w × VF30w

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MADDAi : Product Sum; by I register, to Accumulator

Multiplies each field of VF[fs] by the I register, then adds the product to the corresponding field of ACC and stores the result in ACC.

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56-53	52-48	47-43	42-32		
I	E	M	D	T	-	-	dest	ft reg	fs reg	MADDAi		
-	-	-	-	-	0	0	----	00000	-----	01000	1111	11
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

MADDAi.dest **ACC**dest, **VF[**fs**]**dest, **I**

Operation

if (x ⊆ dest) then ACCx = ACCx + VF[fs]x × I
if (y ⊆ dest) then ACCy = ACCy + VF[fs]y × I
if (z ⊆ dest) then ACCz = ACCz + VF[fs]z × I
if (w ⊆ dest) then ACCw = ACCw + VF[fs]w × I

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MADDAi.xyzw ACCxyzw, VF20xyzw, I
```

ACCx = ACCx + VF20x × I
ACCy = ACCy + VF20y × I
ACCz = ACCz + VF20z × I
ACCw = ACCw + VF20w × I

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MADDAq : Product Sum; by Q Register, to Accumulator

Multiplies each field of VF[fs] by the Q register, then adds the product to the corresponding field of ACC and stores the result in ACC.

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38	37 36 35 34	33 32
I	E	M	D	T	-	-	dest	ft reg	fs reg	MADDAq		
-	-	-	-	-	0	0	----	00000	-----	01000	1111	01
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

MADDAq.dest **ACC**dest, **VF[**fs**]**dest, **Q**

Operation

if (x ⊆ dest) then ACCx = ACCx + VF[fs]x × Q
if (y ⊆ dest) then ACCy = ACCy + VF[fs]y × Q
if (z ⊆ dest) then ACCz = ACCz + VF[fs]z × Q
if (w ⊆ dest) then ACCw = ACCw + VF[fs]w × Q

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MADDAq.xyzw ACCxyzw, VF20xyzw, Q
```

ACCx = ACCx + VF20x × Q
ACCy = ACCy + VF20y × Q
ACCz = ACCz + VF20z × Q
ACCw = ACCw + VF20w × Q

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MADDAbc : Broadcast Product Sum; to Accumulator

Multiplies each field of VF[fs] by the specified field of VF[ft], then adds the product to the corresponding field of ACC and stores the result in ACC.

Operation Code

Upper 32-bit word: UpperOP field type 2

63	62	61	60	59	58	57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38 37 36 35 34	33 32
I	E	M	D	T	-	-	dest	ft reg	fs reg	MADDA?	bc
-	-	-	-	-	0	0	----	-----	-----	00010 1111	--
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	9 bits	2 bits

Mnemonic

MADDAbc.dest **ACC**dest, **VF[**fs**]**dest, **VF[**ft**]**bc

Operation

if (x ⊆ dest) then ACCx = ACCx + VF[fs]x × VF[ft]bc
if (y ⊆ dest) then ACCy = ACCy + VF[fs]y × VF[ft]bc
if (z ⊆ dest) then ACCz = ACCz + VF[fs]z × VF[ft]bc
if (w ⊆ dest) then ACCw = ACCw + VF[fs]w × VF[ft]bc

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MADDAx.xyzw ACCxyzw, VF20xyzw, VF30x
```

ACCx = ACCx + VF20x × VF30x
ACCy = ACCy + VF20y × VF30x
ACCz = ACCz + VF20z × VF30x
ACCw = ACCw + VF20w × VF30x

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MAX : Maximum Value

Compares VF[fs] with VF[ft] and stores the greater value in VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 1

63	62	61	60	59	58	57	56–53	52–48	47–43	42–38	37–32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	MAX
-	-	-	-	-	0	0	----	-----	-----	-----	101011
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

MAX.dest **VF[**fd**]**dest, **VF[**fs**]**dest, **VF[**ft**]**dest

Operation

if (x ⊆ dest) then
if (VF[fs]x > VF[ft]x)
{VF[fd]x = VF[fs]x}
else
{VF[fd]x = VF[ft]x}

(The same operation is performed for the y and z fields.)

if (w ⊆ dest) then
if (VF[fs]w > VF[ft]w)
{VF[fd]w = VF[fs]w}
else
{VF[fd]w = VF[ft]w}

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
MAX.xyzw VF10xyzw, VF20xyzw, VF30xyzw
```

if (VF20x > VF30x) then {VF10x = VF20x} else {VF10x = VF30x}
if (VF20y > VF30y) then {VF10y = VF20y} else {VF10y = VF30y}
if (VF20z > VF30z) then {VF10z = VF20z} else {VF10z = VF30z}
if (VF20w > VF30w) then {VF10w = VF20w} else {VF10w = VF30w}

MAXi : Maximum Value

Compares each field of VF[fs] with the I register and stores the greater field in the corresponding field of VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 1

63	62	61	60	59	58	57	56–53	52–48	47–43	42–38	37–32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	MAXi
-	-	-	-	-	0	0	----	00000	-----	-----	011101
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

MAXi.dest **VF[**fd**]**dest**, VF[**fs**]**dest**, I**

Operation

if (x ⊆ dest) then
if (VF[fs]x > I) then
VF[fd]x = VF[fs]x
else
VF[fd]x = I

(The same operation is performed for the y and z fields.)

if (w ⊆ dest) then
if (VF[fs]w > I) then
VF[fd]w = VF[fs]w
else
VF[fd]w = I

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
MAXi.xyzw VF10xyzw, VF20xyzw, I
```

if (VF20x > I) then {VF10x = VF20x} else {VF10x = I}
if (VF20y > I) then {VF10y = VF20y} else {VF10y = I}
if (VF20z > I) then {VF10z = VF20z} else {VF10z = I}
if (VF20w > I) then {VF10w = VF20w} else {VF10w = I}

MAXbc : Maximum Value

Compares value of each field of VF[fs] with the specified field of VF[ft] and stores the greater value in the corresponding field of VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 0

63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32

I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	MAX?	bc
-	-	-	-	-	0	0	----	-----	-----	-----	0100	--
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Mnemonic

MAXbc.dest **VF[**fd**]**dest, **VF[**fs**]**dest, **VF[**ft**]**bc

Operation

if (x ⊆ dest) then
if (VF[fs]x > VF[ft]bc)
{VF[fd]x = VF[fs]x}
else
{VF[fd]x = VF[ft]bc}

(The same operation is performed for the y and z fields.)

if (w ⊆ dest) then
if (VF[fs]w > VF[ft]bc)
{VF[fd]w = VF[fs]w}
else
{VF[fd]w = VF[ft]bc}

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
MAXw.xyzw VF01xyzw, VF01xyzw, VF00w
```

A value of less than 1.0 in each field of VF01 is replaced with 1.0.

MINI : Minimum Value

Compares VF[fs] with VF[ft] and stores the smaller value in VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 1

63	62	61	60	59	58	57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38	37 36 35 34 33 32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	MINI
-	-	-	-	-	0	0	----	-----	-----	-----	101111
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

MINI.dest **VF[**fd**]**dest, **VF[**fs**]**dest, **VF[**ft**]**dest

Operation

if (x ⊆ dest) then
if (VF[fs]x < VF[ft]x)
{VF[fd]x = VF[fs]x}
else
{VF[fd]x = VF[ft]x}

(The same operation is performed for the y and z fields.)

if (w ⊆ dest) then
if (VF[fs]w < VF[ft]w)
{VF[fd]w = VF[fs]w}
else
{VF[fd]w = VF[ft]w}

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
MINI.xyzw VF10xyzw, VF20xyzw, VF30xyzw
```

if (VF20x < VF30x) then {VF10x = VF20x} else {VF10x = VF30x}
if (VF20y < VF30y) then {VF10y = VF20y} else {VF10y = VF30y}
if (VF20z < VF30z) then {VF10z = VF20z} else {VF10z = VF30z}
if (VF20w < VF30w) then {VF10w = VF20w} else {VF10w = VF30w}

MINIi : Minimum Value

Compares each field of VF[fs] with the I register and stores the smaller value in the corresponding field of VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 1

63 62 61 60 59 58 57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38	37 36 35 34 33 32
I E M D T - -	dest	ft reg	fs reg	fd reg	MINIi
- - - - - 0 0	----	00000	-----	-----	011111
1 1 1 1 1 1 1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

MINIi.dest **VF[**fd**]**dest**, VF[**fs**]**dest**, I**

Operation

if (x ⊆ dest) then
if (VF[fs]x < I) then
VF[fd]x = VF[fs]x
else
VF[fd]x = I

(The same operation is performed for the y and z fields.)

if (w ⊆ dest) then
if (VF[fs]w < I) then
VF[fd]w = VF[fs]w
else
VF[fd]w = I

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
MINIi.xyzw VF10xyzw, VF20xyzw, I
```

if (VF20x < I) then {VF10x = VF20x} else {VF10x = I}
if (VF20y < I) then {VF10y = VF20y} else {VF10y = I}
if (VF20z < I) then {VF10z = VF20z} else {VF10z = I}
if (VF20w < I) then {VF10w = VF20w} else {VF10w = I}

MINIbc : Minimum Value

Compares each field of VF[fs] with the specified field of VF[ft] and stores the smaller value in the corresponding field of VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 0

63	62	61	60	59	58	57	56-53	52-48	47-43	42-38	37-34	33-32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	MINI?	bc
-	-	-	-	-	0	0	----	-----	-----	-----	0101	--
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Mnemonic

MINIbc.dest **VF[**fd**]**dest, **VF[**fs**]**dest, **VF[**ft**]**bc

Operation

if (x ⊆ dest) then
if (VF[fs]x < VF[ft]bc)
{VF[fd]x = VF[fs]x}
else
{VF[fd]x = VF[ft]bc}

(The same operation is performed for the y and z fields.)

if (w ⊆ dest) then
if (VF[fs]w < VF[ft]bc)
{VF[fd]w = VF[fs]w}
else
{VF[fd]w = VF[ft]bc}

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
MINIx.xyzw VF10xyzw, VF20xyzw, VF30x
```

if (VF20x < VF30x) then {VF10x = VF20x} else {VF10x = VF30x}
if (VF20y < VF30x) then {VF10y = VF20y} else {VF10y = VF30x}
if (VF20z < VF30x) then {VF10z = VF20z} else {VF10z = VF30x}
if (VF20w < VF30x) then {VF10w = VF20w} else {VF10w = VF30x}

MSUB : Multiply and Subtract

Multiplies VF[fs] and VF[ft], then subracts the product obtained from the value of ACC, and stores the result in VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 1

63	62	61	60	59	58	57	56-53	52-48	47-43	42-38	37-32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	MSUB
-	-	-	-	-	0	0	----	-----	-----	-----	101101
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

MSUB.dest **VF[**fd**]**dest, **VF[**fs**]**dest, **VF[**ft**]**dest

Operation

if (x ⊆ dest) then VF[fd]x = ACCx - VF[fs]x × VF[ft]x
if (y ⊆ dest) then VF[fd]y = ACCy - VF[fs]y × VF[ft]y
if (z ⊆ dest) then VF[fd]z = ACCz - VF[fs]z × VF[ft]z
if (w ⊆ dest) then VF[fd]w = ACCw - VF[fs]w × VF[ft]w

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MSUB.xyzw VF10xyzw, VF20xyzw, VF30xyzw
```

VF10x = ACCx - VF20x × VF30x
VF10y = ACCy - VF20y × VF30y
VF10z = ACCz - VF20z × VF30z
VF10w = ACCw - VF20w × VF30w

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MSUBi : Multiply and Subtract; with I Register

Multiplies each field of VF[fs] by the I register, then subtracts the product from the corresponding field of ACC and stores the result in the corresponding field of VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 1

63	62	61	60	59	58	57	56–53	52–48	47–43	42–38	37–32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	MSUBi
-	-	-	-	-	0	0	----	00000	-----	-----	100111
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

MSUBi.dest **VF[**fd**]**dest**, VF[**fs**]**dest**, I**

Operation

if (x ⊆ dest) then VF[fd]x = ACCx - VF[fs]x × I
if (y ⊆ dest) then VF[fd]y = ACCy - VF[fs]y × I
if (z ⊆ dest) then VF[fd]z = ACCz - VF[fs]z × I
if (w ⊆ dest) then VF[fd]w = ACCw - VF[fs]w × I

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MSUBi.xyzw VF10xyzw, VF20xyzw, I
```

VF10x = ACCx - VF20x × I
VF10y = ACCy - VF20y × I
VF10z = ACCz - VF20z × I
VF10w = ACCw - VF20w × I

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MSUBq : Multiply and Subtract; by Q Register

Multiplies each field of VF[fs] by the Q register, then subtracts the product from the corresponding field of ACC and stores the result in the corresponding field of VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 1

63	62	61	60	59	58	57	56-53	52-48	47-43	42-38	37-32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	MSUBq
-	-	-	-	-	0	0	----	00000	-----	-----	100101
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

MSUBq.dest **VF[**fd**]**dest**, VF[**fs**]**dest**, Q**

Operation

if (x ⊆ dest) then VF[fd]x = ACCx - VF[fs]x × Q
if (y ⊆ dest) then VF[fd]y = ACCy - VF[fs]y × Q
if (z ⊆ dest) then VF[fd]z = ACCz - VF[fs]z × Q
if (w ⊆ dest) then VF[fd]w = ACCw - VF[fs]w × Q

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MSUBq.xyzw VF10xyzw, VF20xyzw, Q
```

VF10x = ACCx - VF20x × Q
VF10y = ACCy - VF20y × Q
VF10z = ACCz - VF20z × Q
VF10w = ACCw - VF20w × Q

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MSUBbc : Broadcast Multiply and Subtract

Multiplies each field of VF[fs] by the specified field of VF[ft], then subtracts the product from the corresponding field of ACC and stores the result in the corresponding field of VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 0

63	62	61	60	59	58	57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38	37 36 35 34	33 32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	MSUB?	bc
-	-	-	-	-	0	0	----	-----	-----	-----	0011	--
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Mnemonic

MSUBbc.dest **VF[**fd**]**dest, **VF[**fs**]**dest, **VF[**ft**]**bc

Operation

if (x ⊆ dest) then VF[fd]x = ACCx - VF[fs]x × VF[ft]bc
if (y ⊆ dest) then VF[fd]y = ACCy - VF[fs]y × VF[ft]bc
if (z ⊆ dest) then VF[fd]z = ACCz - VF[fs]z × VF[ft]bc
if (w ⊆ dest) then VF[fd]w = ACCw - VF[fs]w × VF[ft]bc

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MSUBx.xyzw VF10xyzw, VF20xyzw, VF30x
```

VF10x = ACCx - VF20x × VF30x
VF10y = ACCy - VF20y × VF30x
VF10z = ACCz - VF20z × VF30x
VF10w = ACCw - VF20w × VF30x

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MSUBA : Multiply and Subtract; to Accumulator

Multiplies VF[fs] and VF[ft], then subtracts the product from ACC and stores the result in ACC.

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56-53	52-48	47-43	42-32
I	E	M	D	T	-	-	dest	ft reg	fs reg	MSUBA
-	-	-	-	-	0	0	----	-----	-----	01011 1111 01
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

MSUBA.dest **ACC**dest, **VF[**fs**]**dest, **VF[**ft**]**dest

Operation

if (x ⊆ dest) then ACCx = ACCx - VF[fs]x × VF[ft]x
if (y ⊆ dest) then ACCy = ACCy - VF[fs]y × VF[ft]y
if (z ⊆ dest) then ACCz = ACCz - VF[fs]z × VF[ft]z
if (w ⊆ dest) then ACCw = ACCw - VF[fs]w × VF[ft]w

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MSUBA.xyzw ACCxyzw, VF20xyzw, VF30xyzw
```

ACCx = ACCx - VF20x × VF30x
ACCy = ACCy - VF20y × VF30y
ACCz = ACCz - VF20z × VF30z
ACCw = ACCw - VF20w × VF30w

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MSUBAi : Multiply and Subtract; with I Register, to Accumulator

Multiplies each field of VF[fs] and the I register, then subtracts the product from the corresponding field of ACC and stores the result in ACC.

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56-53	52-48	47-43	42-32
I	E	M	D	T	-	-	dest	ft reg	fs reg	MSUBAi
-	-	-	-	-	0	0	----	00000	-----	01001 1111 11
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

MSUBAi.dest **ACC**dest, **VF[**fs**]**dest, **I**

Operation

if (x ⊆ dest) then ACCx = ACCx - VF[fs]x × I
if (y ⊆ dest) then ACCy = ACCy - VF[fs]y × I
if (z ⊆ dest) then ACCz = ACCz - VF[fs]z × I
if (w ⊆ dest) then ACCw = ACCw - VF[fs]w × I

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MSUBAi.xyzw ACCxyzw, VF20xyzw, I
```

ACCx = ACCx - VF20x × I
ACCy = ACCy - VF20y × I
ACCz = ACCz - VF20z × I
ACCw = ACCw - VF20w × I

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MSUBAq : Multiply and Subtract; by Q Register, to Accumulator

Multiplies VF[fs] by the Q register, then subtracts the product from the corresponding field of ACC and stores the result in the corresponding field of ACC.

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56-53	52-48	47-43	42-32		
I	E	M	D	T	-	-	dest	ft reg	fs reg	MSUBAq		
-	-	-	-	-	0	0	----	00000	-----	01001	1111	01
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

MSUBAq.dest **ACC**dest, **VF[**fs**]**dest, **Q**

Operation

if (x ⊆ dest) then ACCx = ACCx - VF[fs]x × Q
if (y ⊆ dest) then ACCy = ACCy - VF[fs]y × Q
if (z ⊆ dest) then ACCz = ACCz - VF[fs]z × Q
if (w ⊆ dest) then ACCw = ACCw - VF[fs]w × Q

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MSUBAq.xyzw ACCxyzw, VF20xyzw, Q
```

ACCx = ACCx - VF20x × Q
ACCy = ACCy - VF20y × Q
ACCz = ACCz - VF20z × Q
ACCw = ACCw - VF20w × Q

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MSUBAbc : Broadcast Multiply and Subtract; to Accumulator

Multiplies each field of VF[fs] and the specified field of VF[ft], then subtracts the product from the corresponding field of ACC and stores the result in the corresponding field of ACC.

Operation Code

Upper 32-bit word: UpperOP field type 2

63 62 61 60 59 58 57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38 37 36 35 34	33 32
I E M D T - -	dest	ft reg	fs reg	MSUBA?	bc
- - - - - 0 0	----	-----	-----	00011 1111	--
1 1 1 1 1 1 1	4 bits	5 bits	5 bits	9 bits	2 bits

Mnemonic

MSUBAbc.dest **ACC**dest, **VF[**fs**]**dest, **VF[**ft**]**bc

Operation

if (x ⊆ dest) then ACCx = ACCx - VF[fs]x × VF[ft]bc
if (y ⊆ dest) then ACCy = ACCy - VF[fs]y × VF[ft]bc
if (z ⊆ dest) then ACCz = ACCz - VF[fs]z × VF[ft]bc
if (w ⊆ dest) then ACCw = ACCw - VF[fs]w × VF[ft]bc

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MSUBAx.xyzw ACCxyzw, VF20xyzw, VF30x
```

ACCx = ACCx - VF20x × VF30x
ACCy = ACCy - VF20y × VF30x
ACCz = ACCz - VF20z × VF30x
ACCw = ACCw - VF20w × VF30x

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MUL : Multiply

Multiplies VF[fs] by VF[ft] and stores the result in VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 1

63	62	61	60	59	58	57	56–53	52–48	47–43	42–38	37–32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	MUL
-	-	-	-	-	0	0	----	-----	-----	-----	101010
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

MUL.dest **VF[**fd**]**dest, **VF[**fs**]**dest, **VF[**ft**]**dest

Operation

if (x ⊆ dest) then VF[fd]x = VF[fs]x × VF[ft]x
if (y ⊆ dest) then VF[fd]y = VF[fs]y × VF[ft]y
if (z ⊆ dest) then VF[fd]z = VF[fs]z × VF[ft]z
if (w ⊆ dest) then VF[fd]w = VF[fs]w × VF[ft]w

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MUL.xyzw VF10xyzw, VF20xyzw, VF30xyzw
```

VF10x = VF20x × VF30x
VF10y = VF20y × VF30y
VF10z = VF20z × VF30z
VF10w = VF20w × VF30w

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MULi : Multiply by I Register

Multiplies each field of VF[fs] by the I register and stores the result in the corresponding field of VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 1

63	62	61	60	59	58	57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38	37 36 35 34 33 32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	MULi
-	-	-	-	-	0	0	----	00000	-----	-----	011110
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

MULi.dest **VF[**fd**]**dest, **VF[**fs**]**dest, **I**

Operation

if (x ⊆ dest) then VF[fd]x = VF[fs]x × I
if (y ⊆ dest) then VF[fd]y = VF[fs]y × I
if (z ⊆ dest) then VF[fd]z = VF[fs]z × I
if (w ⊆ dest) then VF[fd]w = VF[fs]w × I

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MULi.xyzw VF10xyzw, VF20xyzw, I
```

VF10x = VF20x × I
VF10y = VF20y × I
VF10z = VF20z × I
VF10w = VF20w × I

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MULq : Multiply by Q Register

Multiplies each field of VF[fs] by the Q register and stores the result in the corresponding field of VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 1

63	62	61	60	59	58	57	56–53	52–48	47–43	42–38	37–32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	MULq
-	-	-	-	-	0	0	----	00000	-----	-----	011100
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

MULq.dest **VF[**fd**]**dest, **VF[**fs**]**dest, **Q**

Operation

if (x ⊆ dest) then VF[fd]x = VF[fs]x × Q
if (y ⊆ dest) then VF[fd]y = VF[fs]y × Q
if (z ⊆ dest) then VF[fd]z = VF[fs]z × Q
if (w ⊆ dest) then VF[fd]w = VF[fs]w × Q

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MULq.xyzw VF10xyzw, VF20xyzw, Q
```

VF10x = VF20x × Q
VF10y = VF20y × Q
VF10z = VF20z × Q
VF10w = VF20w × Q

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MULbc : Multiply by Broadcast

Multiplies each field of VF[fs] by the specified field of VF[ft] and stores the result in the corresponding field of VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 0

63	62	61	60	59	58	57	56-53	52-48	47-43	42-38	37-34	33-32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	MUL?	bc
-	-	-	-	-	0	0	----	-----	-----	-----	0110	--
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Mnemonic

MULbc.dest **VF[**fd**]**dest, **VF[**fs**]**dest, **VF[**ft**]**bc

Operation

if (x ⊆ dest) then VF[fd]x = VF[fs]x × VF[ft]bc
if (y ⊆ dest) then VF[fd]y = VF[fs]y × VF[ft]bc
if (z ⊆ dest) then VF[fd]z = VF[fs]z × VF[ft]bc
if (w ⊆ dest) then VF[fd]w = VF[fs]w × VF[ft]bc

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MULx.xyzw VF10xyzw, VF20xyzw, VF30x
```

VF10x = VF20x × VF30x
VF10y = VF20y × VF30x
VF10z = VF20z × VF30x
VF10w = VF20w × VF30x

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MULA : Multiply; to Accumulator

Multiplies VF[fs] by VF[ft] and stores the result in ACC.

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56-53	52-48	47-43	42-32		
I	E	M	D	T	-	-	dest	ft reg	fs reg	MULA		
-	-	-	-	-	0	0	----	-----	-----	01010	1111	10
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

MULA.dest **ACC**dest, **VF[**fs**]**dest, **VF[**ft**]**dest

Operation

if (x ⊆ dest) then ACCx = VF[fs]x × VF[ft]x
if (y ⊆ dest) then ACCy = VF[fs]y × VF[ft]y
if (z ⊆ dest) then ACCz = VF[fs]z × VF[ft]z
if (w ⊆ dest) then ACCw = VF[fs]w × VF[ft]w

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MULA.xyzw ACCxyzw, VF20xyzw, VF30xyzw
```

ACCx = VF20x × VF30x
ACCy = VF20y × VF30y
ACCz = VF20z × VF30z
ACCw = VF20w × VF30w

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MULAi : Multiply by I Register, to Accumulator

Multiplies each field of VF[fs] by the value of the I register and stores the result in ACC.

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56-53	52-48	47-43	42-38	37-34	33-32
I	E	M	D	T	-	-	dest	ft reg	fs reg	MULAi		
-	-	-	-	-	0	0	----	00000	-----	00111	1111	10
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

MULAi.dest **ACC**dest, **VF[**fs**]**dest, **I**

Operation

if (x ⊆ dest) then ACCx = VF[fs]x × I
if (y ⊆ dest) then ACCy = VF[fs]y × I
if (z ⊆ dest) then ACCz = VF[fs]z × I
if (w ⊆ dest) then ACCw = VF[fs]w × I

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MULAi.xyzw ACCxyzw, VF20xyzw, I
```

ACCx = VF20x × I
ACCy = VF20y × I
ACCz = VF20z × I
ACCw = VF20w × I

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MULAq : Multiply by Q Register, to Accumulator

Multiplies each field of VF[fs] by the Q register, and stores the result in ACC.

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56-53	52-48	47-43	42-32		
I	E	M	D	T	-	-	dest	ft reg	fs reg	MULAq		
-	-	-	-	-	0	0	----	00000	-----	00111	1111	00
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

MULAq.dest **ACC**dest, **VF[**fs**]**dest, **Q**

Operation

if (x ⊆ dest) then ACCx = VF[fs]x × Q
if (y ⊆ dest) then ACCy = VF[fs]y × Q
if (z ⊆ dest) then ACCz = VF[fs]z × Q
if (w ⊆ dest) then ACCw = VF[fs]w × Q

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MULAq.xyzw ACCxyzw, VF20xyzw, Q
```

ACCx = VF20x × Q
ACCy = VF20y × Q
ACCz = VF20z × Q
ACCw = VF20w × Q

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

MULAbc : Broadcast Multiply by broadcast, to Accumulator

Multiplies each field of VF[fs] by the specified field of VF[ft] and stores the result in the corresponding field of ACC.

Operation Code

Upper 32-bit word: UpperOP field type 2

63	62	61	60	59	58	57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38 37 36 35 34	33 32
I	E	M	D	T	-	-	dest	ft reg	fs reg	MULA?	bc
-	-	-	-	-	0	0	----	-----	-----	00110 1111	--
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	9 bits	2 bits

Mnemonic

MULAbc.dest **ACC**dest, **VF[**fs**]**dest, **VF[**ft**]**bc

Operation

if (x ⊆ dest) then ACCx = VF[fs]x × VF[ft]bc
if (y ⊆ dest) then ACCy = VF[fs]y × VF[ft]bc
if (z ⊆ dest) then ACCz = VF[fs]z × VF[ft]bc
if (w ⊆ dest) then ACCw = VF[fs]w × VF[ft]bc

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
MULAx.xyzw ACCxyzw, VF20xyzw, VF30x
```

ACCx = VF20x × VF30x
ACCy = VF20y × VF30x
ACCz = VF20z × VF30x
ACCw = VF20w × VF30x

Remarks

There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value. By using VF[fs] as a multiplicand, the results of multiplication with 1 are guaranteed to be accurate.

NOP : No Operation

No operation is performed.

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56-53	52-48	47-43	42-32
I	E	M	D	T	-	-	dest	ft reg	fs reg	NOP
-	-	-	-	-	0	0	0000	00000	00000	01011 1111 11
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

NOP

Operation

None

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

OPMULA : Vector Outer Product

Calculates the first part of the vector outer product of VF[fs] and VF[ft] and stores the result in ACC.

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56-53	52-48	47-43	42-32
I	E	M	D	T	-	-	dest	ft reg	fs reg	OPMULA
-	-	-	-	-	0	0	1110	-----	-----	01011 1111 10
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

OPMULA.xyz ACCxyz, VF[fs]xyz, VF[ft]xyz

Operation

ACCx = VF[fs]y × VF[ft]z
ACCy = VF[fs]z × VF[ft]x
ACCz = VF[fs]x × VF[ft]y

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

Vector Outer Product: VF20 × VF30(Be careful to the description order of VF20 and VF30.)

```
OPMULA.xyz ACCxyz, VF20xyz, VF30xyz
OPMSUB.xyz VF10xyz, VF30xyz, VF20xyz
```

VF10x = VF20y × VF30z - VF30y × VF20z
VF10y = VF20z × VF30x - VF30z × VF20x
VF10z = VF20x × VF30y - VF30x × VF20y

Remarks

The fields subject to the operation are fixed to x,y,z.
There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value.

OPMSUB : Vector Outer Product

Calculates the last part of the vector outer product of VF[fs], VF[ft] and ACC and stores the result in VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 1

63 62 61 60 59 58 57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38	37 36 35 34 33 32
I E M D T - -	dest	ft reg	fs reg	fd reg	OPMSUB
- - - - - 0 0	1110	-----	-----	-----	101110
1 1 1 1 1 1 1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

OPMSUB.xyz VF[fd**]xyz, VF[**fs**]xyz, VF[**ft**]xyz**

Operation

VF[fd]x = ACCx - VF[fs]y × VF[ft]z
VF[fd]y = ACCy - VF[fs]z × VF[ft]x
VF[fd]z = ACCz - VF[fs]x × VF[ft]y

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

Vector Outer Product: VF20 × VF30(Be careful with the description order of VF20 and VF30.)

```
OPMULA.xyz ACCxyz, VF20xyz, VF30xyz
OPMSUB.xyz VF10xyz, VF30xyz, VF20xyz
```

VF10x = VF20y × VF30z - VF30y × VF20z
VF10y = VF20z × VF30x - VF30z × VF20x
VF10z = VF20x × VF30y - VF30x × VF20y

Remarks

The fields subject to the operation are fixed to x,y,z.
There is an operation error of 1 bit in multiplication, so the value multiplied by 1 may not be the same as the original value.

SUB : Subtract

Subtracts VF[ft] from VF[fs] and stores the result in VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 1

63	62	61	60	59	58	57	56-53	52-48	47-43	42-38	37-32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	SUB
-	-	-	-	-	0	0	----	-----	-----	-----	101100
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

SUB.dest **VF[**fd**]**dest, **VF[**fs**]**dest, **VF[**ft**]**dest

Operation

if (x ⊆ dest) then VF[fd]x = VF[fs]x - VF[ft]x
if (y ⊆ dest) then VF[fd]y = VF[fs]y - VF[ft]y
if (z ⊆ dest) then VF[fd]z = VF[fs]z - VF[ft]z
if (w ⊆ dest) then VF[fd]w = VF[fs]w - VF[ft]w

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
SUB.xyzw VF01xyzw, VF00xyzw, VF00xyzw
```

The values of all fields of VF01 all become 0.0.

Remarks

When VF00 is specified as the destination, the instruction is used to compare VF[fs] with VF[ft].

SUBi : Subtract I Register

Subtracts the I register from each field of VF[fs] and stores the result in the corresponding fields of VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 1

63	62	61	60	59	58	57	56-53	52-48	47-43	42-38	37-32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	SUBi
-	-	-	-	-	0	0	----	00000	-----	-----	100110
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

SUBi.dest **VF[**fd**]**dest**, VF[**fs**]**dest**, I**

Operation

if (x ⊆ dest) then VF[fd]x = VF[fs]x - I
if (y ⊆ dest) then VF[fd]y = VF[fs]y - I
if (z ⊆ dest) then VF[fd]z = VF[fs]z - I
if (w ⊆ dest) then VF[fd]w = VF[fs]w - I

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
SUBi.xyzw VF10xyzw, VF20xyzw, I
```

VF10x = VF20x - I
VF10y = VF20y - I
VF10z = VF20z - I
VF10w = VF20w - I

Remarks

When VF00 is specified as the destination, the instruction is used to compare each field of VF[fs] with the I register.

SUBq : Subtract Q Register

Subtracts the Q register from each field of VF[fs] and stores the result in the corresponding field of VF[ft].

Operation Code

Upper 32-bit word: UpperOP field type 1

63	62	61	60	59	58	57	56-53	52-48	47-43	42-38	37-32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	SUBq
-	-	-	-	-	0	0	----	00000	-----	-----	100100
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

SUBq.dest **VF[**fd**]**dest, **VF[**fs**]**dest, **Q**

Operation

if (x ⊆ dest) then VF[fd]x = VF[fs]x - Q
if (y ⊆ dest) then VF[fd]y = VF[fs]y - Q
if (z ⊆ dest) then VF[fd]z = VF[fs]z - Q
if (w ⊆ dest) then VF[fd]w = VF[fs]w - Q

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
SUBq.xyzw VF10xyzw, VF20xyzw, Q
```

VF10x = VF20x - Q
VF10y = VF20y - Q
VF10z = VF20z - Q
VF10w = VF20w - Q

Remarks

When VF00 is specified as the destination, the instruction is used to compare each field of VF[fs] with the Q register .

SUBbc : Broadcast Subtract

Subtracts the specified field of VF[ft] from each field of VF[fs] and stores the result in the corresponding field of VF[fd].

Operation Code

Upper 32-bit word: UpperOP field type 0

63	62	61	60	59	58	57	56-53	52-48	47-43	42-38	37-34	33-32
I	E	M	D	T	-	-	dest	ft reg	fs reg	fd reg	SUB?	bc
-	-	-	-	-	0	0	----	-----	-----	-----	0001	--
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Mnemonic

`SUBbc.dest VF[fd]dest, VF[fs]dest, VF[ft]bc`

Operation

if (x ⊆ dest) then VF[fd]x = VF[fs]x - VF[ft]bc
if (y ⊆ dest) then VF[fd]y = VF[fs]y - VF[ft]bc
if (z ⊆ dest) then VF[fd]z = VF[fs]z - VF[ft]bc
if (w ⊆ dest) then VF[fd]w = VF[fs]w - VF[ft]bc

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

`SUBx.xyzw VF10xyzw, VF20xyzw, VF30x`

VF10x = VF20x - VF30x
VF10y = VF20y - VF30x
VF10z = VF20z - VF30x
VF10w = VF20w - VF30x

Remarks

When VF00 is specified as the destination, the instruction is used to compare each field of VF[fs] with the VF[ft]bc field.

SUBA : Substract; to Accumulator

Subtracts VF[ft] from VF[fs] and stores the result in ACC.

Operation Code

Upper 32-bit word: UpperOP field type 3

63 62 61 60 59 58 57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38 37 36 35 34 33 32		
I E M D T - -	dest	ft reg	fs reg	SUBA		
- - - - - 0 0	----	-----	-----	01011	1111	00
1 1 1 1 1 1 1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

SUBA.dest **ACC**dest, **VF[**fs**]**dest, **VF[**ft**]**dest

Operation

if (x ⊆ dest) then ACCx = VF[fs]x - VF[ft]x
if (y ⊆ dest) then ACCy = VF[fs]y - VF[ft]y
if (z ⊆ dest) then ACCz = VF[fs]z - VF[ft]z
if (w ⊆ dest) then ACCw = VF[fs]w - VF[ft]w

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
SUBA.xyzw ACCxyzw, VF20xyzw, VF30xyzw
```

ACCx = VF20x - VF30x
ACCy = VF20y - VF30y
ACCz = VF20z - VF30z
ACCw = VF20w - VF30w

SUBAi : Subtract I Register; to Accumulator

Subtracts the I register from each field of VF[fs] and stores the result in the corresponding field of ACC.

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56-53	52-48	47-43	42-38	37-34	33-32
I	E	M	D	T	-	-	dest	ft reg	fs reg	SUBAi		
-	-	-	-	-	0	0	----	00000	-----	01001	1111	10
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

SUBAi.dest **ACC**dest, **VF[**fs**]**dest, **I**

Operation

if (x ⊆ dest) then ACCx = VF[fs]x - I
if (y ⊆ dest) then ACCy = VF[fs]y - I
if (z ⊆ dest) then ACCz = VF[fs]z - I
if (w ⊆ dest) then ACCw = VF[fs]w - I

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
SUBAi.xyzw ACCxyzw, VF20xyzw, I
```

ACCx = VF20x - I
ACCy = VF20y - I
ACCz = VF20z - I
ACCw = VF20w - I

SUBAq : Subtract Q Register; to Accumulator

Subtracts the Q register from each field of VF[fs] and stores the result in the corresponding field of ACC.

Operation Code

Upper 32-bit word: UpperOP field type 3

63	62	61	60	59	58	57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38	37 36 35 34	33 32
I	E	M	D	T	-	-	dest	ft reg	fs reg	SUBAq		
-	-	-	-	-	0	0	----	00000	-----	01001	1111	00
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

SUBAq.dest **ACC**dest, **VF[**fs**]**dest, **Q**

Operation

if (x ⊆ dest) then ACCx = VF[fs]x - Q
if (y ⊆ dest) then ACCy = VF[fs]y - Q
if (z ⊆ dest) then ACCz = VF[fs]z - Q
if (w ⊆ dest) then ACCw = VF[fs]w - Q

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
SUBAq.xyzw ACCxyzw, VF20xyzw, Q
```

ACCx = VF20x - Q
ACCy = VF20y - Q
ACCz = VF20z - Q
ACCw = VF20w - Q

SUBAbc : Broadcast Subtract; to Accumulator

Subtracts the specified field of VF[ft] from each field of VF[fs] and stores the result in the corresponding field of ACC.

Operation Code

Upper 32-bit word: UpperOP field type 2

63	62	61	60	59	58	57	56 55 54 53	52 51 50 49 48	47 46 45 44 43	42 41 40 39 38	37 36 35 34	33 32
I	E	M	D	T	-	-	dest	ft reg	fs reg	SUBA?		bc
-	-	-	-	-	0	0	----	-----	-----	00001	1111	--
1	1	1	1	1	1	1	4 bits	5 bits	5 bits	9 bits		2 bits

Mnemonic

SUBAbc.dest **ACC**dest, **VF[**fs**]**dest, **VF[**ft**]**bc

Operation

if (x ⊆ dest) then ACCx = VF[fs]x - VF[ft]bc
if (y ⊆ dest) then ACCy = VF[fs]y - VF[ft]bc
if (z ⊆ dest) then ACCz = VF[fs]z - VF[ft]bc
if (w ⊆ dest) then ACCw = VF[fs]w - VF[ft]bc

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
XXXX	XXXX	XXXX	XXXX	-	-	X	X	X	X	-	-	X	X	X	X	-

Throughput/Latency

1 / 4

Example

```
SUBAx.xyzw ACCxyzw, VF20xyzw, VF30x
```

ACCx = VF20x - VF30x
ACCy = VF20y - VF30x
ACCz = VF20z - VF30x
ACCw = VF20w - VF30x

4.3. Lower Instruction Reference

This section describes the function, operation code, mnemonic, operation, flag changes, and throughput/latency of Lower instructions. They are listed in alphabetical order in mnemonic form. The descriptions also include examples, programming notes, and reference information.

B : Unconditional Branch

Branches to the PC relative address specified with the immediate value.

Operation Code

Lower 32-bit word: LowerOP field type 7

B (31–25)	dest (24–21)	it reg (20–16)	is reg (15–11)	Imm11 (10–00)
0100000	0000	00000	00000	-----------
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

B Imm11

Imm11 is a signed interger of 11-bit long; specify the value obtained by dividing the offset to branch destination by 8.

Operation

PC = PC + Imm11 × 8

The branch destination is determined by adding the value of Imm11, to the address of the instruction in the branch delay slot (one instruction).

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

2 / 2

Example

In the example below, the branch destination varies when VI10 matches with either VI01, VI02, or VI03.

```
NOP     BEQ VI10, VI01, PROG1
NOP     NOP
NOP     BEQ VI10, VI02, PROG2
NOP     NOP
NOP     BEQ VI10, VI03, PROG3
NOP     NOP
NOP     B DEFAULT
```

Remarks

This instruction cannot be placed in the E bit delay slot.

BAL : Unconditional Branch with Saving Address

Stores the address before branching in VI[it] and branches to the PC relative address specified with the immediate value.

Operation Code

Lower 32-bit word: LowerOP field type 7

31–25	24–21	20–16	15–11	10–00
BAL 0100001	dest 0000	it reg -----	is reg 00000	Imm11 -----------
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

BAL VI[it**],** Imm11

Imm11 is an 11-bit signed integer; specify the value obtained by dividing the offset to branch destination by 8.

Operation

VI[it] = PC + (2 × 8)
PC = PC + Imm11 × 8

The address of the instruction next to the branch delay slot (one instruction) is stored in VI[it].
The branch destination is determined by adding the value of Imm11 to the address of the instruction in the slot.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

2 / 2

Example

```
BAL VI15, LABEL:
```

VI15 = PC+(2 × 8)

Branches to LABEL (PC relative address)

Remarks

This instruction cannot be placed in the E bit delay slot.

DIV : Divide

Divides fsf field of VF[fs] by ftf field of VF[ft] and stores the result in the Q register.

Operation Code

Lower 32-bit word: LowerOP field type 4

Lower OP. (bits 31–25)	ftf (24–23)	fsf (22–21)	ft reg (20–16)	fs reg (15–11)	DIV (10–00)		
1000000	--	--	-----	-----	01110	1111	00
7 bits	2 bits	2 bits	5 bits	5 bits	11 bits		

Mnemonic

DIV Q, VF[fs**]**fsf**, VF[**ft**]**ftf

Operation

Q = VF[fs]fsf ÷ **VF[ft]**ftf

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	X	X	-	-	-	-	X	X	-	-	-	-	-

Throughput/Latency

7 / 7

Example

```
DIV Q, VF10x, VF20y
```

Q = VF10x ÷ VF20y

Remarks

A data dependency check is not performed with the Q register. To execute subsequent instructions after the results of the DIV instruction are written to the Q register, use the WAITQ instruction for synchronization.

EATAN : Arctangent

Calculates the arc tangent of fsf field of VF[fs] and stores the result in the P register.

Operation Code

Lower 32-bit word: LowerOP field type 4

Lower OP. (bits 31–25)	ftf (24–23)	fsf (22–21)	ft reg (20–16)	fs reg (15–11)	EATAN (10–00)		
1000000	00	--	00000	-----	11111	1111	01
7 bits	2 bits	2 bits	5 bits	5 bits	11 bits		

Mnemonic

EATAN P, VF[fs**]**fsf

Operation

P = arctan(VF[fs]fsf **)**

The following approximation formula is used for calculating arctan.

Calculation is valid for: 0 <= x <= 1

$$\arctan(x) = (T_1 \times t + T_2 \times t^3 + T_3 \times t^5 + T_4 \times t^7 + T_5 \times t^9 + T_6 \times t^{11} + T_7 \times t^{13} + T_8 \times t^{15}) + \frac{\pi}{4}$$

Provided $t = \frac{(x-1)}{(x+1)}$

Constants	Decimal expressions	Single precision floating-point expressions			Hex. Expressions
		S	E	F	
T_1	0.999999344348907	0	01111110	11111111111111111110101	3f7ffff5
T_2	-0.333298563957214	1	01111101	01010101010011000011100	beaaa61c
T_3	0.199465364217758	0	01111100	10011000100000010100110	3e4c40a6
T_4	-0.139085337519646	1	01111100	00011100110110001100011	be0e6c63
T_5	0.096420042216778	0	01111011	10001010111011111011111	3dc577df
T_6	-0.055909886956215	1	01111010	11001010000000111000100	bd6501c4
T_7	0.021861229091883	0	01111001	01100110001011001010010	3cb31652
T_8	-0.004054057877511	1	01110111	00001001101011111100111	bb84d7e7
$\pi/4$	0.785398185253143	0	01111110	10010010000111111011011	3f490fdb

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

53 / 54

Example

```
EATAN P, VF10x
```

P = arctan(VF10x)

Remarks

A data dependency check is not performed with the P register. To execute subsequent instructions after the results of the EATAN instruction are written to the P register, use the WAITP instruction for synchronization.

EATANxy : Arctangent

Calculates the arctangent based on the x, y fields of VF[fs] and stores in the P register.

Operation Code

Lower 32-bit word: LowerOP field type 3

Lower OP. (31–25)	dest (24–21)	ft reg (20–16)	fs reg (15–11)	EATANxy (10–00)		
1000000	1100	00000	-----	11101	1111	00
7 bits	4 bits	5 bits	5 bits	11 bits		

Mnemonic

EATANxy P, VF[fs**]**

Operation

P = arctan(VF[fs]y / VF[fs]x)

The following approximation formula is used for calculating arctan.

Calculation is valid for: 0 <= y <= x (Excluding 0 = y =x)

$$\arctan(x) = (T_1 \times t + T_2 \times t^3 + T_3 \times t^5 + T_4 \times t^7 + T_5 \times t^9 + T_6 \times t^{11} + T_7 \times t^{13} + T_8 \times t^{15}) + \frac{\pi}{4}$$

Provided $t = \frac{(y-1)}{(y+x)}$

Constants	Decimal Expressions	Single precision floating-point expressions			Hex. expressions
		S	E	F	
T_1	0.999999344348907	0	01111110	11111111111111111110101	3f7ffff5
T_2	-0.333298563957214	1	01111101	01010101010011000011100	beaaa61c
T_3	0.199465364217758	0	01111100	10011000100000010100110	3e4c40a6
T_4	-0.139085337519646	1	01111100	00011100110110001100011	be0e6c63
T_5	0.096420042216778	0	01111011	10001010111011111011111	3dc577df
T_6	-0.055909886956215	1	01111010	11001010000000111000100	bd6501c4
T_7	0.021861229091883	0	01111001	01100110001011001010010	3cb31652
T_8	-0.004054057877511	1	01110111	00001001101011111100111	bb84d7e7
$\pi/4$	0.785398185253143	0	01111110	10010010000111111011011	3f490fdb

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

53 / 54

Example

```
EATANxy P, VF10
```

P = arctan(VF10y / VF10x)

Remarks

A data dependency check is not performed with the P register. To execute subsequent instructions after the results of the EATANxy instruction are written to the P register, use the WAITP instruction for synchronization.

EATANxz : Arctangent

Calculates the arctangent based on the x, z fields of VF[fs] and stores in the P register.

Operation Code

Lower 32-bit word: LowerOP field type 3

Lower OP. (31–25)	dest (24–21)	ft reg (20–16)	fs reg (15–11)	EATANxz (10–00)		
1000000	1010	00000	-----	11101	1111	01
7 bits	4 bits	5 bits	5 bits	11 bits		

Mnemonic

EATANxz P, VF[fs**]**

Operation

P = arctan(VF[fs]z / VF[fs]x)

The following approximation formula is used for calculating arctan.

Calculation is valid for: 0 <= z <= x (Excluding 0 = z =x)

$$\arctan(x) = (T_1 \times t + T_2 \times t^3 + T_3 \times t^5 + T_4 \times t^7 + T_5 \times t^9 + T_6 \times t^{11} + T_7 \times t^{13} + T_8 \times t^{15}) + \frac{\pi}{4}$$

Provided $t = \frac{(z - x)}{(z + x)}$

Constants	Decimal Expressions	Single precision floating-point expressions			Hex. expressions
		S	E	F	
T_1	0.999999344348907	0	01111110	11111111111111111110101	3f7ffff5
T_2	-0.333298563957214	1	01111101	01010101010011000011100	beaaa61c
T_3	0.199465364217758	0	01111100	10011000100000010100110	3e4c40a6
T_4	-0.139085337519646	1	01111100	00011100110110001100011	be0e6c63
T_5	0.096420042216778	0	01111011	10001010111011111011111	3dc577df
T_6	-0.055909886956215	1	01111010	11001010000000111000100	bd6501c4
T_7	0.021861229091883	0	01111001	01100110001011001010010	3cb31652
T_8	-0.004054057877511	1	01110111	00001001101011111100111	bb84d7e7
$\pi/4$	0.785398185253143	0	01111110	10010010000111111011011	3f490fdb

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

53 / 54

Example

```
EATANxz P, VF10
```

P = arctan(VF10z / VF10x)

Remarks

A data dependency check is not performed with the P register. To execute subsequent instructions after the results of the EATANxz instruction are written to the P register, use the WAITP instruction for synchronization.

EEXP : Exponent

Calculates the exponent of fsf field of VF[fs] and stores the result in the P register.

Operation Code

Lower 32-bit word: LowerOP field type 4

Lower OP. (31–25)	ftf (24–23)	fsf (22–21)	ft reg (20–16)	fs reg (15–11)	EEXP (10–00)
1000000	00	--	00000	-----	11111 1111 10
7 bits	2 bits	2 bits	5 bits	5 bits	11 bits

Mnemonic

EEXP P, VF[fs**]**fsf

Operation

P = exp(- VF[fs]fsf **)**

The following approximation formula is used for calculating exp.

Calculation is valid for: 0 <= x <= +MAX

$$e^{-x} = \frac{1}{(1+E_1 \times x + E_2 \times x^2 + E_3 \times x^3 + E_4 \times x^4 + E_5 \times x^5 + E_6 \times x^6)^4}$$

Constants	Decimal Expressions	Single precision floating-point expressions			Hex. expressions
		S	E	F	
E_1	0.249998688697815	0	01111100	11111111111111110101000	3e7fffa8
E_2	0.031257584691048	0	01111010	00000000000011111110100	3d0007f4
E_3	0.002591371303424	0	01110110	01010011101001111111111	3b29d3ff
E_4	0.000171562001924	0	01110010	01100111110010101010011	3933e553
E_5	0.000005430199963	0	01101101	01101100011010100010000	36b63510
E_6	0.000000690600018	0	01101010	01110010110000110101100	353961ac

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

43 / 44

Example

```
EEXP P, VF10x
```

P = exp(- VF10x)

Remarks

A data dependency check is not performed with the P register. To execute subsequent instructions after the results of the EEXP instruction are written to the P register, use the WAITP instruction for synchronization.

ELENG : Length

Calculates the length from the origin, with the x, y, z fields of VF[fs] as the three-dimensional coordinates, and stores the result in the P register.

Operation Code

Lower 32-bit word: LowerOP field type 3

31–25	24–21	20–16	15–11	10–00		
Lower OP.	dest	ft reg	fs reg	ELENG		
1000000	1110	00000	-----	11100	1111	10
7 bits	4 bits	5 bits	5 bits	11 bits		

Mnemonic

ELENG P, VF[fs**]**

Operation

$$P = \sqrt{\{(VF[fs]x)^2+(VF[fs]y)^2+(VF[fs]z)^2\}}$$

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

17 / 18

Example

```
ELENG P, VF10
```

$$P = \sqrt{\{(VF10x)^2+(VF10y)^2+(VF10z)^2\}}$$

Remarks

A data dependency check is not performed with the P register. To execute subsequent instructions after the results of the ELENG instruction are written to the P register, use the WAITP instruction for synchronization.

ERCPR : Reciprocal Number

Calculates the reciprocal of the fsf field of VF[fs] and stores it in the P register.

Operation Code

Lower 32-bit word: LowerOP field type 4

31–25	24–23	22–21	20–16	15–11	10–00		
Lower OP.	ftf	fsf	ft reg	fs reg	ERCPR		
1000000	00	--	00000	-----	11110	1111	10
7 bits	2 bits	2 bits	5 bits	5 bits	11 bits		

Mnemonic

ERCPR P, VF[fs]fsf

Operation

P = 1 / VF[fs]fsf

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

11 / 12

Example

```
ERCPR P, VF10x
```

P = 1 / VF10x

Remarks

A data dependency check is not performed with the P register. To execute subsequent instructions after the results of the ERCPR instruction are written to the P register, use the WAITP instruction for synchronization.

ERLENG : Reciprocal Number of Length

Calculates the reciprocal of the length from the origin, using the x, y, z fields of VF[fs] as the three-dimensional coordinates, and stores the result in the P register.

Operation Code

Lower 32-bit word: LowerOP field type 3

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

Lower OP.	dest	ft reg	fs reg	ERLENG		
1000000	1110	00000	-----	11100	1111	11
7 bits	4 bits	5 bits	5 bits	11 bits		

Mnemonic

ERLENG P, VF[fs**]**

Operation

$$\mathbf{P = \frac{1}{\sqrt{\{(VF[fs]x)^2+(VF[fs]y)^2+(VF[fs]z)^2\}}}}$$

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

23 / 24

Example

```
ERLENG P, VF10
```

$$P = \frac{1}{\sqrt{\{(VF10x)^2+(VF10y)^2+(VF10z)^2\}}}$$

Remarks

A data dependency check is not performed with the P register. To execute subsequent instructions after the results of the ERLENG instruction are written to the P register, use the WAITP instruction for synchronization.

ERSADD : Reciprocal Number

Calculates the reciprocal of the sum of the squares of the x, y, z fields of VF[fs] and stores the result in the P register.

Operation Code

Lower 32-bit word: LowerOP field type 3

Lower OP. (31–25)	dest (24–21)	ft reg (20–16)	fs reg (15–11)	ERSADD (10–00)		
1000000	1110	00000	-----	11100	1111	01
7 bits	4 bits	5 bits	5 bits	11 bits		

Mnemonic

ERSADD P, VF[fs**]**

Operation

$P = 1 / \{ (VF[fs]x)^2 + (VF[fs]y)^2 + (VF[fs]z)^2 \}$

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

17 / 18

Example

```
ERSADD P, VF10
```

$P = 1 / \{ (VF10x)^2 + (VF10y)^2 + (VF10z)^2 \}$

Remarks

A data dependency check is not performed with the P register. To execute subsequent instructions after the results of the ERSADD instruction are written to the P register, use the WAITP instruction for synchronization.

ERSQRT : Reciprocal Number of Square Root

Calculates the reciprocal of the square root of the fsf field of VF[fs] and stores the result in the P register.

Operation Code

Lower 32-bit word: LowerOP field type 4

31–25	24–23	22–21	20–16	15–11	10–00		
Lower OP.	ftf	fsf	ft reg	fs reg	ERSQRT		
1000000	00	--	00000	-----	11110	1111	01
7 bits	2 bits	2 bits	5 bits	5 bits	11 bits		

Mnemonic

ERSQRT P, VF[fs**]**$_{fsf}$

Operation

$$\mathbf{P} = \frac{1}{\sqrt{\mathbf{VF[fs]}_{fsf}}}$$

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

17 / 18

Example

`ERSQRT P, VF10x`

$$P = \frac{1}{\sqrt{VF10x}}$$

Remarks

A data dependency check is not performed with the P register. To execute subsequent instructions after the results of the ERSQRT instruction are written to the P register, use the WAITP instruction for synchronization.

ESADD : Sum of Square Numbers

Calculates the sum of the squares of the x, y, z fields of VF[fs] and stores the result in the P register.

Operation Code

Lower 32-bit word: LowerOP field type 3

31–25	24–21	20–16	15–11	10–00
Lower OP.	dest	ft reg	fs reg	ESADD
1000000	1110	00000	-----	11100 1111 00
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

ESADD P, VF[fs**]**

Operation

$P = (VF[fs]x)^2 + (VF[fs]y)^2 + (VF[fs]z)^2$

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

10 / 11

Example

```
ESADD P, VF10
```

$P = (VF10x)^2 + (VF10y)^2 + (VF10z)^2$

Remarks

A data dependency check is not performed with the P register. To execute subsequent instructions after the results of the ESADD instruction are written to the P register, use the WAITP instruction for synchronization.

ESIN : Sine

Calculates the sine of the fsf field of VF[fs] and stores the result in the P register.

Operation Code

Lower 32-bit word: LowerOP field type 4

31–25	24–23	22–21	20–16	15–11	10–00		
Lower OP.	ftf	fsf	ft reg	fs reg	ESIN		
1000000	00	--	00000	-----	11111	1111	00
7 bits	2 bits	2 bits	5 bits	5 bits	11 bits		

Mnemonic

ESIN P, VF[fs**]**fsf

Operation

P = sin(VF[fs]fsf **)**

The following approximation formula is used for calculating sin.

Calculation is valid for: -π/2 <= x <= +π/2

$$\sin(x) = S_1 \times x + S_2 \times x^3 + S_3 \times x^5 + S_4 \times x^7 + S_5 \times x^9$$

Constants	Decimal Expressions	Single precision floating-point expressions			Hex. expressions
		S	E	F	
S_1	1.00000000000000	0	01111111	00000000000000000000000	3f800000
S_2	-0.166666567325592	1	01111100	01010101010101010100100	be2aaaa4
S_3	0.008333025500178	0	01111000	00010001000011100111110	3c08873e
S_4	-0.000198074136279	1	01110010	10011111011001000011111	b94fb21f
S_5	0.000002601886990	0	01101100	01011101001110000010100	362e9c14

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

28 / 29

Example

```
ESIN P, VF10x
```

P = sin(VF10x)

Remarks

A data dependency check is not performed with the P register. To execute subsequent instructions after the results of the ESIN instruction are written to the P register, use the WAITP instruction for synchronization.

ESQRT : Square Root

Calculates the square root of the fsf field of VF[fs] and stores the result in the P register.

Operation Code

Lower 32-bit word: LowerOP field type 4

31–25	24–23	22–21	20–16	15–11	10–00		
Lower OP.	ftf	fsf	ft reg	fs reg	ESQRT		
1000000	00	--	00000	-----	11110	1111	00
7 bits	2 bits	2 bits	5 bits	5 bits	11 bits		

Mnemonic

ESQRT P, VF[fs**]**fsf

Operation

$P = \sqrt{VF[fs]_{fsf}}$

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

11 / 12

Example

```
ESQRT P, VF10x
```

$P = \sqrt{VF10x}$

Remarks

A data dependency check is not performed with the P register. To execute subsequent instructions after the results of the ESQRT instruction are written to the P register, use the WAITP instruction for synchronization.

ESUM : Sum of Each Field

Calculates the total sum of the four fields of VF[fs] and stores the result in the P register.

Operation Code

Lower 32-bit word: LowerOP field type 3

31–25	24–21	20–16	15–11	10–00
Lower OP.	dest	ft reg	fs reg	ESUM
1000000	1111	00000	-----	11101 1111 10
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

ESUM P, VF[fs**]**

Operation

P = VF[fs]x + VF[fs]y + VF[fs]z + VF[fs]w

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

11 / 12

Example

```
ESUM P, VF10
```

P = VF10x + VF10y + VF10z + VF10w

Remarks

A data dependency check is not performed with the P register. To execute subsequent instructions after the results of the ESUM instruction are written to the P register, use the WAITP instruction for synchronization.

FCAND : Test Clipping Flag

Tests the value of the clipping flag and stores the result in VI01.

Operation Code

Lower 32-bit word: LowerOP field type 9

31–25	24	23–00
FCAND 0010010	- 0	Imm24 ------------------------
7 bits	1	24 bits

Mnemonic

FCAND VI01, Imm24

Operation

Calculates the AND (logical product) of the clipping flag and Imm24 for every bit. A 1 is written to VI01 if there is at least one 1 in the results; a 0 is written to VI01 if there are no 1's.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 1

Example

; Checks if there is "+x" even only one time in the 4 previous clipping tests or not.

```
NOP    FCAND VI01, 0x041041
NOP    IBEQ  VI01, VI00, NEVER_CLIP_X
NOP    B     CLIP_X
```

FCEQ : Test Clipping Flag

Tests the value of clipping flag and stores the result in VI01.

Operation Code

Lower 32-bit word: LowerOP field type 9

31–25	24	23–00
FCEQ	-	Imm24
0010000	0	------------------------
7 bits	1	24 bits

Mnemonic

FCEQ VI01, Imm24

Operation

Compares clipping flag and Imm24. A 1 is written to VI01 if they are the same; a 0 is written if they are not.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 1

Example

; Checks if all of the results of the 4 previous clipping tests are "+x" or not.

```
NOP     FCEQ    VI01, 0x041041
NOP     IBEQ    VI01, VI00, IN_RANGE_X
NOP     B       OUT_OF_RANGE_X
```

FCGET : Get Clipping Flag

Stores a part of the values of the clipping flag in VI[it].

Operation Code

Lower 32-bit word: LowerOP field type 8

31–25	24–21	20–16	15–11	10–00
FCGET 0011100	Imm15 0000	it reg -----	is reg 00000	Imm15 00000000000
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

FCGET VI[it**]**

Operation

The lower 12 bits of the clipping flag (information from the two most recent checks) are written into VI[it]. 0 is written to the upper 4 bits of VI[it].

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 1

Example

```
FCGET VI01
```

FCOR : Test Clipping Flag

Tests the value of clipping flag and stores the result in VI01.

Operation Code

Lower 32-bit word: LowerOP field type 9

31–25	24	23–00
FCOR	-	Imm24
0010011	0	------------------------
7 bits	1	24 bits

Mnemonic

FCOR VI01, Imm24

Operation

Calculates the OR (logical sum) of the clipping flag and Imm24 at every bit. A 1 is written into VI01 if all of the results are 1, and 0 is written if they are not all 1's.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 1

Example

```
FCOR VI01, 0xffdf7d  ;B1111,1111,1101,1111,0111,1101
```

VI01 becomes 1 if all of the results of the 3 previous clipping tests are "-x."

FCSET : Setting Clipping Flag

Sets the value of the clipping flag.

Operation Code

Lower 32-bit word: LowerOP field type 9

31–25	24	23–00
FCSET 0010001	- 0	Imm24 ------------------------
7 bits	1	24 bits

Mnemonic

FCSET Imm24

Operation

Sets Imm24 to the clipping flag. It is set is at the S stage of the basic pipeline.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	X

Throughput/Latency

1 / 4

Example

```
FCSET 0x000000     ;B0000,0000,0000,0000,0000,0000
```

Clears the clipping flag.

FMAND : Test MAC Flag Check

Calculates the logical product of the MAC flag and VI[is] and stores the result in VI[it].

Operation Code

Lower 32-bit word: LowerOP field type 8

31–25	24–21	20–16	15–11	10–00
FMAND 0011010	Imm15 0000	it reg -----	is reg -----	Imm15 00000000000
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

FMAND VI[it**], VI[**is**]**

Operation

Calculates the AND (logical product) of the MAC flag and VI[is] at every bit and stores the result in VI[it].

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 1

Example

; Branches if the Z field of the calculation result is negative.

```
NOP     IADDIU      VI10, VI00, 0x0020
NOP     FMAND       VI01, VI10
NOP     IBNE        VI01, VI00, Z_MINUS
```

FMEQ : Test MAC Flag Check

Compares the MAC flag and VI[is] and stores the result in VI[it].

Operation Code

Lower 32-bit word: LowerOP field type 8

31–25	24–21	20–16	15–11	10–00
FMEQ 0011000	Imm15 0000	it reg -----	is reg -----	Imm15 00000000000
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

FMEQ VI[it**], VI[**is**]**

Operation

Writes 1 into VI[it] if the values of the MAC flag and VI[is] are the same, and writes 0 if they are not.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 1

Example

; Branches if only the Z field of the calculation result is negative.

```
NOP     IADDIU      VI10, VI00, 0x0020
NOP     FMEQ        VI01, VI10
NOP     IBNE        VI01, VI00, Z_MINUS
```

FMOR : Test MAC Flag Check

Calculates the logical sum of the MAC flag and VI[is] and stores the result in VI[it].

Operation Code

Lower 32-bit word: LowerOP field type 8

31–25	24–21	20–16	15–11	10–00
FMOR 0011011	Imm15 0000	it reg -----	is reg -----	Imm15 00000000000
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

FMOR VI[it**], VI[**is**]**

Operation

Calculates the OR (logical sum) of the MAC flag and VI[is] at every bit and stores the result in VI[it] .

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 1

Example

```
FMOR VI01, VI10
```

FSAND : Test Status Flag Check

Calculates the logical product (AND) of the status flag and the immediate value and stores the result in VI[it].

Operation Code

Lower 32-bit word: LowerOP field type 8

31–25	24–21	20–16	15–11	10–00
FSAND 0010110	Imm12 000-	it reg -----	is reg 00000	Imm12 -----------
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

FSAND VI[it**],** Imm12

Operation

Calculates the logical product (AND) of the status flag and Imm12 at every bit and stores the result in VI[it].

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 1

Example

; Checks if there was an overflow in the past.

```
NOP     FSAND  VI01, 0x0200
NOP     IBNE   VI01, VI00, OVERFLOW
```

FSEQ : Test Status Flag Check

Compares the status flag and the immediate value and stores the result in VI[it].

Operation Code

Lower 32-bit word: LowerOP field type 8

31–25	24–21	20–16	15–11	10–00
FSEQ	Imm12	it reg	is reg	Imm12
0010100	000-	-----	00000	-----------
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

FSEQ VI[it**],** Imm12

Operation

Writes 1 into VI[it] if the values of the status flag and Imm12 are the same, and writes 0 if they are not.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 1

Example

; Checks if there was an overflow immediately before.

```
NOP     FSEQ    VI01, 0x0008
NOP     IBNE    VI01, VI00, OVERFLOW
```

FSOR : Test Status Flag

Calculates the logical sum of the status flag and the immediate value and stores the result in VI[it].

Operation Code

Lower 32-bit word: LowerOP field type 8

31–25	24–21	20–16	15–11	10–00
FSOR 0010111	Imm12 000-	it reg -----	is reg 00000	Imm12 -----------
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

FSOR VI[it**],** Imm12

Operation

Calculates the logical sum (OR) of the status flag and Imm12 at every bit and stores the result in VI[it].

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 1

Example

```
FSOR VI01, chk_flag
```

FSSET : Set Sticky Flags

Writes the immediate value to the sticky flags. (DS-ZS)

Operation Code

Lower 32-bit word: LowerOP field type 8

31–25	24–21	20–16	15–11	10–00
FSSET 0010101	Imm12 000-	it reg 00000	is reg 00000	Imm12 -----000000
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

```
FSSET Imm12
```

Operation

Writes the upper 6 bits of Imm 12 to the upper 6 bits (sticky flag) of the 12-bit status flag. The lower 6 bits of Imm 12 are not set, and are therefore ignored. It is set at the S stage of the basic pipleline.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	X	X	X	X	X	X	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
FSSET 0x0
```

Clears all the sticky flag.

IADD : ADD Integer

Adds VI[is] and VI[it] and stores the result in VI[id].

Operation Code

Lower 32-bit word: LowerOP field type 1

Lower OP. (31–25)	dest (24–21)	it reg (20–16)	is reg (15–11)	id reg (10–06)	IADD (05–00)
1000000	0000	-----	-----	-----	110000
7 bits	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

```
IADD VI[id], VI[is], VI[it]
```

Operation

VI[id] = VI[is] + VI[it]

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 1

Example

```
IADD VI04, VI05, VI06
```

VI04 = VI05 + VI06

IADDI : Add Immediate Value Integer

Adds the immediate value to VI[is] and stores the result in VI[it].

Operation Code

Lower 32-bit word : LowerOP field type 5

Lower OP. (31–25)	dest (24–21)	it reg (20–16)	is reg (15–11)	Imm5 (10–06)	IADDI (05–00)
1000000	0000	-----	-----	-----	110010
7 bits	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

IADDI VI[it**], VI[**is**],** Imm5

Operation

VI[it] = VI[is] + Imm5

Imm5 is considered as a signed integer.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 1

Example

```
IADDI VI04, VI05, -1
```

VI04 = VI05 - 1

IADDIU : Add Immediate Integer

Adds the immediate value to VI[is] and stores the result in VI[it].

Operation Code

Lower 32-bit word: LowerOP field type 8

31–25	24–21	20–16	15–11	10–00
IADDIU 0001000	Imm15 ----	it reg -----	is reg -----	Imm15 ---------------
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

IADDIU VI[it**], VI[**is**],** Imm15

Operation

VI[it] = VI[is] + Imm15

Imm15 is considered as a unsigned integer.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 1

Example

```
IADDIU VI04, VI05, 1
```

VI04 = VI05 + 1

IAND : Logical Product

Calculates the AND (logical product) of VI[is] and VI[it] at every bit and stores the result in VI[id].

Operation Code

Lower 32-bit word: LowerOP field type 1

31–25	24–21	20–16	15–11	10–06	05–00
Lower OP. 1000000	dest 0000	it reg -----	is reg -----	id reg -----	IAND 110100
7 bits	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

IAND VI[id**], VI[**is**], VI[**it**]**

Operation

VI[id] = VI[is] AND VF[it]

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 1

Example

```
IAND VI04, VI05, VI06
```

VI04 = VI05 AND VI06

IBEQ : Conditional Branch

Branches to the relative address specified by the immediate value if the contents of VI[is] and VI[it] are equal.

Operation Code

Lower 32-bit word: LowerOP field type 7

IBEQ (31–25)	dest (24–21)	it reg (20–16)	is reg (15–11)	Imm11 (10–00)
0101000	0000	-----	-----	-----------
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

IBEQ VI[it**], VI[**is**],** Imm11

Imm11 is an 11-bit signed integer and holds the value of offset to the branch destination by 8.

Operation

if (VI[it] == VI[is]) then jump (PC+Imm11 × 8)

An 11-bit signed offset Imm11 multiplied by 8 is added to the address of the instruction following the branch (NOT the branch itself), in the branch delay slot, to form a PC-relative effective target address.
If the contents of VI[it] and VI[is] are equal, a branch to the PC-relative effective target address after the instruction in the delay slot is executed.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

2 / 2

Example

```
IBEQ VI04, VI05, LABEL:
```

Remarks

This instruction cannot be placed in the E bit delay slot.
After the instruction for setting the value to VI[is] or VI[it], leave a slot for one instruction, then write IBEQ instruction. Refer to "3.4.8. Conditional Branching and Pipeline."

IBGEZ : Conditional Branch

Branches to the relative address specified by the immediate value if the contents of VI[is] is greater than or equal to 0.

Operation Code

Lower 32-bit word: LowerOP field type 7

31–25	24–21	20–16	15–11	10–00
IBGEZ 0101111	dest 0000	it reg 00000	is reg -----	Imm11 -----------
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

IBGEZ VI[is**],** Imm11

Imm11 is an 11-bit signed integer and holds the value of offset to the branch target address divided by 8.

Operation

if (VI[is] >= 0) then jump (PC+Imm11 × 8)

An 11-bit signed offset Imm11 multiplied by 8 is added to the address of the instruction following the branch (NOT the branch itself), in the branch delay slot, to form a PC-relative effective target address.
If the contents of VI[is] are greater than or equal to 0, a branch to the PC-relative effective target address after the instruction in the delay slot is executed.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

2 / 2

Example

```
IBGEZ VI04, LABEL:
```

Remarks

This instruction cannot be placed in the E bit delay slot.
After an instruction that writes a value to VI[is], leave a slot for one instruction before an IBGEZ instruction. Refer to "3.4.8. Conditional Branching and Pipeline."

IBGTZ : Conditional Branch

Branches to the relative address specified by the immediate value if the value of VI[is] is greater than 0.

Operation Code

Lower 32-bit word: LowerOP field type 7

31–25	24–21	20–16	15–11	10–00
IBGTZ 0101101	dest 0000	it reg 00000	is reg -----	Imm11 -----------
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

IBGTZ VI[is**],** Imm11

Imm11 is an 11-bit signed integer and holds the value of offset to the branch target address divided by 8.

Operation

if (VI[is] > 0) then jump (PC+Imm11 × 8)

An 11-bit signed offset Imm11 multiplied by 8 is added to the address of the instruction following the branch (NOT the branch itself), in the branch delay slot, to form a PC-relative effective target address. If the contents of VI[is] are greater than 0, a branch to the PC-relative effective target address after the instruction in the delay slot is executed.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

2 / 2

Example

```
IBGTZ VI04, LABEL:
```

Remarks

This instruction cannot be placed in the E bit delay slot.
After an instruction that writes a value to VI[is], leave a slot for one instruction before the IBGTZ instruction. Refer to "3.4.8. Conditional Branching and Pipeline."

IBLEZ : Conditional Branch

Branches to the relative address specified by the immediate value if the value of VI[is] is less than or equal to 0.

Operation Code

Lower 32-bit word: LowerOP field type 7

IBLEZ (bits 31–25)	dest (bits 24–21)	it reg (bits 20–16)	is reg (bits 15–11)	Imm11 (bits 10–00)
0101110	0000	00000	-----	-----------
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

IBLEZ VI[is**],** Imm11

Imm11 is an 11-bit signed integer and holds the value of offset to branch destination by 8.

Operation

if (VI[is] <= 0) then jump (PC+Imm11 × 8)

An 11-bit signed offset Imm11 multiplied by 8 is added to the address of the instruction following the branch (NOT the branch itself), in the branch delay slot, to form a PC-relative effective target address.
If the contents of VI[is] are less than or equal to 0, a branch to the PC-relative effective target address after the instruction in the delay slot is executed.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

2 / 2

Example

```
IBLEZ VI04, LABEL:
```

Remarks

This instruction cannot be placed in the E bit delay slot.
After an instruction that writes a value to VI[is], leave a slot for one instruction before the IBLEZ instruction. Refer to "3.4.8. Conditional Branching and Pipeline."

IBLTZ : Conditional Branch

Branches to the relative address specified by the immediate value, if the contents of VI[is] are less than 0.

Operation Code

Lower 32-bit word: LowerOP field type 7

31–25	24–21	20–16	15–11	10–00
IBLTZ 0101100	dest 0000	it reg 00000	is reg -----	Imm11 -----------
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

IBLTZ VI[is**],** Imm11

Imm11 is an 11-bit signed integer and holds the value of offset to the branch target address divided by 8.

Operation

if (VI[is] < 0) then jump (PC+Imm11 × 8)

An 11-bit signed offset Imm11 multiplied by 8 is added to the address of the instruction following the branch (NOT the branch itself), in the branch delay slot, to form a PC-relative effective target address. If the contents of VI[is] are less than 0, a branch to the PC-relative effective target address after the instruction in the delay slot is executed.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

2 / 2

Example

```
IBLTZ VI04, LABEL:
```

Remarks

This instruction cannot be placed in the E bit delay slot.
After an instruction that writes a value to VI[is], leave a slot for one instruction before the IBLTZ instruction. Refer to "3.4.8. Conditional Branching and Pipeline."

IBNE : Conditional Branch

Branches to the PC relative address determined by the immediate value, if the contents of VI[is] and VI[it] are not equal.

Operation Code

Lower 32-bit word: LowerOP field type 7

31–25	24–21	20–16	15–11	10–00
IBNE 0101001	dest 0000	it reg -----	is reg -----	Imm11 -----------
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

IBNE VI[it**], VI[**is**],** Imm11

Imm11 is an 11-bit signed integer and holds the value of offset to the branch target address divided by 8.

Operation

if (VI[it] != VI[is]) then jump (PC+Imm11 × 8)

An 11-bit signed offset Imm11 multiplied by 8 is added to the address of the instruction following the branch (NOT the branch itself), in the branch delay slot, to form a PC-relative effective target address.
If the contents of VI[is] and VI[it] are not equal, a branch to the PC-relative effective target address after the instruction in the delay slot is executed.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

2 / 2

Example

```
IBNE VI04, VI05, LABEL:
```

Remarks

This instruction cannot be placed in the E bit delay slot.
After an instruction that writes values to VI[is] and VI[it], leave a slot for one instruction before the IBNE instruction. Refer to "3.4.8. Conditional Branching and Pipeline."

ILW : Integer Load with Offset Specification

Loads the specific field of data, whose address is specified by VI[is] and the immediate value, to VI[it] from VU Mem.

Operation Code

Lower 32-bit word: LowerOP field type 7

31–25	24–21	20–16	15–11	10–00
ILW 0000100	dest ----	it reg -----	is reg -----	Imm11 -----------
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

ILW.dest **VI[**it**],** Imm11**(VI[**is**])**dest

Specifies any one of x, y, z or w for dest. Operation is undefined when multiple specifications are made. Imm11 is an 11-bit signed integer. For Imm11 and VI[is], specifies the address divided by 16.

Operation

VI[it] = VU_MEM$_{dest}$**((Imm11 + VI[is]) × 16)**

Loads the lower 16 bits of dest field of data located in VU Mem, whose address is determined with Imm11 and VI[is] to VI[it].

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
ILW.y VI10, offset(VI04)y
```

VI10 = VU_MEMy(offset + VI04)

Remarks

This instruction cannot be placed in the E bit delay slot.

ILWR : Integer Load

Loads the specific field of data, whose address is specified by VI[is], to VI[it] from VU Mem.

Operation Code

Lower 32-bit word: LowerOP field type 3

Lower OP. (31–25)	dest (24–21)	it reg (20–16)	is reg (15–11)	ILWR (10–00)		
1000000	----	-----	-----	01111	1111	10
7 bits	4 bits	5 bits	5 bits	11 bits		

Mnemonic

ILWR.dest **VI[**it**], (VI[**is**])**dest

Specifies any one of x, y, z or w for dest. Operation is undefined when multiple specifications are made. For VI[is], specifies the address divided by 16.

Operation

VI[it] = VU_MEM$_{dest}$**(VI[is] × 16)**

Loads lower 16 bits of dest field of data located in VU Mem, whose address is determined by VI[is] to VI[it].

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
ILWR.y VI10, (VI04) y
```

VI10 = VU_MEMy(VI04)

Remarks

This instruction cannot be placed in the E bit delay slot.

IOR : Logical Sum

Calculates the logical sum of VI[is] and VI[it] at every bit and stores the result in VI[id].

Operation Code

Lower 32-bit word: LowerOP field type 1

31–25	24–21	20–16	15–11	10–06	05–00
Lower OP.	dest	it reg	is reg	id reg	IOR
1000000	0000	-----	-----	-----	110101
7 bits	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

IOR VI[id**], VI[**is**], VI[**it**]**

Operation

VI[id] = VI[is] OR VI[it]

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 1

Example

```
IOR VI04, VI05, VI06
```

VI04 = VI05 OR VI06

ISUB : Integer Subtract

Subtracts VI[it] from VI[is] and stores the result in VI[id].

Operation Code

Lower 32-bit word: LowerOP field type 1

Lower OP. (31–25)	dest (24–21)	it reg (20–16)	is reg (15–11)	id reg (10–06)	ISUB (05–00)
1000000	0000	-----	-----	-----	110001
7 bits	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

ISUB VI[id**], VI[**is**], VI[**it**]**

Operation

VI[id] = VI[is] - VI[it]

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 1

Example

```
ISUB VI04, VI05, VI06
```

VI04 = VI05 - VI06

ISUBIU : Immediate Value Integer Subtract

Subtracts the immediate value from VI[is] and stores the result in VI[it].

Operation Code

Lower 32-bit word: LowerOP field type 8

31–25	24–21	20–16	15–11	10–00
ISUBIU 0001001	Imm15 ----	it reg -----	is reg -----	Imm15 ---------------
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

ISUBIU VI[it**], VI[**is**],** Imm15

Operation

VI[it] = VI[is] - Imm15

Imm15 is considered as unsigned integer.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 1

Example

```
ISUBIU VI04, VI05, 1
```

VI04 = VI05 - 1

ISW : Integer Store with Offset

Stores data from VI[it] in VU Mem. The destination is the field specified with dest of the address determined by VI[is] and the immediate value.

Operation Code

Lower 32-bit word: LowerOP field type 7

31–25	24–21	20–16	15–11	10–00
ISW 0000101	dest ----	it reg -----	is reg -----	Imm11 -----------
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

`ISW.dest VI[it], Imm11(VI[is])dest`

Imm11 is a 11-bit signed integer. For Imm11 and VI[is], specifies the value obtained by dividing the address by 16.

Operation

if (x ⊆ dest) then VU_MEMx(Imm11 + VI[is]) = VI[it]
if (y ⊆ dest) then VU_MEMy(Imm11 + VI[is]) = VI[it]
if (z ⊆ dest) then VU_MEMz(Imm11 + VI[is]) = VI[it]
if (w ⊆ dest) then VU_MEMw(Imm11 + VI[is]) = VI[it]

The value of VI[it] is stored in the lower 16 bits of dest field of data located in VU Mem, whose address is determined by Imm11 and VI[is]. The upper 16 bits are filled with 0.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

`ISW.y VI10, offset(VI04)y`

VU_MEMy(offset + VI04) = VI10

Remarks

This instruction cannot be placed in the E bit delay slot.

ISWR : Integer Store

Stores data from VI[it] in VU Mem. The destination is the field specified with dest of the address determined by VI[is].

Operation Code

Lower 32-bit word: LowerOP field type 3

31–25	24–21	20–16	15–11	10–00
Lower OP. 1000000	dest ----	it reg -----	is reg -----	ISWR 01111 1111 11
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

ISWR.dest **VI[**it**], (VI[**is**])**dest

For VI[is], specifies the value obtained by dividing the address by 16.

Operation

if (x ⊆ dest) then VU_MEMx(VI[is]) = VI[it]
if (y ⊆ dest) then VU_MEMy(VI[is]) = VI[it]
if (z ⊆ dest) then VU_MEMz(VI[is]) = VI[it]
if (w ⊆ dest) then VU_MEMw(VI[is]) = VI[it]

The value of VI[it] is stored in lower 16 bits of dest field of data located in VU Mem, whose address is determined by VI[is]. The upper 16 bits are filled with 0.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
ISWR.y VI10, (VI04)y
```

VU_MEMy(VI04) = VI10

Remarks

This instruction cannot be placed in the E bit delay slot.

JALR : Unconditional Jump with Address Saving

Stores the instruction address in VI[it] and jumps to the address specified with VI[is].

Operation Code

Lower 32-bit word: LowerOP field type 7

31–25	24–21	20–16	15–11	10–00
JALR 0100101	dest 0000	it reg -----	is reg -----	Imm11 00000000000
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

JALR VI[it**], VI[**is**]**

For VI[is], specifies the branch destination address aligned on a dword boundary.

Operation

VI[it] = PC + (2 × 8)
PC = VI[is]

The branch delay slot is one instruction.

The addresss of the next instruction after the branch delay slot is stored in VI[it].

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

2 / 2

Example

; Calls a subroutine indicated with the label SYMBOL

```
IADDIU VI04, VI00, SYMBOL
NOP
JALR VI15, VI04
```

Remarks

This instruction cannot be placed in the E bit delay slot.

JR : Unconditional Jump

Jumps to the address specified with VI[is].

Operation Code

Lower 32-bit word: LowerOP field type 7

31–25	24–21	20–16	15–11	10–00
JR 0100100	dest 0000	it reg 00000	is reg -----	Imm11 00000000000
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

JR VI[is**]**

As the branch destination, specifies the address in dword alignment for VI[is].

Operation

It jumps to the address specified with VI[is]. The branch delay slot is one instruction.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

2 / 2

Example

```
JR VI04
```

Branches to the address indicated by VI04.

Remarks

This instruction cannot be placed in the E bit delay slot.

LQ : Load Qword

Loads data from a qword, whose address is specified with VI[is] and the immediate value, to VF[ft] from VU Mem.

Operation Code

Lower 32-bit word: LowerOP field type 7

31–25	24–21	20–16	15–11	10–00
LQ 0000000	dest ----	ft reg -----	is reg -----	Imm11 -----------
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

LQ.dest **VF[**ft**]**dest, Imm11**(VI[**is**])**

Imm11 is a 11-bit signed integer.

For Imm11 and VI[is], specifies the value obtained by dividing the address by 16.

Operation

if (x ⊆ dest) then VF[ft]x = VU_MEMx((Imm11 + VI[is]) × 16)
if (y ⊆ dest) then VF[ft]y = VU_MEMy((Imm11 + VI[is]) × 16)
if (z ⊆ dest) then VF[ft]z = VU_MEMz((Imm11 + VI[is]) × 16)
if (w ⊆ dest) then VF[ft]w = VU_MEMw((Imm11 + VI[is]) × 16)

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
LQ.xyzw VF10xyzw, offset(VI04)
```

VF10x = VU_MEMx((offset + VI04) × 16)
VF10y = VU_MEMy((offset + VI04) × 16)
VF10z = VU_MEMz((offset + VI04) × 16)
VF10w = VU_MEMw((offset + VI04) × 16)

Remarks

This instruction cannot be placed in the E bit delay slot.

When an Upper instruction in the same cycle writes data to the VF[ft] register, the result of this instruction is discarded with priority given to the Upper instruction, regardless of whether the data is written to the same field or not.

LQD : Load Qword with Pre-Decrement

Subtracts 1 from VI[is] and loads data from a qword specified by VI[is] from VU Mem to VF[ft].

Operation Code

Lower 32-bit word: LowerOP field type 3

31–25	24–21	20–16	15–11	10–00
Lower OP. 1000000	dest ----	ft reg -----	is reg -----	LQD 01101 1111 10
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

LQD.dest **VF[**ft**]**dest, **(--VI[**is**])**

For VI[is], specifies the value obtained by dividing the address by 16.

Operation

VI[is] = VI[is] - 1
if (x ⊆ dest) then VF[ft]x = VU_MEMx(VI[is] × 16)
if (y ⊆ dest) then VF[ft]y = VU_MEMy(VI[is] × 16)
if (z ⊆ dest) then VF[ft]z = VU_MEMz(VI[is] × 16)
if (w ⊆ dest) then VF[ft]w = VU_MEMw(VI[is] × 16)

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
LQD.xyzw VF10xyzw, (--VI04)
```

VI04 = VI04 - 1
VF10x = VU_MEMx(VI04 × 16)
VF10y = VU_MEMy(VI04 × 16)
VF10z = VU_MEMz(VI04 × 16)
VF10w = VU_MEMw(VI04 × 16)

Remarks

This instruction cannot be placed in the E bit delay slot.
When an Upper instruction in the same cycle writes data to the VF[ft] register, the result of this instruction is discarded with priority given to the Upper instruction, regardless of whether the data is written to the same field or not.

LQI : Load with Post-Increment

Loads data from VI[is] from VU Mem to VF[ft] and adds 1 to VI[is].

Operation Code

Lower 32-bit word: LowerOP field type 3

Lower OP. (31–25)	dest (24–21)	ft reg (20–16)	is reg (15–11)	LQI (10–00)
1000000	----	-----	-----	01101 1111 00
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

LQI.dest **VF[**ft**]**dest**, (VI[**is**]++)**

For VI[is], specifies the value obtained by dividing the address by 16.

Operation

if (x ⊆ dest) then VF[ft]x = VU_MEMx(VI[is] × 16)
if (y ⊆ dest) then VF[ft]y = VU_MEMy(VI[is] × 16)
if (z ⊆ dest) then VF[ft]z = VU_MEMz(VI[is] × 16)
if (w ⊆ dest) then VF[ft]w = VU_MEMw(VI[is] × 16)
VI[is] = VI[is] + 1

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
LQI.xyzw VF10xyzw, (VI04++)
```

VF10x = VU_MEMx(VI04 × 16)
VF10y = VU_MEMy(VI04 × 16)
VF10z = VU_MEMz(VI04 × 16)
VF10w = VU_MEMw(VI04 × 16)
VI04 = VI04 + 1

Remarks

This instruction cannot be placed in the E bit delay slot.
When an Upper instruction in the same cycle writes data to the VF[ft] register, the result of this instruction is discarded with priority given to the Upper instruction, regardless of whether the data is written to the same field or not.

MFIR : Move from Integer Register to Floating-Point Register

Moves the data of VI[is] to VF[ft].

Operation Code

Lower 32-bit word: LowerOP field type 3

Lower OP. (31–25)	dest (24–21)	ft reg (20–16)	is reg (15–11)	MFIR (10–00)		
1000000	----	-----	-----	01111	1111	01
7 bits	4 bits	5 bits	5 bits	11 bits		

Mnemonic

MFIR.dest **VF[**ft**]**dest**, VI[**is**]**

Operation

if (x ⊆ dest) then VF[ft]x = VI[is]
if (y ⊆ dest) then VF[ft]y = VI[is]
if (z ⊆ dest) then VF[ft]z = VI[is]
if (w ⊆ dest) then VF[ft]w = VI[is]

The data of VI[is] is sign extended from 16 bits to 32 bits.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
MFIR.y VF10y, VI12
```

VF10y = VI12

Remarks

When an Upper instruction in the same cycle writes data to the VF[ft] register, the result of this instruction is discarded with priority given to the Upper instruction, regardless of whether the data is written to the same field or not.

MFP : Move from P Register to Floating-Point Register

Moves the value of the P register to VF[ft].

Operation Code

Lower 32-bit word: LowerOP field type 3

Lower OP. (31–25)	dest (24–21)	ft reg (20–16)	fs reg (15–11)	MFP (10–00)		
1000000	----	-----	00000	11001	1111	00
7 bits	4 bits	5 bits	5 bits	11 bits		

Mnemonic

MFP.dest **VF[**ft**]**dest**, P**

Operation

if (x ⊆ dest) then VF[ft]x = P
if (y ⊆ dest) then VF[ft]y = P
if (z ⊆ dest) then VF[ft]z = P
if (w ⊆ dest) then VF[ft]w = P

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
MFP.x VF10x, P
```

VF10x = P

Remarks

A data dependency check is not performed with the P register. To read the appropriate operation results, use the WAITP instruction for synchronization.
When an Upper instruction in the same cycle writes data to the VF[ft] register, the result of this instruction is discarded with priority given to the Upper instruction, regardless of whether the data is written to the same field or not.

MOVE : Transfer between Floating-Point Registers

Transfers the value of VF[fs] to VF[ft].

Operation Code

Lower 32-bit word: LowerOP field type 3

Lower OP. (bits 31–25)	dest (bits 24–21)	ft reg (bits 20–16)	fs reg (bits 15–11)	MOVE (bits 10–00)		
1000000	----	-----	-----	01100	1111	00
7 bits	4 bits	5 bits	5 bits	11 bits		

Mnemonic

MOVE.dest **VF[**ft**]**dest, **VF[**fs**]**dest

Operation

if (x ⊆ dest) then VF[ft]x = VF[fs]x
if (y ⊆ dest) then VF[ft]y = VF[fs]y
if (z ⊆ dest) then VF[ft]z = VF[fs]z
if (w ⊆ dest) then VF[ft]w = VF[fs]w

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
MOVE.xyzw VF10xyzw, VF20xyzw
```

VF10x = VF20x
VF10y = VF20y
VF10z = VF20z
VF10w = VF20w

Remarks

When an Upper instruction in the same cycle writes data to the VF[ft] register, the result of this instruction is discarded with priority given to the Upper instruction, regardless of whether the data is written to the same field or not.

MR32 : Move with Rotate

Moves the value of VF[fs] to VF[ft], rotating field by field.

Operation Code

Lower 32-bit word: LowerOP field type 3

Lower OP. (bits 31–25)	dest (bits 24–21)	ft reg (bits 20–16)	fs reg (bits 15–11)	MR32 (bits 10–00)		
1000000	----	-----	-----	01100	1111	01
7 bits	4 bits	5 bits	5 bits	11 bits		

Mnemonic

MR32.dest **VF[**ft**]**dest, **VF[**fs**]**dest

Operation

if (x ⊆ dest) then VF[ft]x = VF[fs]y
if (y ⊆ dest) then VF[ft]y = VF[fs]z
if (z ⊆ dest) then VF[ft]z = VF[fs]w
if (w ⊆ dest) then VF[ft]w = VF[fs]x

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
MR32.xyzw VF10xyzw, VF20xyzw
```

VF10x = VF20y
VF10y = VF20z
VF10z = VF20w
VF10w = VF20x

Remarks

When an Upper instruction in the same cycle writes data to the VF[ft] register, the result of this instruction is discarded with priority given to the Upper instruction, regardless of whether the data is written to the same field or not.

MTIR : Move from Floating-Point Register to Integer Register

Moves the lower 16 bits of the fsf field of VF[fs] to VI[it].

Operation Code

Lower 32-bit word: LowerOP field type 4

Lower OP. (bits 31–25)	ftf (24–23)	fsf (22–21)	it reg (20–16)	fs reg (15–11)	MTIR (10–00)
1000000	00	--	-----	-----	01111 1111 00
7 bits	2 bits	2 bits	5 bits	5 bits	11 bits

Mnemonic

MTIR VI[it**], VF[**fs**]**fsf

Operation

VI[it] = VF[fs]fsf

Only the lower 16 bits of VF[fs]fsf are transferred.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 1

Example

```
MTIR VI12, VF10y
```

VI12 = VF10y

RGET : Get Random Number

Gets a random number and stores it in VF[ft].

Operation Code

Lower 32-bit word: LowerOP field type 3

Lower OP. (31–25)	dest (24–21)	ft reg (20–16)	fs reg (15–11)	RGET (10–00)		
1000000	----	-----	00000	10000	1111	01
7 bits	4 bits	5 bits	5 bits	11 bits		

Mnemonic

RGET.dest **VF[**ft**]**dest**, R**

Operation

if (x ⊆ dest) then VF[ft]x = R*
if (y ⊆ dest) then VF[ft]y = R*
if (z ⊆ dest) then VF[ft]z = R*
if (w ⊆ dest) then VF[ft]w = R*

The random number obtained here (R*) consists of the 23-bit R register as mantissa and 001111111 as sign and exponent. The value range is: +1.0 < R* < +2.0

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
RGET.x VF10x, R
RNEXT.y VF10y, R
RNEXT.z VF10z, R
```

Remarks

When multiple fields are specified as destinations, the same value is stored in each field.
When an Upper instruction in the same cycle writes data to the VF[ft] register, the result of this instruction is discarded with priority given to the Upper instruction, regardless of whether the data is written to the same field or not.

RINIT : Random Number Intialize

Sets the R register to the field specified by the fsf field of VF[fs].

Operation Code

Lower 32-bit word: LowerOP field type 4

31–25	24–23	22–21	20–16	15–11	10–00		
Lower OP.	ftf	fsf	ft reg	fs reg	RINIT		
1000000	00	--	00000	-----	10000	1111	10
7 bits	2 bits	2 bits	5 bits	5 bits	11 bits		

Mnemonic

RINIT R, VF[fs**]**fsf

Operation

R = VF[fs]fsf

The lower 23 bits of VF[fs]fsf are transferred to the R register.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 1

Example

```
RINIT R, VF10x
```

RNEXT : Next Random Number

Generates a new random number and stores it in VF[ft].

Operation Code

Lower 32-bit word: LowerOP field type 3

Lower OP. (bits 31–25)	dest (bits 24–21)	ft reg (bits 20–16)	fs reg (bits 15–11)	RNEXT (bits 10–00)		
1000000	----	-----	00000	10000	1111	00
7 bits	4 bits	5 bits	5 bits	11 bits		

Mnemonic

RNEXT.dest **VF[**ft**]**dest**, R**

Operation

R = next(R)
if (x ⊆ dest) then VF[ft]x = R*
if (y ⊆ dest) then VF[ft]y = R*
if (z ⊆ dest) then VF[ft]z = R*
if (w ⊆ dest) then VF[ft]w = R*

Random numbers are generated in M series.

The random number obtained here (R*) consists of the 23-bit R register as mantissa and 001111111 as sign and exponent. The value range is: +1.0 < R* < +2.0

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
RGET.x VF10x, R
RNEXT.y VF10y, R
RNEXT.z VF10z, R
```

Remarks

When multiple fields are specified as destinations, the same value is stored in each field.

When an Upper instruction in the same cycle writes data to the VF[ft] register, the result of this instruction is discarded with priority given to the Upper instruction, regardless of whether the data is written to the same field or not.

RSQRT : Square Root Division

Divides the fsf field of VF[fs] by the square root of the ftf field of VF[ft] and stores the result in the Q register.

Operation Code

Lower 32-bit word: LowerOP field type 4

31–25	24 23	22 21	20–16	15–11	10–00
Lower OP.	ftf	fsf	ft reg	fs reg	RSQRT
1000000	--	--	-----	-----	01110 1111 10
7 bits	2 bits	2 bits	5 bits	5 bits	11 bits

Mnemonic

RSQRT Q, VF[fs**]**fsf, **VF[**ft**]**ftf

Operation

Q = VF[fs]$_{fsf}$ ÷ $\sqrt{\textbf{VF[ft]}_{ftf}}$

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	X	X	-	-	-	-	X	X	-	-	-	-	-

Throughput/Latency

13 / 13

Example

```
RSQRT Q, VF10x, VF20y
```

Q = VF10x ÷ $\sqrt{|VF20y|}$

Remarks

A data dependency check is not performed with the Q register. To execute subsequent instructions after the results of the RSQRT instruction are written to the Q register, use the WAITQ instruction for synchronization.

RXOR : Random Number Set

Sets the exclusive logical sum of the R register and the fsf field of VF[fs] to the R register.

Operation Code

Lower 32-bit word: LowerOP field type 4

Lower OP. (31–25)	ftf (24–23)	fsf (22–21)	ft reg (20–16)	fs reg (15–11)	RXOR (10–00)
1000000	00	--	00000	-----	10000 1111 11
7 bits	2 bits	2 bits	5 bits	5 bits	11 bits

Mnemonic

RXOR R, VF[fs**]**$_{fsf}$

Operation

R = xor(R, VF[fs]$_{fsf}$ **)**

Only the lower 23 bits of VF[fs]$_{fsf}$ are used.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 1

Example

```
RXOR R, VF10x
```

SQ : Store Qword with Offset

Stores the data of VF[fs] in the VU Mem address specified by VI[it] and the immediate value.

Operation Code

Lower 32-bit word: LowerOP field type 7

SQ (31–25)	dest (24–21)	it reg (20–16)	fs reg (15–11)	Imm11 (10–00)
0000001	----	-----	-----	-----------
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

SQ.dest **VF[**fs**]**dest, Imm11**(VI[**it**])**

Imm11 is a 11-bit signed integer.

For Imm11 and VI[it], specify the value obtained by dividing the address by 16.

Operation

if (x ⊆ dest) then VU_MEMx((Imm11 + VI[it]) × 16) = VF[fs]x
if (y ⊆ dest) then VU_MEMy((Imm11 + VI[it]) × 16) = VF[fs]y
if (z ⊆ dest) then VU_MEMz((Imm11 + VI[it]) × 16) = VF[fs]z
if (w ⊆ dest) then VU_MEMw((Imm11 + VI[it]) × 16) = VF[fs]w

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
SQ.xyzw VF10xyzw, offset(VI04)
```

VU_MEMx((offset + VI04) × 16) = VF10x
VU_MEMy((offset + VI04) × 16) = VF10y
VU_MEMz((offset + VI04) × 16) = VF10z
VU_MEMw((offset + VI04) × 16) = VF10w

Remarks

This instruction cannot be placed in the E bit delay slot.

SQD : Store Qword with Pre-Decrement

Subtracts 1 from VI[it] and stores the data of VF[fs] to the address of VU Mem.

Operation Code

Lower 32-bit word: LowerOP field type 3

Lower OP. (31–25)	dest (24–21)	it reg (20–16)	fs reg (15–11)	SQD (10–00)		
1000000	----	-----	-----	01101	1111	11
7 bits	4 bits	5 bits	5 bits	11 bits		

Mnemonic

SQD.dest **VF[**fs**]**dest**, (--VI[**it**])**

For VI[it], specifies the value obtained by dividing the address by 16.

Operation

VI[is] = VI[is] – 1
if (x ⊆ dest) then VU_MEMx(VI[it] × 16) = VF[fs]x
if (y ⊆ dest) then VU_MEMy(VI[it] × 16) = VF[fs]y
if (z ⊆ dest) then VU_MEMz(VI[it] × 16) = VF[fs]z
if (w ⊆ dest) then VU_MEMw(VI[it] × 16) = VF[fs]w

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
SQD.xyzw VF10xyzw, (--VI04)
```

VI04 = VI04 – 1
VU_MEMx(VI04 × 16) = VF10x
VU_MEMy(VI04 × 16) = VF10y
VU_MEMz(VI04 × 16) = VF10z
VU_MEMw(VI04 × 16) = VF10w

Remarks

This instruction cannot be placed in the E bit delay slot.

SQI : Store with Post-Increment

Stores the data of VF[fs] to the VU Mem address specified by VI[it] and adds 1 to VI[it].

Operation Code

Lower 32-bit word: LowerOP field type 3

31–25	24–21	20–16	15–11	10–00		
Lower OP.	dest	it reg	fs reg	SQI		
1000000	----	-----	-----	01101	1111	01
7 bits	4 bits	5 bits	5 bits	11 bits		

Mnemonic

SQI.dest **VF[**fs**]**dest**, (VI[**it**]++)**

For VI[it], specifies the value obtained by dividing the address by 16.

Operation

if (x ⊆ dest) then VU_MEMx(VI[it] × 16) = VF[fs]x
if (y ⊆ dest) then VU_MEMy(VI[it] × 16) = VF[fs]y
if (z ⊆ dest) then VU_MEMz(VI[it] × 16) = VF[fs]z
if (w ⊆ dest) then VU_MEMw(VI[it] × 16) = VF[fs]w
VI[is] = VI[is] + 1

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 4

Example

```
SQI.xyzw VF10xyzw, (VI04++)
```

VU_MEMx(VI04 × 16) = VF10x
VU_MEMy(VI04 × 16) = VF10y
VU_MEMz(VI04 × 16) = VF10z
VU_MEMw(VI04 × 16) = VF10w
VI04 = VI04 + 1

Remarks

This instruction cannot be placed in the E bit delay slot.

SQRT : Square Root

Obtains the square root of the ftf field of VF[ft] and stores the result in the Q register.

Operation Code

Lower 32-bit word: LowerOP field type 4

31–25	24–23	22–21	20–16	15–11	10–00		
Lower OP.	ftf	fsf	ft reg	fs reg	SQRT		
1000000	--	00	-----	00000	01110	1111	01
7 bits	2 bits	2 bits	5 bits	5 bits	11 bits		

Mnemonic

SQRT Q, VF[ft**]**ftf

Operation

$$Q = \sqrt{|VF[ft]_{ftf}|}$$

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	X	X	-	-	-	-	X	X	-	-	-	-	-

Throughput/Latency

7 / 7

Example

```
SQRT Q, VF10x
```

$Q = \sqrt{|VF10x|}$

Remarks

A data dependency check is not performed with the Q register. To execute subsequent instructions after the results of the SQRT instruction are written to the Q register, use the WAITQ instruction for synchronization.

WAITP : P Register Syncronize

Waits until the calculation result is written to the P register.

Operation Code

Lower 32-bit word: LowerOP field type 3

Lower OP. (31–25)	dest (24–21)	ft reg (20–16)	fs reg (15–11)	WAITP (10–00)
1000000	0000	00000	00000	11110 1111 11
7 bits	4 bits	5 bits	5 bits	11 bits

Mnemonic

WAITP

Operation

Interlocks until the calculation result of the previous EFU instruction (e.g. ESIN) is written to the P register.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1~54 / 1~54

Example

```
NOP ELENG P, VF10
NOP WAITP
NOP MFP.x VF20.x, P
```

Remarks

A data dependency check is not performed with the P register. Use the WAITP instruction to synchronize after an EFU instruction, if the subsequent instruction uses the result.

WAITQ : Q Register Syncronize

Waits until the calculation result is written into Q register.

Operation Code

Lower 32-bit word: LowerOP field type 3

Lower OP. (bits 31–25)	dest (bits 24–21)	ft reg (bits 20–16)	fs reg (bits 15–11)	WAITQ (bits 10–00)		
1000000	0000	00000	00000	01110	1111	11
7 bits	4 bits	5 bits	5 bits	11 bits		

Mnemonic

WAITQ

Operation

Interlocks until the calculation result of the previous FDIV instruction (DIV/SQRT/RSQRT) is written into Q register.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1~13 / 1~13

Example

```
NOP SQRT Q, VF10x
NOP WAITQ
ADDq.xyzw VF20xyzw, VF11xyzw, Q NOP
```

Remarks

A data dependency check is not performed with the Q register. Use the WAITQ instruction to synchronize after the FDIV instruction, if the subsequent instruction uses the result.

XGKICK : GIF Control

Activates GIF transfer via PATH1 using the value of VI[is].

Operation Code

Lower 32-bit word: LowerOP field type 3

31–25	24–21	20–16	15–11	10–00		
Lower OP.	dest	ft reg	is reg	XGKICK		
1000000	0000	00000	-----	11011	1111	00
7 bits	4 bits	5 bits	5 bits	11 bits		

Mnemonic

XGKICK VI[is**]**

Operation

Provides the value of VI[is] for the external unit (GIF) and gives an instruction to start transfer via PATH1. If the preceding transfer via PATH1 is still in process, stalls until it ends.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1~ / 1~

Example

```
XGKICK VI10
```

Remarks

XGKICK instruction may be used only on VU1.

It cannot be placed in the E bit delay slot. If the E bit is set by an Upper instruction in the same cycle, the VIFcode FLUSH may not wait for the end of the XGKICK instruction properly. For details, see "FLUSH" in the "EE User's Manual".

For operations with stalls, refer to "3.4.9. XGKICK Pipeline".

GIF considers the provided value of VI[is] as a GS packet (leading GIFtag) address, and transfers data.

Since the XGKICK instruction itself ends when the GIF starts transferring, caution should be taken against rewriting data in the VU Mem during data transfer.

In addition, give attention to the latency of a store instruction so that the transfer starts after the data stored immediately before the XGKICK instruction is written in the VU Mem.

XITOP : VIF Control

Reads the value of the ITOP register of VIF and stores it in VI[it].

Operation Code

Lower 32-bit word: LowerOP field type 3

31–25	24–21	20–16	15–11	10–00		
Lower OP.	dest	it reg	is reg	XITOP		
1000000	0000	-----	00000	11010	1111	01
7 bits	4 bits	5 bits	5 bits	11 bits		

Mnemonic

XITOP VI[it**]**

Operation

VI[it] = ITOP

Reads the value of the ITOP register of the external unit (VIF). The value is read at the T stage.
In VU0, whose data memory capacity is 4 Kbytes, the lower 8 bits of the VIF0_ITOP register are zero-extended and read. In VU1, the lower 10 bits of the VIF1_ITOP register are zero-extended and read.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 1

Example

```
XITOP VI10
```

Remarks

This instruction cannot be placed in the E bit delay slot.

XTOP : VIF Control

Reads the value of the TOP register of VIF and stores it in VI[it].

Operation Code

Lower 32-bit word: LowerOP field type 3

Lower OP. (31–25)	dest (24–21)	it reg (20–16)	is reg (15–11)	XTOP (10–00)		
1000000	0000	-----	00000	11010	1111	00
7 bits	4 bits	5 bits	5 bits	11 bits		

Mnemonic

XTOP VI[it]

Operation

VI[it] = TOP

Reads the value of the TOP register from the external unit (VIF). The value is read at the T stage.

Flag Changes

MAC flag				status flag												clipping flag
Oxyzw	Uxyzw	Sxyzw	Zxyzw	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z	
----	----	----	----	-	-	-	-	-	-	-	-	-	-	-	-	-

Throughput/Latency

1 / 1

Example

```
XTOP VI10
```

Remarks

XTOP can be used only on VU1.

This instruction cannot be placed in the E bit delay slot.

5. Macro Mode

Macro mode is an operation mode that exists only for VU0. From the point of view of the EE Core, the VU operates as a coprocessor (COP2) in this mode. Operation of the VU can be specified by coprocessor instructions.

The VU can be used as an FPU by using coprocessor calculation instructions in macro mode. Basic floating-point instructions such as add, subtract, multiply, and divide are defined.

In addition, it is possible to transfer data to VU registers by using coprocessor transfer instructions, and to make the VU execute micro subroutines by using micro subroutine execution instructions. Since the VU then enters micro mode and operates independently, high performance can be realized.

This chapter describes the method of controlling VU1 from the EE Core as well as macro mode in VU0.

5.1. Macro Mode Register Set

5.1.1. Floating-Point Registers

The VU0 floating-point registers are allocated to the COP2 data register. The register fields etc. are the same as those in the micro mode. For details, see "**3.1.1. Floating-Point Registers**".

	32 bits 127 96	32 bits 95 64	32 bits 63 32	32 bits 31 0
CPR[2,00]	VF00w	VF00z	VF00y	VF00x
CPR[2,01]	VF01w	VF01z	VF01y	VF01x
CPR[2,02]	VF02w	VF02z	VF02y	VF02x
CPR[2,03]	VF03w	VF03z	VF03y	VF03x
		:		
		:		
CPR[2,31]	VF31w	VF31z	VF31y	VF31x

5.1.2. Integer Registers

The VU0 integer registers are allocated to the lower 16 bits of the first 16 COP2 control registers. For details of the integer registers, see "**3.1.2. Integer Registers**".

	31	15 0
CCR[2,00]	0000000000000000	VI00 (0 register)
CCR[2,01]	0000000000000000	VI01
CCR[2,02]	0000000000000000	VI02
CCR[2,03]	0000000000000000	VI03
CCR[2,04]	0000000000000000	VI04
CCR[2,05]	0000000000000000	VI05
CCR[2,06]	0000000000000000	VI06
CCR[2,07]	0000000000000000	VI07
CCR[2,08]	0000000000000000	VI08
CCR[2,09]	0000000000000000	VI09
CCR[2,10]	0000000000000000	VI10
CCR[2,11]	0000000000000000	VI11
CCR[2,12]	0000000000000000	VI12
CCR[2,13]	0000000000000000	VI13
CCR[2,14]	0000000000000000	VI14 (Stack pointer: recommended)
CCR[2,15]	0000000000000000	VI15 (Link register: recommended)

5.1.3. Control Registers

Control registers such as flag registers are allocated to the last 16 COP2 control registers.
These control registers can be accessed only by the CTC2/CFC2 instruction, and cannot be accessed by other macroinstructions and microinstructions.

	31 23 15 7 0
CCR[2,16]	Status flag
CCR[2,17]	MAC flag
CCR[2,18]	clipping flag
CCR[2,19]	reserved
CCR[2,20]	R register
CCR[2,21]	I register
CCR[2,22]	Q register
CCR[2,23]	reserved
CCR[2,24]	reserved
CCR[2,25]	reserved
CCR[2,26]	TPC register
CCR[2,27]	CMSAR0 register
CCR[2,28]	FBRST register
CCR[2,29]	VPU-STAT register
CCR[2,30]	reserved
CCR[2,31]	CMSAR1 register

The initial values and access limits of the control registers are shown in the table below. r* and w* indicate the necessity of inserting a VNOP instruction considering the access timing.

Control register	Initial value	VU0 is stopped	VU0 is operating (Running state)
status flag	all 0	r/w	r*/w*
MAC flag	all 0	r/-	r*/-
clipping flag	all 0	r/w	r*/w*
R register	indeterminate	r/w	-/-
I register	indeterminate	r/w	-/-
Q register	indeterminate	r/w	-/-
TPC register	indeterminate	r/-	-/-
CMSAR0 register	indeterminate	r/w	r/w
FBRST register	all 0	r/w	r/w
VPU-STAT register	all 0	r/-	r/-
CMSAR1 register	indeterminate	-/w	-/w

Writes to the CMSAR1 register are ignored while VU1 is operating.

Status flag

31-24	23-16	15-12	11	10	9	8	7	6	5	4	3	2	1	0
00000000	00000000	0000	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z

The 12-bit status flag, which reflects VU0 calculation results, is allocated to the lower 12 bits of the control register, CCR[2,16]. For details of the flag, see "**3.3.2. Status Flags (SF)**".
When accessing the status flag during VU0 operation, it is necessary to insert a VNOP instruction etc. considering the timing. When performing a write operation, values written to the lower 6 bits (D, I, O, U, S, Z) are ignored.

MAC flag

31–24	23–16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00000000	00000000	Ox	Oy	Oz	Ow	Ux	Uy	Uz	Uw	Sx	Sy	Sz	Sw	Zx	Zy	Zz	Zw

The 16-bit MAC flag, which reflects VU0 calculation results, is allocated to the lower 16 bits of the control register, CCR[2,17]. For details of the flag, see "**3.3.1. MAC Flags**".
When accessing the MAC flag during VU0 operation, an instruction such as VNOP should be inserted in consideration of access timing.

Clipping flag

31–24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00000000	-z	+z	-y	+y	-x	+x	-z	+z	-y	+y	-x	+x	-z	+z	-y	+y	-x	+x	-z	+z	-y	+y	-x	+x

The 24-bit clipping flag, which reflects VU0 clipping judgment results, is allocated to the lower 24 bits of the control register, CCR[2,18]. For details of the flag, see "**3.3.3. Clipping Flags (CF)**".
When accessing the clipping flag during VU0 operation, an instruction such as VNOP should be inserted in consideration of access timing.

R, I, Q register

31–24	23	22–0
00000000	0	R
I		
Q		

These registers are used for accessing the R, I, Q registers in VU0 during debugging.

TPC register

31–24	23–16	15–0
00000000	00000000	TPC

This register indicates the PC (Program Counter) at which the micro subroutine has stopped. The cause of the stop can be checked with the VPU-STAT register.

CMSAR0 register

31–24	23–16	15–0
00000000	00000000	CMSAR0

This register specifies the starting address when attempting to make VU0 execute a micro subroutine by means of the VCALLMSR instruction.

FBRST register

31 - 24	23 - 16	15 - 12	11	10	9	8	7 - 4	3	2	1	0
00000000	00000000	0000	TE1	DE1	RS1	FB1	0000	TE0	DE0	RS0	FB0

This register controls the operation status of VU0 and VU1. The function of each bit is shown in the table below.

Bit	Function	Write	Read
FB0	VU0 Force Break	0: - 1: Force Break	Always 0
RS0	VU0 Reset	0: - 1: Reset	Always 0
DE0	VU0 D bit enable	0: Disable 1: Enable	Current setting
TE0	VU0 T bit enable	0: Disable 1: Enable	Current setting
FB1	VU1 Force Break	0: - 1: Force Break	Always 0
RS1	VU1 Reset	0: - 1: Reset	Always 0
DE1	VU1 D bit enable	0: Disable 1: Enable	Current setting
TE1	VU1 T bit enable	0: Disable 1: Enable	Current setting

For details of the status of VU operations such as Force Break and Reset, see **"1.3. VU Operation Status"**.

VPU-STAT register

31 - 24	23 - 16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00000000	00000000	0	EFU1	DIV1	VGW1	VFS1	VTS1	VDS1	VBS1	IBS0	0	DIV0	0	VFS0	VTS0	VDS0	VBS0

This is a read-only register, which reflects the operation status of VU0 and VU1. The bits are described in the table below.

Bit	Definition
VBS0	VU0 operation status 0: idle (Stopped by E bit or Reset) 1: busy (Executing micro subroutine)
VDS0	VU0 operation status 0: Operating or stopped by a factor other than D bit 1: Stopped by D bit
VTS0	VU0 operation status 0: Operating or stopped by a factor other than T bit 1: Stopped by T bit
VFS0	VU0 operation status 0: Operating or stopped by a factor other than Force Break 1: Stopped by Force Break
DIV0	VU0 DIV unit operation status 0: idle 1: busy
IBS0	VIF0 operation status 0: idle 1: busy
VBS1	VU1 operation status 0: idle (Stopped by E bit or Reset) 1: busy (Executing micro subroutine)
VDS1	VU1 operation status 0: Not D bit stop 1: Stopped by D bit
VTS1	VU1 operation status 0: Not T bit stop 1: Stopped by T bit
VFS1	VU1 operation status 0: Not Force Break stop 1: Stopped by Force Break
VGW1	VU1 XGKICK related status 0: Not-waiting 1: Waiting for sync with external unit
DIV1	VU1 DIV unit operation status 0: idle 1: busy
EFU1	VU1 EFU unit operation status 0: idle 1: busy

CMSAR1 register

31 … 24	23 … 16	15 … 8 7 … 0
00000000	00000000	CMSAR1

This is the control register to activate a VU1 micro subroutine. By writing a 16-bit address (an address in byte units in Micro Mem) to this register when VU1 is in the stopped state, VU1 starts micro subroutine execution from this address. (In contrast to CMSAR0, a VU1 micro subroutine can be activated just with a write operation.)

5.1.4. Special Registers

ACC, I, Q, and R registers in the macro mode function in the same way as those in the micro mode. See "**3.1. Micro Mode Register Set**".

Since the macro instruction set does not have the I bit and the Lower instruction field, a coprocessor transfer instruction is used to set the I register to a floating-point immediate value.

5.2. Macro Instruction Set Overview

5.2.1. MIPS COP2 Instructions

Macro instructions conform to the MIPS COP2 Instruction format. They are broadly divided into four classifications:

- Coprocessor transfer instructions
- Coprocessor branch instructions
- Macro (primitive) calculation instructions
- Micro subroutine execution instructions

The VU macroinstructions start with the letter "V" to differentiate them from other MIPS instructions.

5.2.2. Coprocessor Transfer Instructions

Coprocessor transfer instructions are used to transfer data between VU0 registers and the EE Core. When VU0 is executing a micro subroutine, the user can specify whether or not the EE Core operation interlocks. For details, see the description of each instruction.

Category	Instruction	Function
Coprocessor transfer instruction	QMFC2	Floating-point data transfer from VU to EE Core
	QMTC2	Floating-point data transfer from EE Core to VU
	LQC2	Floating-point data transfer from EE Core to VU
	SQC2	Floating-point data transfer from VU to EE Core
	CFC2	Integer data transfer from VU to EE Core
	CTC2	Integer data transfer from EE Core to VU

5.2.3. Coprocessor Branch Instructions

Coprocessor branch instructions branch by means of the MIPS COP2 condition signal. They are used to judge VU1 status. For details, see the description of each instruction.

Category	Instruction	Function
Coprocessor branch instruction	BC2F	Branch on COP2 condition signal
	BC2FL	Branch on COP2 condition signal
	BC2T	Branch on COP2 condition signal
	BC2TL	Branch on COP2 condition signal

5.2.4. Coprocessor Calculation Instructions

Coprocessor calculation instructions are the instructions to make VU0 execute floating-point calculations such as add, subtract, multiply, and divide. Since they are a subset of the micro mode instructions, refer to Chapter 3 if necessary.

Category	Instruction	Function
Floating-point calculation instruction	VABS	absolute
	VADD	addition
	VADDi	ADD broadcast I register
	VADDq	ADD broadcast Q register
	VADDbc	ADD broadcast bc field
	VADDA	ADD output to ACC
	VADDAi	ADD output to ACC broadcast I register
	VADDAq	ADD output to ACC broadcast Q register
	VADDAbc	ADD output to ACC broadcast bc field
	VSUB	subtraction
	VSUBi	SUB broadcast I register
	VSUBq	SUB broadcast Q register
	VSUBbc	SUB broadcast bc field
	VSUBA	SUB output to ACC
	VSUBAi	SUB output to ACC broadcast I register
	VSUBAq	SUB output to ACC broadcast Q register
	VSUBAbc	SUB output to ACC broadcast bc field
	VMU	multiply
	VMULi	MUL broadcast I register
	VMULq	MUL broadcast Q register
	VMULbc	MUL broadcast bc field
	VMULA	MUL output to ACC
	VMULAi	MUL output to ACC broadcast I register
	VMULAq	MUL output to ACC broadcast Q register
	VMULAbc	MUL output to ACC broadcast bc field
	VMADD	MUL and ADD (SUB)
	VMADDi	MUL and ADD (SUB) broadcast I register
	VMADDq	MUL and ADD (SUB) broadcast Q register
	VMADDbc	MUL and ADD (SUB) broadcast bc field
	VMADDA	MUL and ADD (SUB) output to ACC
	VMADDAi	MUL and ADD (SUB) output to ACC broadcast I register
	VMADDAq	MUL and ADD (SUB) output to ACC broadcast Q register
	VMADDAbc	MUL and ADD (SUB) output to ACC broadcast bc field
	VMSUB	Multiply and SUB
	VMSUBi	Multiply and SUB broadcast I register
	VMSUBq	Multiply and SUB broadcast Q register
	VMSUBbc	Multiply and SUB broadcast bc field
	VMSUBA	Multiply and SUB output to ACC
	VMSUBAi	Multiply and SUB output to ACC broadcast I register
	VMSUBAq	Multiply and SUB output to ACC broadcast Q register
	VMSUBAbc	Multiply and SUB output to ACC broadcast bc field
	VMAX	maximum
	VMAXi	maximum broadcast I register
	VMAXbc	maximum broadcast bc field
	VMINI	minimum
	VMINIi	minimum broadcast I register
	VMINIbc	minimum broadcast bc field
	VOPMULA	outer product MULA
	VOPMSUB	outer product MSUB
	VNOP	no operation

Category	Instruction	Function
Floating-point/ fixed point conversion instructions	VFTOI0	float to integer, fixed point 0 bit
	VFTOI4	float to integer, fixed point 4 bits
	VFTOI12	float to integer, fixed point 12 bits
	VFTOI15	float to integer, fixed point 15 bits
	VITOF0	integer to float, fixed point 0 bit
	VITOF4	integer to float, fixed point 4 bits
	VITOF12	integer to float, fixed point 12 bits
	VITOF15	integer to float, fixed point 15 bits
Clipping judgment instruction	VCLIP	clipping
Floating-point divider instructions	VDIV	floating divide
	VSQRT	floating square-root
	VRSQRT	floating reciprocal square-root
Integer calculation instructions	VIADD	integer ADD
	VIADDI	integer ADD immediate
	VIAND	integer AND
	VIOR	integer OR
	VISUB	integer SUB
Register-register transfer instructions	VMOVE	move floating register
	VMFIR	move from integer register
	VMTIR	move to integer register
	VMR32	rotate right 32 bits
Load/Store instructions	VLQD	Load Quadword with pre-decrement
	VLQI	Load Quadword with post-increment
	VSQD	Store Quadword with pre-decrement
	VSQI	Store Quadword with post-increment
	VILWR	integer load word register
	VISWR	integer store word register
Random numbers	VRINIT	random-unit init R register
	VRGET	random-unit get R register
	VRNEXT	random-unit next M sequence
	VRXOR	random-unit XOR R register
Synchronization instruction	VWAITQ	wait Q register

5.2.5. Micro Subroutine Execution Instructions

Micro subroutine execution instructions execute microinstruction programs in the Micro Mem. For details, see the description of each instruction.

Category	Instruction	Function
Micro subroutine execution instruction	VCALLMS	Call micro subroutine
	VCALLMSR	Call micro subroutine register

5.3. Flags

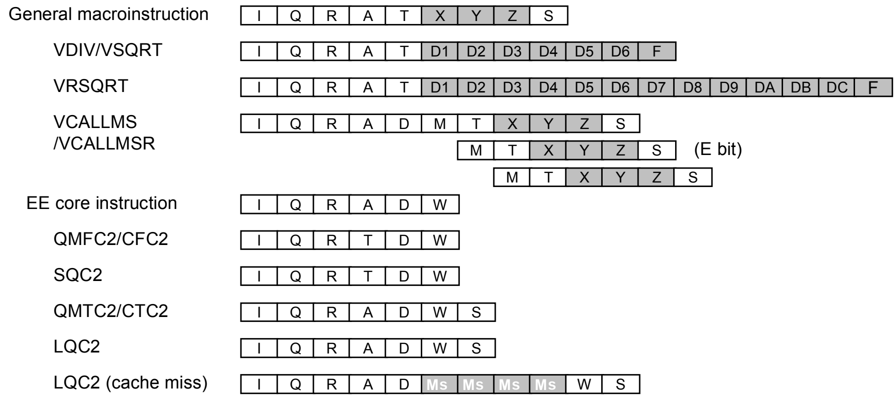
Functions and operations of MAC flags, status flags, and clipping flags in the macro mode are the same as those in the micro mode. See **"3.3.1. MAC Flags"**, **"3.3.2. Status Flags (SF)"**, and **"3.3.3. Clipping Flags (CF)"** respectively.

Since there is no flag set instruction in the macro mode, the CFC2 instruction is used to set a value to the flag.

5.4. Macro Mode Pipeline

5.4.1. Pipeline Structure of Macroinstructions

Figure 5-1 illustrates the pipeline structure of macroinstructions.



I, Q	: instruction fetch(EE Core)
R	: instruction decode, read GPR(EE Core)
A	: execute(EE Core)
W	: write(EE Core)
T	: read VGPR
D	: data transaction between EE Core and VPU
M	: read micro-Mem
X -Z	: execution stage
S	: store VGPR
D1-	: FDIV execution stage

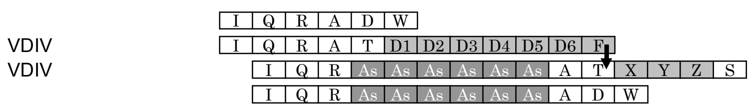
Figure 5-1 Macroinstruction Pipeline List

5.4.2. Hazards in Macro Mode

The following situations can generate stalls in macro mode:

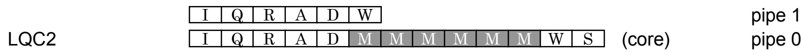
DIV resource hazard

When a macroinstruction that uses the floating-point divider unit (VDIV/VSQRT/VRSQRT) is being executed, and another macroinstruction of this type is executed concurrently.



D cache miss

When a D cache miss occurs, the LQC2/SQC2 instruction stalls at the M stage. Since this is a blocking load, the next instruction stalls at the same time.

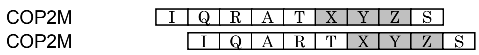


Data hazard

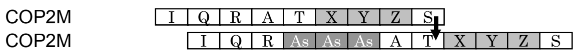
An instruction that uses the value in a register is executed before instruction results are written to the register (RAW: Read After Write Hazard), or the next instruction writes the results to the same register before instruction results are written to the register (WAW: Write After Write Hazard).
In macro mode, register data dependency checks are performed in units of register numbers. Unlike micro mode, no distinction is made between the respective fields of the floating-point register. Moreover, the floating-point register and integer register of the same number are regarded as the same register, and may cause hazards occasionally.
VF00 and VI00 are not subject to hazard checks.
Data hazards do not occur to the special registers (ACC/I/Q/R). For the Q register, however, synchronization is enabled by means of the VWAITQ instruction.

5.4.3. Macroinstruction Operation

The following figure illustrates the normal macroinstruction pipeline operation.

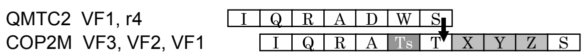


When there is a data hazard between macroinstructions, stalls occur as below.

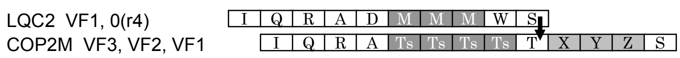


5.4.4. Operation when Transferring Data with EE Core

After a QMTC2/CTC2/LQC2 instruction which transfers data from the EE Core to the VU, the next macroinstruction always stalls for one cycle. This allows the next macroinstruction to use the transferred data.

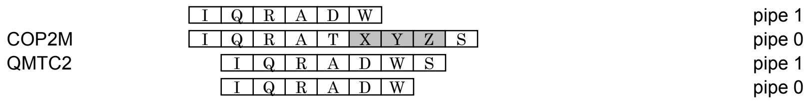


When a D cache miss is generated with the LQC2 instruction, stalls occur as illustrated below.

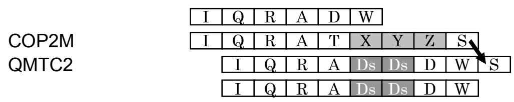


If the output register is the same when transferring data to the VU in the QMTC2/CTC2/LQC2 instruction following a macroinstruction, a WAW hazard occurs and results in stalls as illustrated below.

No WAW hazard

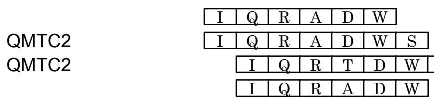


WAW hazard

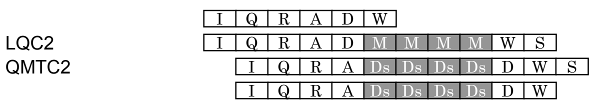


When transferring data successively with the QMTC2/CTC2/LQC2 instruction, if there is not a D cache miss, stalls do not occur.

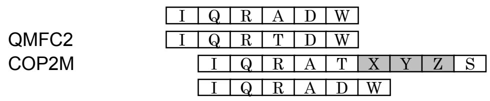
No D cache miss



D cache miss

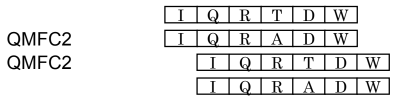


When data is transferred from the VU to the EE Core with the QMFC2/CFC2/SQC2 instruction, stalls do not occur normally as illustrated below.

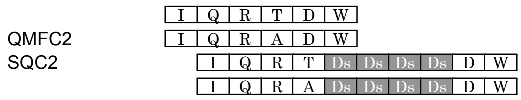


Even when executing QMFC2/CFC2/SQC2 instructions successively, no stalls occur as long as the SQC2 instruction does not overflow the store buffer.

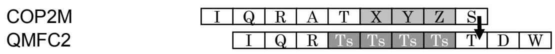
Normal conditions



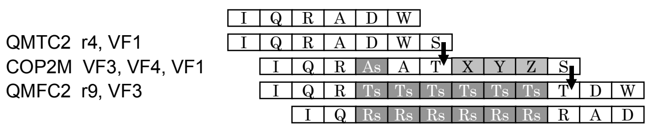
Store buffer full



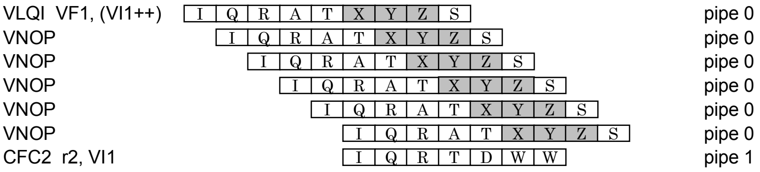
When transferring macroinstruction results to the EE Core with the QMFC2/CFC2/SQC2 instruction, stalls occur as illustrated below.



The flow of transferring data from the EE Core, processing it with a macroinstruction, and returning it to the EE Core is summarized in the figure below:

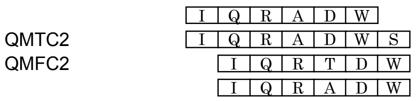


To read a VI register that is incremented/decremented by a VLQI/VLQD/VSQI/VSQD instruction, using the CFC2 instruction, the program should be written to enable the following pipeline operation:

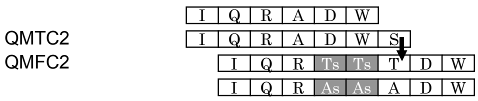


Furthermore, if there is a RAW hazard when reading data with QMFC2/CFC2/SQC2 immediately after the data is transferred from the EE Core with QMTC2/CTC2/LQC2, stalls occur for two cycles.

No RAW hazard

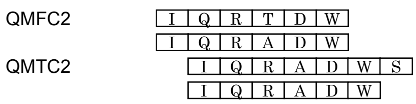


RAW hazard

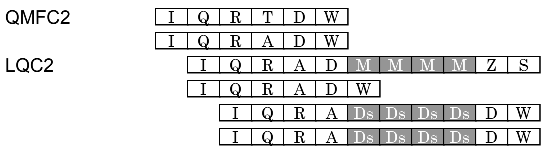


Conversely, QMTC2/CTC2/LQC2 after QMFC2/CFC2/SQC2 does not stall as long as a D cache miss is not generated or the store buffer is not full.

Normal conditions

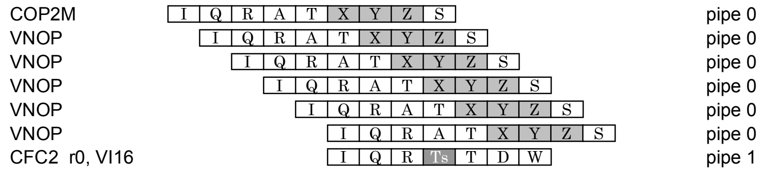


D cache miss

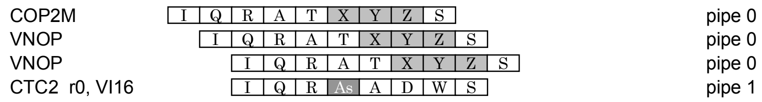


Status flag read/write

The following figure shows an example in which the VU0 status flag (CCR[2,16] register) is read with the CFC2 instruction.

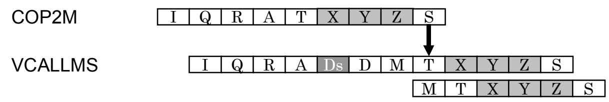


The following figure shows an example in which the status flag is written by means of the CTC2 instruction.

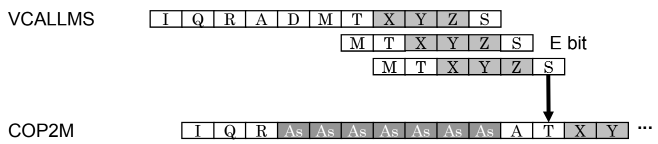


5.4.5. Operation when Executing a Micro Subroutine

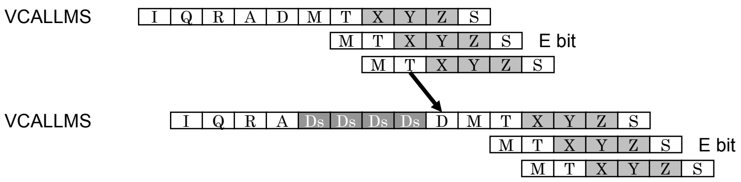
A VCALLMS instruction executed following a macroinstruction always stalls for one cycle. This enables the micro subroutine to use the results of the preceding macroinstruction.

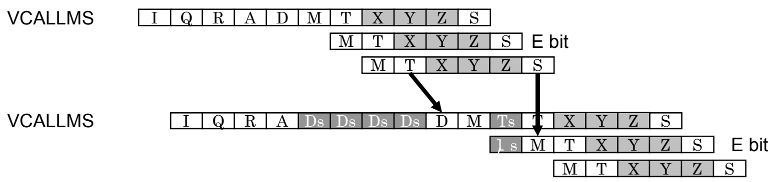


A macroinstruction executed following VCALLMS stalls at the T stage overlapping with the S stage of the last microinstruction. This enables the subsequent macroinstruction to use the results of the micro subroutine.



When two VCALLMS instructions are executed successively, the second VCALLMS stalls while the first VCALLMS is executing the microinstruction. If there is a data hazard between the called microinstructions, stalls are generated in the microinstruction.

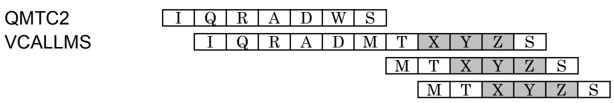




5.4.6. Micro Subroutine and Data Transfer Operations

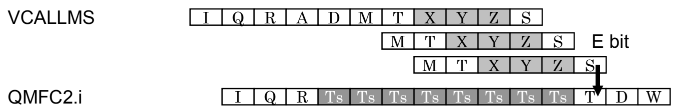
The VCALLMS instruction does not stall after transferring data from the EE Core with QMTC2/CTC2/LQC2, if there is no D cache miss. However, the transferred data can be used for the micro subroutine as illustrated below.

Normal conditions

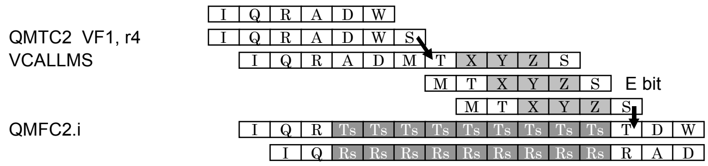


D cache miss with LQC2

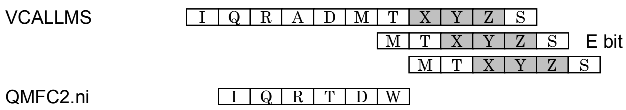
When transferring the micro subroutine results to the EE Core, a QMFC2.i/CFC2.i instruction with interlocks is used. QMFC2.i/CFC2.i stalls in the same way as the macroinstruction that follows VCALLMS, and can read the calculation results of the final instruction of the micro subroutine.



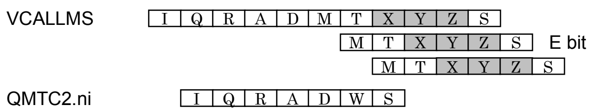
Examples of transferring data from the EE Core, executing a micro subroutine, and reading the results are shown below.



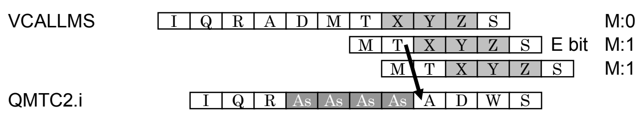
The interlock-free QMFC2.ni/CFC2.ni/SQC2 instruction is executed independently of the preceding VCALLMS instruction and does not stall. This method can be used to read the data, which is irrelevant to the micro subroutine.



When transferring data to the VU with QMTC2/CTC2/LQC2 during execution of a micro subroutine, operation is determined by the specification of interlock and the M bit of the microinstruction.
When transferring data with LQC2 and QMTC2.ni/CTC2.ni without interlocks, no stalls are generated, as shown below. This method can be used to write registers irrelevant to the micro subroutine.



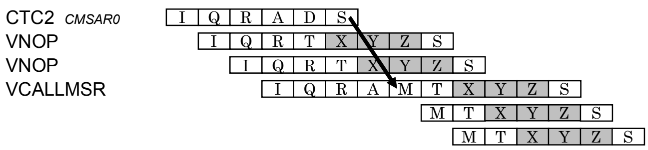
When transferring data with QMTC2.i/CTC2.i with interlocks, stalls occur and continue until a microinstruction, which sets the M bit to 1, is executed in the micro subroutine. This means that prohibition and cancellation of data transfer to the VU can be controlled on the micro subroutine side. In the following example, stalls are cancelled by setting the M bit in the second microinstruction to 1.



Using this method, the next input data can be set during execution of the micro subroutine and performance can be improved.
Once stalls are cancelled, set the M bit in all the instructions to 1 until the completion of the micro subroutine.
Operations that repeat prohibition and cancellation of data transfer are undefined.

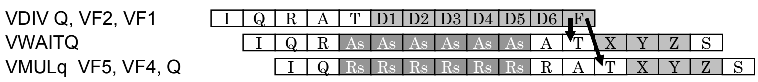
VCALLMSR instruction execution timing

The VCALLMSR instruction is executed after writing the micro subroutine address in the CMSAR0 register with the CTC2 instruction. Execute this with the following timing:



5.4.7. Q Register Synchronization

The VWAITQ instruction stalls until the VDIV/VSQRT/VRSQRT instruction executed most recently completes. Insertion of this VWAITQ enables the next macroinstruction to use the results of VDIV/VSQRT/VRSQRT.

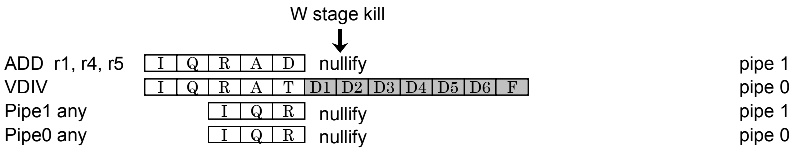


There are many factors that stop the instruction stream, such as cache miss and interrupt in micro mode. So if the Q register is referred to following an instruction such as VDIV, calculation results are not guaranteed to be correct. Therefore, perform synchronization with VWAITQ before accessing the Q register.

5.4.8. Notes on Other Pipeline Operations

W stage kill

W stage kill occurs when EE Core exceptions are generated. All the instructions operating concurrently are stopped by the W stage kill. However, COP2 divide instructions (VDIV, VSQRT, and VRSQRT) continue operation. For this reason, if a COP2 divide instruction follows an instruction to read the Q register within two cycles, it may overwrite the Q register value to be read at the occurrence of an exception. To avoid this, place two VNOP or five NOP instructions between the instruction to read the Q register and a COP2 divide instruction.



D stage stall

D stage stall occurs as a cache resource hazard when D cache misses are overlapped. The pipeline operation is shown below. The VDIV instruction is executed regardless of the D stage stall.

5.5. VU1 Control

Since VU1 is not connected to the EE Core via a coprocessor connection, it does not have macro mode. VU1 is controlled by the registers, which are mapped to VU0 data memory (VU Mem0), and some other control registers.

5.5.1. MIPS COP2 Condition Signal

The COP2 condition signal becomes TRUE when a VU1 micro subroutine is operating, and FALSE when not operating. By performing polling with instructions such as BC2T, VU1 operation status can be checked.

5.5.2. MIPS COP2 Control Register

As described in "5.1.3. Control Registers", the VU control registers are allocated to the last 16 COP2 control registers. The registers related to VU1 are shown below.

FBRST register

31–24	23–16	15–12	11	10	9	8	7–4	3	2	1	0
00000000	00000000	0000	TE1	DE1	RS1	FB1	0000	TE0	DE0	RS0	FB0

This register controls the operation status of VU0 and VU1. The function of each bit related to VU1 is shown in the table below.

Bit	Function	Write	Read
FB1	VU1 Force Break	0: - 1: Force Break	Always 0
RS1	VU1 Reset	0: - 1: Reset	Always 0
DE1	VU1 D bit enable	0: Disable 1: Enable	Current setting
TE1	VU1 T bit enable	0: Disable 1: Enable	Current setting

For details of the status of VU operations such as Force Break and Reset, see "**1.3. VU Operation Status**".

VPU-STAT register

31–24	23–16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00000000	00000000	0	EFU1	DIV1	VGW1	VFS1	VTS1	VDS1	VBS1	IBS0	0	DIV0	0	VFS0	VTS0	VDS0	VBS0

This is a read only register, which reflects the operation status of VU0 and VU1. The bits related to VU1 are described in the table below.

Bit	Definition
VBS1	VU1 operation status 0: idle (Stopped by E bit or Reset) 1: busy (Executing micro subroutine)
VDS1	VU1 operation status 0: Operating or stopped by a factor other than D bit 1: Stopped by D bit
VTS1	VU1 operation status 0: Operating or stopped by a factor other than T bit 1: Stopped by T bit
VFS1	VU1 operation status 0: Operating or stopped by a factor other than Force Break 1: Stopped by Force Break
VGW1	VU1 XGKICK-related status 0: Not-waiting 1: Waiting for sync with external unit
DIV1	VU1 DIV unit operation status 0: idle 1: busy
EFU1	VU1 EFU unit operation status 0: idle 1: busy

CMSAR1 register

31–24	23–16	15–0
00000000	00000000	CMSAR1

This is the control register to activate a VU1 micro subroutine. By writing a 16-bit address (an address in byte units in Micro Mem) to this register when VU1 is in the stopped state, VU1 starts micro subroutine execution from this adress.

5.5.3. Floating-Point Registers

VU1 floating-point registers are mapped to VU Mem0 as shown below. These registers are readable/writable only when VU1 is stopped. Access while VU1 is operating causes an indeterminate result.

VU Mem0 Address	32 bits (127–96)	32 bits (95–64)	32 bits (63–32)	32 bits (31–0)
0x4000	VF00w	VF00z	VF00y	VF00x
0x4010	VF01w	VF01z	VF01y	VF01x
0x4020	VF02w	VF02z	VF02y	VF02x
0x4030	VF03w	VF03z	VF03y	VF03x
		:		
		:		
0x41F0	VF31w	VF31z	VF31y	VF31x

5.5.4. Integer Registers

VU1 integer registers are mapped to the VU Mem0 as below. These registers are readable/writable only when VU1 is stopped. Access while VU1 is operating causes an indeterminate result.

VU Mem0 Address	127	15 0
0x4200	all 0	VI00 (0 register)
0x4210	all 0	VI01
0x4220	all 0	VI02
0x4230	all 0	VI03
0x4240	all 0	VI04
0x4250	all 0	VI05
0x4260	all 0	VI06
0x4270	all 0	VI07
0x4280	all 0	VI08
0x4290	all 0	VI09
0x42A0	all 0	VI10
0x42B0	all 0	VI11
0x42C0	all 0	VI12
0x42D0	all 0	VI13
0x42E0	all 0	VI14 (Stack pointer recommended)
0x42F0	all 0	VI15 (Link register recommended)

5.5.5. Control Registers

VU1 flag registers and special registers are mapped to the VU Mem0 as below. These registers are readable/writable only when VU1 and the GIF are both stopped. Access while either of them is operating causes an indeterminate result.

VU Mem0 address	127	31 0
0x4300	all 0	Status flag
0x4310	all 0	MAC flag
0x4320	all 0	Clipping flag
0x4330	all 0	Reserved
0x4340	all 0	R register
0x4350	all 0	I register
0x4360	all 0	Q register
0x4370	all 0	P register
0x4380	all 0	Reserved
0x4390	all 0	Reserved
0x43A0	all 0	TPC
0x43B0	all 0	Reserved
0x43C0	all 0	Reserved
0x43D0	all 0	Reserved
0x43E0	all 0	Reserved
0x43F0	all 0	Reserved

The initial value and access limit of each register are shown in the table below.

Control register	Initial Value	VU1/GIF is stopped	VU1/GIF is operating
Status flag	all 0	r/w	-/-
MAC flag	all 0	r/-	-/-
Clipping flag	all 0	r/w	-/-
R register	Indeterminate	r/w	-/-
I register	Indeterminate	r/w	-/-
Q register	Indeterminate	r/w	-/-
P register	Indeterminate	r/w	-/-
TPC register	Indeterminate	r/-	-/-

Status flag: VU Mem0 0x4300

127–16	15–12	11	10	9	8	7	6	5	4	3	2	1	0
all 0	0000	DS	IS	OS	US	SS	ZS	D	I	O	U	S	Z

The status flag (12-bit), which reflects VU1 calculation results, is allocated to the lower 12 bits. When performing a write operation, values written to the lower 6 bits are ignored.

MAC flag: VU Mem0 0x4310

127–16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
all 0	Ox	Oy	Oz	Ow	Ux	Uy	Uz	Uw	Sx	Sy	Sz	Sw	Zx	Zy	Zz	Zw

The MAC flag (16-bit), which reflects VU1 calculation results, is allocated to the lower 16 bits.

Clipping flag: VU Mem0 0x4320

127–24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
all 0	-z	+z	-y	+y	-x	+x	-z	+z	-y	+y	-x	+x	-z	+z	-y	+y	-x	+x	-z	+z	-y	+y	-x	+x

The clipping flag (24-bit), which reflects VU1 clipping judgment results, is allocated to the lower 24 bits.

R, I, Q, P register: VU Mem0 0x4340, 0x4350, 0x4360, 0x4370

127–24	23	22–0
all 0	0	R register

127–32	31–0
all 0	I register
all 0	Q register
all 0	P register

These registers are used for accessing the R, I, Q, P registers during debugging.

TPC register: VU Mem0 0x43A0

127–16	15–0
all 0	TPC

This register indicates the PC (Program Counter) in which the micro subroutine has stopped.

(This page is left blank intentionally)

6. Macro Mode Instruction Reference

6.1. Macro Instruction Operation Code

6.1.1. Macro Instruction Operation Type

There are seven types of macro instructions:

MacroOP field type 0

32-bit word: MacroOP field type 0

31 30 29 28 27 26	25	24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 09 08 07 06	05 04 03 02	01 00
COP2	co	dest	ft reg	fs reg	fd reg	OPCODE	bc
010010	1	----	-----	-----	-----	----	--
6 bits	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Specifies three registers and the broadcast field.

MacroOP field type 1

32-bit word: MacroOP field type 1

31 30 29 28 27 26	25	24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 09 08 07 06	05 04 03 02 01 00
COP2	co	dest	ft reg	fs reg	fd reg	OPCODE
010010	1	----	-----	-----	-----	------
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Specifies three registers.

MacroOP field type 2

32-bit word: MacroOP field type 2

31 30 29 28 27 26	25	24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 09 08 07 06 05 04 03 02	01 00
COP2	co	dest	ft reg	fs reg	OPCODE?	bc
010010	1	----	-----	-----	----- 1111	--
6 bits	1	4 bits	5 bits	5 bits	9 bits	2 bits

Specifies two registers and the broadcast field.

MacroOP field type 3

32-bit word: MacroOP field type 3

31 30 29 28 27 26	25	24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 09 08 07 06 05 04 03 02 01 00
COP2	co	dest	ft reg	fs reg	OPCODE
010010	1	----	-----	-----	----- 1111 --
6 bits	1	4 bits	5 bits	5 bits	11 bits

Specifies two registers.

MacroOP field type 4

32-bit word: MacroOP field type 4

31 30 29 28 27 26	25	24 23	22 21	20 19 18 17 16	15 14 13 12 11	10 09 08 07 06 05 04 03 02 01 00
COP2	co	ftf	fsf	ft reg	fs reg	OPCODE
010010	1	--	--	-----	-----	----- 1111 --
6 bits	1	2 bits	2 bits	5 bits	5 bits	11 bits

Specifies one field each of two registers.

MacroOP field type 5

32-bit word: MacroOP field type 5

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

COP2	co	dest	it reg	is reg	Imm5	OPCODE
010010	1	0000	-----	-----	-----	------
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Specifies two registers and a 5-bit immediate value.

MacroOP field type 6

32-bit word: MacroOP field type 6

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

COP2	co	dest	Imm15	OPCODE
010010	1	0000	---------------	------
6 bits	1	4 bits	15 bits	6 bits

Specifies a 15-bit immediate value.

MacroOP field type 11

32-bit word: MacroOP field type 11

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

COP2	opcode	rt reg	id reg	----------	I
010010	0----	-----	-----	00 0000 0000	-
6 bits	5 bits	5 bits	5 bits	10 bits	1

Coprocessor transfer instructions, e.g. CFC2. The I bit (bit 0) specifies whether or not to wait for the completion of the preceding VCALLMS instruction. If the I bit is set to1, the EE Core stalls, and starts transferring upon completion of the micro sub-routine.

MacroOP field type 12

32-bit word: MacroOP field type 12

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

opcode	base	ft reg	offset
-----	-----	-----	----------------
6 bits	5 bits	5 bits	16 bits

Branch instruction using COP2 conditional signal and transfer instruction for main memory and coprocessor register.

6.1.2. Macro Instruction Operation Field

This section explains the operation fields used in operation codes.

dest field

32-bit word: MacroOP field type 0

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

COP2	co	dest	ft reg	fs reg	fd reg	OPCODE	bc
010010	1	----	-----	-----	-----	----	--
6 bits	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

The dest field specifies the FMAC units to be operated in parallel: that is, the x, y, z or w fields of the 128-bit data to be operated on. Bits 24 through 21 can be specified independently. When the bit is set to 1, the corresponding FMAC unit/field becomes effective.

Bit	Corresponding FMAC/field
24	x
23	y
22	z
21	w

bc field

32-bit word: MacroOP field type 0

31–26	25	24–21	20–16	15–11	10–06	05–02	01–00
COP2	co	dest	ft reg	fs reg	fd reg	OPCODE	bc
010010	1	----	-----	-----	-----	----	--
6 bits	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

The bc field specifies the field to be broadcasted for broadcast-series instructions, as follows:

Specified value of bc field	Broadcast field
00	x
01	y
10	z
11	w

fsf/ftf field

32-bit word: MacroOP field type 4

31–26	25	24–23	22–21	20–16	15–11	10–00
COP2	co	ftf	fsf	ft reg	fs reg	OPCODE
010010	1	--	--	-----	-----	----- 1111 --
6 bits	1	2 bits	2 bits	5 bits	5 bits	11 bits

The combinations of the fsf field with the fs reg field and the ftf field with the ft reg field specify the field to be calculated by the instruction. Bits 22 and 21 are used for the fsf field, and bits 24 and 23 for the ftf field.

Specified value for fsf/ftf field	Field to be an arithmetic object
00	x
01	y
10	z
11	w

6.2. Macro Instruction Set

This section describes the function, operation code, mnemonic, operation, flag changes, and throughput/latency of macroinstructions. They are listed in alphabetical order in mnemonic form. The descriptions also include examples, programming notes, and reference information.

BC2F : Branch on COP2 Conditional Signal

Branches if COP2 conditional signal is FALSE, i.e., branches if VU1 is not active.

Operation Code

32-bit word: MacroOP field type 12

31–26	25–21	20–16	15–00
COP2 010010	BC 01000	BCF 00000	offset ----------------
6 bits	5 bits	5 bits	16 bits

Mnemonic

```
BC2F offset
```

Operation

If the COP2 condition signal (CPCOND) sampled during execution of the previous instruction is FALSE, the program branches to the specified PC relative address. The branch delay slot is one instruction.
The branch target address is obtained by adding a signed 16-bit offset multiplied by 4 to the branch delay slot.

Throughput/Latency

2 / 2

Example

// no operation during VU execution

```
vu_run:
    bc2f   vu_idle
    nop
    j vu_run
    nop
vu_idle:
```

(The rest is omitted.)

BC2FL : Branch on COP2 Conditional Signal

Branches if COP2 conditional signal is FALSE, i.e. ,VU1 is not activated.

Operation Code

32-bit word: MacroOP field type 12

31–26	25–21	20–16	15–00
COP2	BC	BCFL	offset
010010	01000	00010	----------------
6 bits	5 bits	5 bits	16 bits

Mnemonic

BC2FL offset

Operation

Branches to the specified PC-relative address, and one instruction delay occurs, if the COP2 condition signal (CPCOND) sampled during execution of the previous instruction is FALSE. If a conditional branch does not occur, the instruction in the branch delay slot is nullified.
The branch target address is obtained by adding the signed 16-bit offset multiplied by 4 to the address of the instruction in the branch delay slot.

Throughput/Latency

2 / 2

Example

// no operation during VU execution

```
vu_run:
    bc2fl  vu_idle
    nop
    j vu_run
    nop
vu_idle:
```

(The rest is omitted.)

BC2T : Branch on COP2 Conditional Signal

Branches if COP2 conditional signal is TRUE, i.e., VU1 is activated.

Operation Code

32-bit word: MacroOP field type 12

31–26	25–21	20–16	15–00
COP2 010010	BC 01000	BCT 00001	offset ----------------
6 bits	5 bits	5 bits	16 bits

Mnemonic

```
BC2T offset
```

Operation

Branches to the specified PC-relative address, and one instruction delay occurs, if the COP2 condition signal (CPCOND) sampled during execution of the previous instruction is TRUE.
The branch target address is obtained by adding the signed 16-bit offset multiplied by 4 to the address of the instruction in the branch delay slot.

Throughput/Latency

2 / 2

Example

// no operation while VU is stopped

```
vu_idle:
    bc2t   vu_run
    nop
    j vu_idle
    nop
vu_run:
```

(The rest is omitted.)

BC2TL : Branch on COP2 Conditional signal

Branches if COP2 conditional signal is TRUE, i.e., VU1 is activated.

Operation Code

32-bit word: MacroOP field type 12

31–26	25–21	20–16	15–00
COP2 010010	BC 01000	BCTL 00011	offset ----------------
6 bits	5 bits	5 bits	16 bits

Mnemonic

```
BC2TL offset
```

Operation

Branches to the specified PC-relative address, and one instruction delay occurs, if the COP2 condition signal (CPCOND) sampled during execution of the previous instruction is TRUE. If a conditional branch does not occur, the instruction in the branch delay slot is nullified.
The branch target address is obtained by adding the signed 16-bit offset multiplied by 4 to the address of the instruction in the branch delay slot.

Throughput/Latency

2 / 2

Example

// no operation while VU is stopped

```
vu_idle:
    bc2tl  vu_run
    nop
    j vu_idle
    nop
vu_run:
```

(The rest is omitted.)

CFC2 : Transfer Integer Data from VU to EE Core

Transfers the content of CCR[2,id] to GPR[rt].

Operation Code

32-bit word: MacroOP field type 11

31-26	25-21	20-16	15-11	10-01	00
COP2 010010	CFC2 00010	rt reg -----	id reg -----	---------- 00 0000 0000	I -
6 bits	5 bits	5 bits	5 bits	10 bits	1

Mnemonic

```
CFC2     rt, id   (I=0: Without interlock)
CFC2.NI  rt, id   (I=0: Without interlock)
CFC2.I   rt, id   (I=1: With interlock)
```

Operation

Transfers the sign-extended 32-bit data of CCR[2,id], the COP2 control register in which the VU0 integer/control register is mapped, to GPR[rt], a CPU general purpose register. (However, the integer registers VI00 through VI15 are 16 bits and are mapped to the lower 16 bits of CCR[2,0] through CCR[2,15], therefore, these registers are in fact not sign-extended.)

When interlocking is specified, the CFC2 instruction stalls until the previously executed VCALLMS (micro subroutine) instruction completes, and therefore the results of the micro subroutine instruction can be read. When interlocking is not specified, the CFC2 instruction does not stall, and it immediately reads the contents of the CCR[2,id] register.

Throughput/Latency

1 / 1

Example

// Copies VI01 to t0.

```
cfc2    t0, vi1
sw      t0, 0(t1)
```

Remarks

The interlock specification does not affect synchronization with macro instructions. When the preceding macro instruction writes to the CCR[2,id] register or the CPR[2,id] register, the CFC2 instruction stalls until the macro instruction completes, regardless of the interlock specification.

CTC2 : Transfer Integer Data from EE Core to VU

Transfers the contents of GPR[rt] to CCR[2,id].

Operation Code

32-bit word: MacroOP field type 11

31–26	25–21	20–16	15–11	10–01	00
COP2	CTC2	rt reg	id reg	----------	I
010010	00110	-----	-----	00 0000 0000	-
6 bits	5 bits	5 bits	5 bits	10 bits	1

Mnemonic

```
CTC2     rt, id   (I=0: Without interlock)
CTC2.NI  rt, id   (I=0: Without interlock)
CTC2.I   rt, id   (I=1: With interlock)
```

Operation

Transfers the lower 32-bit data of GPR[rt], the CPU general purpose register, to CCR[2,id], the COP2 control register in which the integer/control register of VU0 is mapped.
When interlocking is specified, the CTC2 instruction stalls until the preceding VCALLMS (micro sub-routine) instruction completes or until a micro instruction with the M bit set to 0 is executed in that micro sub-routine. When interlocking is not specified, the CTC2 instruction immediately writes the data into the CCR[2,id] register even if the micro sub-routine is currently being executed.

Throughput/Latency

1 / 1

Example

// Copies t0 to VI01.

```
lq      t0, 0(t1)
ctc2    t0, vi1
```

Remarks

The interlock specification does not affect synchronization with macro instructions. When the preceding macro instruction writes to the CCR[2,id] register or the CPR[2,id] register, the CTC2 instruction stalls until the macro instruction completes, regardless of the interlock specification.

LQC2 : Floating-Point Data Transfer from EE Core to VU

Transfers the 128-bit data specified with GPR[base] and the immediate value offset to CPR[2,ft].

Operation Code

32-bit word: MacroOP field type 12

31–26	25–21	20–16	15–00
LQC2 110110	base -----	ft reg -----	offset ----------------
6 bits	5 bits	5 bits	16 bits

Mnemonic

```
LQC2    ft, offset(base)
```

Operation

Transfers the 128-bit data located at the address obtained by adding the sign-extended offset to the GPR[base] register, to the CPR[2,ft] register (VF[ft] floating-point register of VU0).

Throughput/Latency

1 / 1

Example

```
lqc2   vf1, 0(t0)       ;VF01 = t0[0]
qmtc2  a0, vf2          ;VF02 = a0
vmul   vf3, vf2, vf1    ;VF03 = VF02 * VF01
qmfc2  a1, vf3          ;a1 = VF03
sqc2   vf3, 0(t1)       ;t1[0] = VF03
```

Remarks

Synchronization with micro instructions is not taken into consideration. LQC2 writes to the CPR[2,ft] register (VF[ft] register) even when a micro sub-routine is currently being executed.
If a D-cache error occurs, LQC2 stalls as well as the subsequent instructions.
If the effective address (GPR[base]+offset) is not on a 128-bit boundary, that is, when the lower 4 bits of the effective address are not all 0, an address error exception is generated on the EE Core side.

QMFC2 : Floating-Point Data Transfer from VU to EE Core

Transfers data from CPR[2,fd] (VF[fd] register of VU0) to GPR[rt].

Operation Code

32-bit word: MacroOP field type 11

31–26	25–21	20–16	15–11	10–01	00
COP2 010010	QMFC2 00001	rt reg -----	fd reg -----	---------- 00 0000 0000	I -
6 bits	5 bits	5 bits	5 bits	10 bits	1

Mnemonic

```
QMFC2     rt, fd  (I=0: Without Interlock)
QMFC2.NI  rt, fd  (I=0: Without Interlock)
QMFC2.I   rt, fd  (I=1: With Interlock)
```

Operation

Transfers the contents of the CPR[2,fd] register (the floating-point register VF[fd] of VU0) to GPR[rt]. When interlocking is specified, the QMFC2 instruction stalls until the preceding VCALLMS instruction (micro sub-routine) completes, and QMFC2 reads the results of the micro sub-routine. When interlocking is not specified, QMFC2 reads the contents of CPR[2,fd] immediately even if the micro sub-routine is currently being executed.

Throughput/Latency

1 / 1

Example

```
lqc2   vf1, 0(t0)      ;VF01 = t0[0]
qmtc2  a0, vf2         ;VF02 = a0
vmul   vf3, vf2, vf1   ;VF03 = VF02 * VF01
qmfc2  a1, vf3         ;a1 = VF03
sqc2   vf3, 0(t1)      ;t1[0] = VF03
```

Remarks

The interlock specification does not affect synchronization with macro-instructions. When the preceding macro-instruction writes to CCR[2,fd] or CPR[2,fd], QMFC2 stalls until the instruction completes, regardless of the interlock specification.

QMTC2 : Floating-Point Data Transfer from EE Core to VU

Transfers data from GPR[rt] to CPR[2,fd] (VF[fd] register).

Operation Code

32-bit word: MacroOP field type 11

31–26	25–21	20–16	15–11	10–01	00
COP2 010010	QMTC2 00101	rt reg -----	fd reg -----	---------- 00 0000 0000	I -
6 bits	5 bits	5 bits	5 bits	10 bits	1

Mnemonic

```
QMTC2     rt, fd  (I=0: Without Interlock)
QMTC2.NI  rt, fd  (I=0: Without Interlock)
QMTC2.I   rt, fd  (I=1: With Interlock)
```

Operation

Transfers the contents of GPR[rt] to CPR[2,fd] (the VF[fd] register of VU0).
When interlocking is specified, QMTC2 stalls until the preceding VCALLMS instruction (micro sub-routine) completes or until a micro instruction with the M bit set to 0 is executed in that micro sub-routine. When interlocking is not specified, QMTC2 immediately writes data to CPR[2,fd] (the VF[fd] register) even if the micro sub-routine is currently being executed.

Throughput/Latency

1 / 1

Example

```
lqc2   vf1, 0(t0)      ;VF01 = t0[0]
qmtc2  a0, vf2         ;VF02 = a0
vmul   vf3, vf2, vf1   ;VF03 = VF02 * VF01
qmfc2  a1, vf3         ;a1 = VF03
sqc2   vf3, 0(t1)      ;t1[0] = VF03
```

Remarks

The interlock specification does not affect synchronization with macro-instructions. When the preceding macro-instruction writes to CCR[2,fd] register or CPR[2,fd] register, QMTC2 stalls until the instruction terminates, regardless of the interlock specification.

SQC2 : Floating-Point Data Transfer from VU to EE Core

Stores data from CPR[2,ft] (VF[ft] register) in the memory address specified with GPR[base] and offset.

Operation Code

32-bit word: MacroOP field type 12

31–26	25–21	20–16	15–00
SQC2 111110	base -----	ft reg -----	offset ----------------
6 bits	5 bits	5 bits	16 bits

Mnemonic

```
SQC2    ft, offset(base)
```

Operation

Stores data from the CPR[2,ft] register (VF[ft] register of VU0) in the address obtained by adding the sign-extended offset to GPR[base].

Throughput/Latency

1 / 1

Example

```
lqc2   vf1, 0(t0)      ;VF01 = t0[0]
qmtc2  a0, vf2         ;VF02 = a0
vmul   vf3, vf2, vf1   ;VF03 = VF02 * VF01
qmfc2  a1, vf3         ;a1 = VF03
sqc2   vf3, 0(t1)      ;t1[0] = VF03
```

Remarks

Synchronization with micro instructions is not taken into consideration. SQC2 immediately reads the contents of CPR[2,ft] (VF[ft] register of VU0) even if the micro sub-routine program is currently being executed.

When the preceding macro-instruction writes to CPR[2,ft] or CCR[2,ft], the SQC2 instruction stalls until the macro-instruction completes.

If the effective address (GPR[base]+offset) is not on a 128-bit boundary, that is, when the lower 4 bits of the execution address are not all 0, an address error exception is generated on the EE Core side.

VABS : Absolute Value

Calculates the absolute value of VF[fs] and stores the result in VF[ft].

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00
COP2	co	dest	ft reg	fs reg	VABS
010010	1	----	-----	-----	00111 1111 01
6 bits	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

`VABS.dest ft`$_{dest}$`, fs`$_{dest}$

Operation

Same as the micro instruction ABS. For details, refer to the appropriate pages in "**4.2. Upper Instruction Reference**".

VADD : Add

Calculates the sum of VF[fs] and VF[ft] and stores it in VF[fd].

Operation Code

32-bit word: MacroOP field type 1

31 30 29 28 27 26	25	24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 09 08 07 06	05 04 03 02 01 00
COP2 010010	co 1	dest ----	ft reg -----	fs reg -----	fd reg -----	VADD 101000
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

VADD.dest fd_{dest}, fs_{dest}, ft_{dest}

Operation

Same as the micro instruction ADD. For details, refer to "**4.2. Upper Instruction Reference**".

VADDi : Add to I Register

Calculates the sum of VF[fs] and the I register and stores the sum in VF[fd].

Operation Code

32-bit word: MacroOP field type 1

31-26	25	24-21	20-16	15-11	10-06	05-00
COP2 010010	co 1	dest ----	ft reg 00000	fs reg -----	fd reg -----	VADDi 100010
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

VADDi.dest fd_{dest}, fs_{dest}, **I**

Operation

Same as the micro instruction ADDi. For details, refer to "**4.2. Upper Instruction Reference**".

VADDq : Add to Q Register

Calculates the sum of VF[fs] and the Q register and stores the sum in VF[fd].

Operation Code

32-bit word: MacroOP field type 1

31–26	25	24–21	20–16	15–11	10–06	05–00
COP2	co	dest	ft reg	fs reg	fd reg	VADDq
010010	1	----	00000	-----	-----	100000
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

VADDq.dest fd$_{dest}$, fs$_{dest}$, **Q**

Operation

Same as the micro instruction ADDq. For details, refer to "**4.2. Upper Instruction Reference**".

VADDbc : Broadcast Add

Calculates the sum of each field of VF[fs] and the specified field of VF[ft] and stores the sum in the corresponding field of VF[fd].

Operation Code

32-bit word: MacroOP field type 0

31–26	25	24–21	20–16	15–11	10–06	05–02	01–00
COP2	co	dest	ft reg	fs reg	fd reg	VADD?	bc
010010	1	----	-----	-----	-----	0000	--
6 bits	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Mnemonic

```
VADDx.dest fddest, fsdest, ftx
VADDy.dest fddest, fsdest, fty
VADDz.dest fddest, fsdest, ftz
VADDw.dest fddest, fsdest, ftw
```

Operation

Same as the micro instruction ADDbc. Refer to "**4.2. Upper Instruction Reference**".

VADDA : Add to Accumulator

Calculates the sum of VF[fs] and VF[ft] and stores the sum in ACC.

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00		
COP2	co	dest	ft reg	fs reg	VADDA		
010010	1	----	-----	-----	01010	1111	00
6 bits	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

`VADDA.dest ACCdest, fsdest, ftdest`

Operation

Same as the micro instruction ADDA. Refer to "**4.2. Upper Instruction Reference**".

VADDAi : Add I Register to Accumulator

Calculates the sum of VF[fs] and the I register and stores the sum in ACC.

Operation Code

32-bit word: MacroOP field type 3

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

COP2	co	dest	ft reg	fs reg	VADDAi
010010	1	----	00000	-----	01000 1111 10
6 bits	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

VADDAi.dest **ACC**$_{dest}$, fs$_{dest}$, **I**

Operation

Same as the micro instruction ADDAi. Refer to "**4.2. Upper Instruction Reference**".

VADDAq : Add Q Register to Accumulator

Calculates the sum of VF[fs] and the Q register and stores the sum in ACC.

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00
COP2	co	dest	ft reg	fs reg	VADDAq
010010	1	----	00000	-----	01000 1111 00
6 bits	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

VADDAq.dest **ACC**dest, fsdest, **Q**

Operation

Same as the micro instruction ADDAq. Refer to "**4.2. Upper Instruction Reference**".

VADDAbc : Broadcast Add to Accumulator

Calculates sum of each field of VF[fs] and the specified field of VF[ft] and stores the sum in ACC.

Operation Code

32-bit word: MacroOP field type 2

31–26	25	24–21	20–16	15–11	10–02	01–00
COP2	co	dest	ft reg	fs reg	VADDA?	bc
010010	1	----	-----	-----	00000 1111	--
6 bits	1	4 bits	5 bits	5 bits	9 bits	2 bits

Mnemonic

```
VADDAx.dest ACCdest, fsdest, ftx
VADDAy.dest ACCdest, fsdest, fty
VADDAz.dest ACCdest, fsdest, ftz
VADDAw.dest ACCdest, fsdest, ftw
```

Operation

Same as the micro instruction ADDAbc. Refer to "**4.2. Upper Instruction Reference**".

VCALLMS : Start Micro Sub-Routine

Starts the micro sub-routine at the address specified by the immediate value.

Operation Code

32-bit word: MacroOP field type 6

31–26	25	24–21	20–06	05–00
COP2	co	dest	Imm15	VCALLMS
010010	1	0000	---------------	111000
6 bits	1	4 bits	15 bits	6 bits

Mnemonic

```
VCALLMS  Imm15
```

For Imm15, specify the address divided by 8.

Operation

Starts the sub-routine in MicroMem0 at the address specified with Imm15.

Example

```
qmtc2    vf1, a0       Micro Program
qmtc2    vf2, a1       MulMatrix:
qmtc2    vf3, a2       NOP   MULAx    ACC, VF02, VF01x
qmtc2    vf4, a3       NOP   MADDAy   ACC, VF02, VF01y
qmtc2    vf5, a4       NOP   MADDAz   ACC, VF02, VF01z
vcallms  MulMatrix     NOP   MADDAw   VF01, VF02, VF01w
qmfc2    vf1, a0       NOP   NOP @ E
```

Remarks

When executed at the same time as other external micro sub-routine calls such as MSCAL of VIFcode, the operation is undefined.

VCALLMSR : Start Micro Sub-Routine by Register

Starts the micro sub-routine at the address specified by the CMSAR0 register.

Operation Code

32-bit word: MacroOP field type 1

31–26	25	24–21	20–16	15–11	10–06	05–00
COP2 010010	co 1	dest 0000	ft reg 00000	fs reg 11011	fd reg 00000	VCALLMSR 111001
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

VCALLMSR VI27

Operation

Starts the micro sub-routine in MicroMem0. The address of the micro sub-routine must be set to CMSAR0 (CCR[2,27] register) in advance using the CTC2 instruction. (Specify the address divided by 8.)

Example

```
CTC2      rt, id
VNOP      ;requires two VNOP instructions in order to adjust
          timing.
VNOP
VCALLMSR VI27
```

Remarks

When executed at the same time as other external micro sub-routine calls such as MSCAL of VIFcode, the operation is undefined.

VCLIP : Clipping Judgment

Performs clipping judgment with the x, y, z fields of VF[fs] and the w field of VF[ft] and puts the result in the clipping flag (CF).

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00		
COP2	co	dest	ft reg	fs reg	VCLIP		
010010	1	1110	-----	-----	00111	1111	11
6 bits	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

VCLIPw.xyz fs**xyz**, ft**w**

Operation

Same as the micro instruction CLIP. Refer to "**4.2. Upper Instruction Reference**".

VDIV : Divide

Divides the fsf field of VF[fs] by the ftf field of VF[ft] and stores the quotient in the Q register.

Operation Code

32-bit word: MacroOP field type 4

31–26	25	24–23	22–21	20–16	15–11	10–00
COP2	co	ftf	fsf	ft reg	fs reg	VDIV
010010	1	--	--	-----	-----	01110 1111 00
6 bits	1	2 bits	2 bits	5 bits	5 bits	11 bits

Mnemonic

`VDIV Q, fsfsf, ftftf`

Operation

Same as the micro instruction DIV. Refer to "**4.3. Lower Instruction Reference**".

VFTOI0 : Conversion to Fixed Point

Converts the contents of VF[fs] into a fixed-point number whose fractional portion is 0 bit, and stores the result in VF[ft].

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00		
COP2	co	dest	ft reg	fs reg	VFTOI0		
010010	1	----	-----	-----	00101	1111	00
6 bits	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

`VFTOI0.dest ftdest, fsdest`

Operation

Same as the micro instruction FTOI0. Refer to "**4.2. Upper Instruction Reference**".

VFTOI4 : Conversion to Fixed Point

Converts the contents of VF[fs] into a fixed-point number whose fractional portion is 4 bits and stores the result in VF[ft].

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00
COP2	co	dest	ft reg	fs reg	VFTOI4
010010	1	----	-----	-----	00101 1111 01
6 bits	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

VFTOI4.dest ft_{dest}, fs_{dest}

Operation

Same as the micro instruction FTOI4. Refer to "**4.2. Upper Instruction Reference**".

VFTOI12 : Conversion to Fixed Point

Converts the contents of VF[fs] into a fixed-point number whose fractional portion is 12 bits and stores the result in VF[ft].

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00		
COP2	co	dest	ft reg	fs reg	VFTOI12		
010010	1	----	-----	-----	00101	1111	10
6 bits	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

```
VFTOI12.dest ftdest, fsdest
```

Operation

Same as the micro instruction FTOI12. Refer to **"4.2. Upper Instruction Reference"**.

VFTOI15 : Conversion to Fixed Point

Converts the contents of VF[fs] into a fixed-point number whose fractional portion is 15 bits and stores the result in VF[ft].

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00
COP2	co	dest	ft reg	fs reg	VFTOI15
010010	1	----	-----	-----	00101 1111 11
6 bits	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

VFTOI15.dest ft_{dest}, fs_{dest}

Operation

Same as the micro instruction FTOI15. Refer to "**4.2. Upper Instruction Reference**".

VIADD : Add Integer

Adds VI[is] and VI[it] and stores the result in VI[id].

Operation Code

32-bit word: MacroOP field type 1

31–26	25	24–21	20–16	15–11	10–06	05–00
COP2 010010	co 1	dest 0000	it reg -----	is reg -----	id reg -----	VIADD 110000
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

```
VIADD id, is, it
```

Operation

Same as the micro instruction IADD. Refer to "**4.3. Lower Instruction Reference**".

VIADDI : Add Immediate Value Integer

Adds the immediate value to VI[is] and stores the sum in VI[it].

Operation Code

32-bit word: MacroOP field type 5

31–26	25	24–21	20–16	15–11	10–06	05–00
COP2 010010	co 1	dest 0000	it reg -----	is reg -----	Imm5 -----	VIADDI 110010
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

```
VIADDI it, is, Imm5
```

Operation

Same as the micro instruction IADDI. Refer to the appropriate page in "**4.3. Lower Instruction Reference**".

VIAND : Logical Product

Calculates the AND (logical product) of VI[is] and VI[it] at every bit and stores the result in VI[id].

Operation Code

32-bit word: MacroOP field type 1

31–26	25	24–21	20–16	15–11	10–06	05–00
COP2	co	dest	it reg	is reg	id reg	VIAND
010010	1	0000	-----	-----	-----	110100
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

```
VIAND id, is, it
```

Operation

Same as the micro instruction IAND. Refer to "**4.3. Lower Instruction Reference**".

VILWR : Integer Load

Loads the specific field of the data, whose address is specified with VI[is], to VI[it] from VU Mem.

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00		
COP2	co	dest	it reg	is reg	VILWR		
010010	1	----	-----	-----	01111	1111	10
6 bits	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

VILWR.dest it, **(**is**)**dest

Operation

Same as the micro instruction ILWR. Refer to "**4.3. Lower Instruction Reference**".

VIOR : Logical Sum

Calculates the logical sum of VI[is] and VI[it] at every bit and stores the result in VI[id].

Operation Code

32-bit word: MacroOP field type 1

31–26	25	24–21	20–16	15–11	10–06	05–00
COP2	co	dest	it reg	is reg	id reg	VIOR
010010	1	0000	-----	-----	-----	110101
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

```
VIOR id, is, it
```

Operation

Same as the micro instruction IOR. Refer to "**4.3. Lower Instruction Reference**".

VISUB : Integer Subtract

Subtracts VI[it] from VI[is] and stores the result in VI[id].

Operation Code

32-bit word: MacroOP field type 1

31–26	25	24–21	20–16	15–11	10–06	05–00
COP2 010010	co 1	dest 0000	it reg -----	is reg -----	id reg -----	VISUB 110001
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

```
VISUB id, is, it
```

Operation

Same as the micro instruction ISUB. Refer to "**4.3. Lower Instruction Reference**".

VISWR : Integer Store

Stores data from VI[it] in VU Mem. The destination address is the dest field specified by VI[is].

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00
COP2	co	dest	it reg	is reg	VISWR
010010	1	----	-----	-----	01111 1111 11
6 bits	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

VISWR.dest it, **(**is**)**dest

Operation

Same as the micro instruction ISWR. Refer to "**4.3. Lower Instruction Reference**".

VITOF0 : Conversion to Floating-Point Number

Considers the value of VF[fs] as a fixed-point number whose fractional portion is 0 bits, converts it to floating-point, and stores the result in VF[ft].

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00
COP2 010010	co 1	dest ----	ft reg -----	fs reg -----	VITOF0 00100 1111 00
6 bits	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

`VITOF0.dest ftdest, fsdest`

Operation

Same as the micro instruction ITOF0. Refer to "**4.2. Upper Instruction Reference**".

VITOF4 : Conversion to Floating-Point Number

Considers the value of VF[fs] as a fixed-point number whose fractional portion is 4 bits, converts it to floating-point, and stores the result in VF[ft].

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00		
COP2	co	dest	ft reg	fs reg	VITOF4		
010010	1	----	-----	-----	00100	1111	01
6 bits	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

`VITOF4.dest ftdest, fsdest`

Operation

Same as the micro instruction ITOF4. Refer to "**4.2. Upper Instruction Reference**".

VITOF12 : Conversion to Floating-Point Number

Considers the value of VF[fs] as a fixed-point number whose fractional portion is 12 bits, converts it to floating-point, and stores the result in VF[ft].

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00
COP2	co	dest	ft reg	fs reg	VITOF12
010010	1	----	-----	-----	00100 1111 10
6 bits	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

`VITOF12.dest ftdest, fsdest`

Operation

Same as the micro instruction ITOF12. Refer to "**4.2. Upper Instruction Reference**".

VITOF15 : Conversion to Floating-Point Number

Considers the value of VF[fs] as a fixed-point number whose fractional portion is 15 bits, converts it to floating-point, and stores the result in VF[ft].

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00
COP2	co	dest	ft reg	fs reg	VITOF15
010010	1	----	-----	-----	00100 1111 11
6 bits	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

`VITOF15.dest ftdest, fsdest`

Operation

Same as the micro instruction ITOF15. Refer to **"4.2. Upper Instruction Reference"**.

VLQD : Load with Pre-Decrement

Loads data from VU Mem at the address VI[is] - 1 into VF[ft].

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00
COP2	co	dest	ft reg	is reg	VLQD
010010	1	----	-----	-----	01101 1111 10
6 bits	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

```
VLQD.dest ftdest, (--is)dest
```

Operation

Same as the micro instruction LQD. Refer to "**4.3. Lower Instruction Reference**".

VLQI : Load with Post-Increment

Loads the data specified by VI[is] from VU Mem to VF[ft] and adds 1 to VI[is].

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00
COP2	co	dest	ft reg	is reg	VLQI
010010	1	----	-----	-----	01101 1111 00
6 bits	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

VLQI.dest ft$_{dest}$, **(**is**++)**$_{dest}$

Operation

Same as the micro instruction LQI. Refer to "**4.3. Lower Instruction Reference**".

VMADD : Product Sum

Adds the value of ACC to the product of VF[fs] and VF[ft], and stores the result in VF[fd].

Operation Code

32-bit word: MacroOP field type 1

31–26	25	24–21	20–16	15–11	10–06	05–00
COP2 010010	co 1	dest ----	ft reg -----	fs reg -----	fd reg -----	VMADD 101001
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

VMADD.dest fd_{dest}, fs_{dest}, ft_{dest}

Operation

Same as the micro instruction MADD. Refer to "**4.2. Upper Instruction Reference**".

VMADDi : Product Sum; with I Register

Adds the product of each field of VF[fs] and the I register to the corresponding value of ACC, and stores the result in VF[fd].

Operation Code

32-bit word: MacroOP field type 1

31–26	25	24–21	20–16	15–11	10–06	05–00
COP2 010010	co 1	dest ----	ft reg 00000	fs reg -----	fd reg -----	VMADDi 100011
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

VMADDi.dest fd$_{dest}$, fs$_{dest}$, **I**

Operation

Same as the micro instruction MADDi. Refer to **"4.2. Upper Instruction Reference"**.

VMADDq : Product Sum; with Q Register

Adds the product of each field of VF[fs] and the Q register to the corresponding value of ACC and stores the result in VF[fd].

Operation Code

32-bit word: MacroOP field type 1

31–26	25	24–21	20–16	15–11	10–06	05–00
COP2 010010	co 1	dest ----	ft reg 00000	fs reg -----	fd reg -----	VMADDq 100001
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

```
VMADDq.dest fddest, fsdest, Q
```

Operation

Same as the micro instruction MADDq. Refer to "**4.2. Upper Instruction Reference**".

VMADDbc : Broadcast Product Sum

Adds the product of each field of VF[fs] and the specified field of VF[ft] to the corresponding value of ACC and stores the result in VF[fd].

Operation Code

32-bit word: MacroOP field type 0

31–26	25	24–21	20–16	15–11	10–06	05–02	01–00
COP2 010010	co 1	dest ----	ft reg -----	fs reg -----	fd reg -----	VMADD? 0010	bc --
6 bits	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Mnemonic

```
VMADDx.dest fd_dest, fs_dest, ft_x
VMADDy.dest fd_dest, fs_dest, ft_y
VMADDz.dest fd_dest, fs_dest, ft_z
VMADDw.dest fd_dest, fs_dest, ft_w
```

Operation

Same as the micro instruction MADDbc. Refer to "**4.2. Upper Instruction Reference**".

VMADDA : Product Sum; to Accumulator

Adds the product of VF[fs] and VF[ft] to the value of ACC and stores the result in ACC.

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00
COP2	co	dest	ft reg	fs reg	VMADDA
010010	1	----	-----	-----	01010 1111 01
6 bits	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

VMADDA.dest **ACC**dest, fsdest, ftdest

Operation

Same as the micro instruction MADDA. Refer to "**4.2. Upper Instruction Reference**".

VMADDAi : Product Sum; with I Register, to Accumulator

Adds the product of each field of VF[fs] and the I register to the corresponding value of ACC and stores the result in ACC.

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00
COP2	co	dest	ft reg	fs reg	VMADDAi
010010	1	----	00000	-----	01000 1111 11
6 bits	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

VMADDAi.dest **ACC**dest, fsdest, **I**

Operation

Same as the micro instruction MADDAi. Refer to "**4.2. Upper Instruction Reference**".

VMADDAq : Product Sum; with Q Register, to Accumulator

Adds the product of each field of VF[fs] and the Q register to the corresponding field of ACC and stores the result in ACC.

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00
COP2	co	dest	ft reg	fs reg	VMADDAq
010010	1	----	00000	-----	01000 1111 01
6 bits	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

VMADDAq.dest **ACC**dest, fsdest, **Q**

Operation

Same as the micro instruction MADDAq. Refer to "**4.2. Upper Instruction Reference**".

VMADDAbc : Broadcast Product Sum; to Accumulator

Adds the product of each field of VF[fs] and the specified field of VF[ft] to the corresponding field of ACC and stores the result in ACC.

Operation Code

32-bit word: MacroOP field type 2

31–26	25	24–21	20–16	15–11	10–02	01–00
COP2	co	dest	ft reg	fs reg	VMADDA?	bc
010010	1	----	-----	-----	00010 1111	--
6 bits	1	4 bits	5 bits	5 bits	9 bits	2 bits

Mnemonic

```
VMADDAx.dest ACCdest, fsdest, ftx
VMADDAy.dest ACCdest, fsdest, fty
VMADDAz.dest ACCdest, fsdest, ftz
VMADDAw.dest ACCdest, fsdest, ftw
```

Operation

Same as the micro instruction MADDAbc. Refer to "**4.2. Upper Instruction Reference**".

VMAX : Maximum Value

Compares the value of VF[fs] with that of VF[ft] and stores the greater value in VF[fd].

Operation Code

32-bit word: MacroOP field type 1

31–26	25	24–21	20–16	15–11	10–06	05–00
COP2 010010	co 1	dest ----	ft reg -----	fs reg -----	fd reg -----	VMAX 101011
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

VMAX.dest fd_{dest}, fs_{dest}, ft_{dest}

Operation

Same as the micro instruction MAX. Refer to "**4.2. Upper Instruction Reference**".

VMAXi : Maximum Value

Compares the value of each field of VF[fs] with that of the I register and stores the greater value in the corresponding field of VF[fd].

Operation Code

32-bit word: MacroOP field type 1

31–26	25	24–21	20–16	15–11	10–06	05–00
COP2 010010	co 1	dest ----	ft reg 00000	fs reg -----	fd reg -----	VMAXi 011101
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

```
VMAXi.dest fd_dest, fs_dest, I
```

Operation

Same as the micro instruction MAXi. Refer to "**4.2. Upper Instruction Reference**".

VMAXbc : Maximum Value

Compares each field of VF[fs] with the specified field of VF[ft] and stores the greater value in the corresponding field of VF[fd].

Operation Code

32-bit word: MacroOP field type 0

31–26	25	24–21	20–16	15–11	10–06	05–02	01–00
COP2 010010	co 1	dest ----	ft reg -----	fs reg -----	fd reg -----	VMAX? 0100	bc --
6 bits	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Mnemonic

```
VMAXx.dest fddest, fsdest, ftx
VMAXy.dest fddest, fsdest, fty
VMAXz.dest fddest, fsdest, ftz
VMAXw.dest fddest, fsdest, ftw
```

Operation

Same as the micro instruction MAXbc. Refer to "**4.2. Upper Instruction Reference**".

VMFIR : Transfer from Integer Register to Floating-Point Register

Transfers the contents of VI[is] to VF[ft].

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00		
COP2	co	dest	ft reg	is reg	VMFIR		
010010	1	----	-----	-----	01111	1111	01
6 bits	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

VMFIR.dest ft$_{dest}$, is

Operation

Same as the micro instruction MFIR. Refer to "**4.2. Upper Instruction Reference**".

VMINI : Minimum Value

Compares the contents of VF[fs] with VF[ft] and stores the smaller value in VF[fd].

Operation Code

32-bit word: MacroOP field type 1

31–26	25	24–21	20–16	15–11	10–06	05–00
COP2 010010	co 1	dest ----	ft reg -----	fs reg -----	fd reg -----	VMINI 101111
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

```
VMINI.dest fddest, fsdest, ftdest
```

Operation

Same as the micro instruction MINI. Refer to "**4.2. Upper Instruction Reference**".

VMINIi : Minimum Value

Compares each field of VF[fs] with the I register and stores the smaller value in the corresponding field of VF[fd].

Operation Code

32-bit word: MacroOP field type 1

31–26	25	24–21	20–16	15–11	10–06	05–00
COP2 010010	co 1	dest ----	ft reg 00000	fs reg -----	fd reg -----	VMINIi 011111
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

VMINIi.dest fd_{dest}, fs_{dest}, **I**

Operation

Same as the micro instruction MINIi. Refer to "**4.2. Upper Instruction Reference**".

VMINIbc : Minimum Value

Compares each field of VF[fs] with the specified field of VF[ft] and stores the smaller value in the corresponding field of VF[fd].

Operation Code

32-bit word: MacroOP field type 0

31–26	25	24–21	20–16	15–11	10–06	05–02	01–00
COP2 010010	co 1	dest ----	ft reg -----	fs reg -----	fd reg -----	VMINI? 0101	bc --
6 bits	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Mnemonic

```
VMINIx.dest fddest, fsdest, ftx
VMINIy.dest fddest, fsdest, fty
VMINIz.dest fddest, fsdest, ftz
VMINIw.dest fddest, fsdest, ftw
```

Operation

Same as the micro instruction MINIbc. Refer to "**4.2. Upper Instruction Reference**".

VMOVE : Transfer between Floating-Point Registers

Transfers the value of VF[fs] to VF[ft].

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00
COP2	co	dest	ft reg	fs reg	VMOVE
010010	1	----	-----	-----	01100 1111 00
6 bits	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

VMOVE.dest ft_{dest}, fs_{dest}

Operation

Same as the micro instruction MOVE. Refer to "**4.3. Lower Instruction Reference**".

VMR32 : Vector Rotate

Rotates the fields of VF[fs] to the right and transfers them to VF[ft].

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00
COP2	co	dest	ft reg	fs reg	VMR32
010010	1	----	-----	-----	01100 1111 01
6 bits	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

VMR32.dest ft_{dest}, fs_{dest}

Operation

Same as the micro instruction MR32. Refer to "**4.3. Lower Instruction Reference**".

VMSUB : Multiply and Subtract

Subtracts the product of VF[fs] and VF[ft] from the value of ACC and stores the result in VF[fd].

Operation Code

32-bit word: MacroOP field type 1

31–26	25	24–21	20–16	15–11	10–06	05–00
COP2 010010	co 1	dest ----	ft reg -----	fs reg -----	fd reg -----	VMSUB 101101
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

VMSUB.dest fd_{dest}, fs_{dest}, ft_{dest}

Operation

Same as the micro instruction MSUB. Refer to "**4.2. Upper Instruction Reference**".

VMSUBi : Multiply and Subtract with I Register

Subtracts the product of the values of each field of VF[fs] and the I register from the corresponding field of ACC, and stores the result in the corresponding field of VF[fd].

Operation Code

32-bit word: MacroOP field type 1

31–26	25	24–21	20–16	15–11	10–06	05–00
COP2 010010	co 1	dest ----	ft reg 00000	fs reg -----	fd reg -----	VMSUBi 100111
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

VMSUBi.dest fd$_{dest}$, fs$_{dest}$, **I**

Operation

Same as the micro instruction MSUBi. Refer to "**4.2. Upper Instruction Reference**".

VMSUBq : Multiply and Subtract; Q Register

Subtracts the product of the values of each field of VF[fs] and the Q register from the corresponding field of ACC, and stores the result in the corresponding field of VF[fd].

Operation Code

32-bit word: MacroOP field type 1

31–26	25	24–21	20–16	15–11	10–06	05–00
COP2 010010	co 1	dest ----	ft reg 00000	fs reg -----	fd reg -----	VMSUBq 100101
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

VMSUBq.dest fd$_{dest}$, fs$_{dest}$, **Q**

Operation

Same as the micro instruction MSUBq. Refer to **"4.2. Upper Instruction Reference"**.

VMSUBbc : Broadcast Multiply and Subtract

Subtracts the product of each field of VF[fs] and the specified field of VF[ft] from the corresponding field of ACC, and stores the result in the corresponding field of VF[fd].

Operation Code

32-bit word: MacroOP field type 0

31–26	25	24–21	20–16	15–11	10–06	05–02	01–00
COP2 010010	co 1	dest ----	ft reg -----	fs reg -----	fd reg -----	VMSUB? 0011	bc --
6 bits	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Mnemonic

```
VMSUBx.dest fddest, fsdest, ftx
VMSUBy.dest fddest, fsdest, fty
VMSUBz.dest fddest, fsdest, ftz
VMSUBw.dest fddest, fsdest, ftw
```

Operation

Same as the micro instruction MSUBbc. Refer to "**4.2. Upper Instruction Reference**".

VMSUBA : Multiply and Subtract; to Accumulator

Subtracts the product of VF[fs] and VF[ft] from ACC, and stores the result in ACC.

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00		
COP2	co	dest	ft reg	fs reg	VMSUBA		
010010	1	----	-----	-----	01011	1111	01
6 bits	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

VMSUBA.dest **ACC**dest, fsdest, ftdest

Operation

Same as the micro instruction MSUBA. Refer to "**4.2. Upper Instruction Reference**".

VMSUBAi : Multiply and Subtract; with I Register, to Accumulator

Subtracts the product of each field of VF[fs] and the I register from the corresponding field of ACC, and stores the result in ACC.

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00		
COP2	co	dest	ft reg	fs reg	VMSUBAi		
010010	1	----	00000	-----	01001	1111	11
6 bits	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

VMSUBAi.dest **ACC**dest, fsdest, **I**

Operation

Same as the micro instruction MSUBAi. Refer to "**4.2. Upper Instruction Reference**".

VMSUBAq : Multiply and Subtract; with Q Register, to Accumulator

Subtracts the product of each field of VF[fs] and the Q register from the corresponding field of ACC, and stores the result in the corresponding field of ACC.

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00
COP2	co	dest	ft reg	fs reg	VMSUBAq
010010	1	----	00000	-----	01001 1111 01
6 bits	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

VMSUBAq.dest **ACC**dest, fsdest, **Q**

Operation

Same as the micro instruction MSUBAq. Refer to "**4.2. Upper Instruction Reference**".

VMSUBAbc : Broadcast Multiply and Subtract; to Accumulator

Subtracts the product of each field of VF[fs] and the specified field of VF[ft] from the corresponding field of the ACC, and stores the result in the corresponding field of ACC.

Operation Code

32-bit word: MacroOP field type 2

31–26	25	24–21	20–16	15–11	10–02	01–00
COP2 010010	co 1	dest ----	ft reg -----	fs reg -----	VMSUBA? 00011 1111	bc --
6 bits	1	4 bits	5 bits	5 bits	9 bits	2 bits

Mnemonic

VMSUBAx.dest **ACC**$_{dest}$, fs$_{dest}$, ft$_{\mathbf{x}}$

VMSUBAy.dest **ACC**$_{dest}$, fs$_{dest}$, ft$_{\mathbf{y}}$

VMSUBAz.dest **ACC**$_{dest}$, fs$_{dest}$, ft$_{\mathbf{z}}$

VMSUBAw.dest **ACC**$_{dest}$, fs$_{dest}$, ft$_{\mathbf{w}}$

Operation

Same as the micro instruction MSUBAbc. Refer to "**4.2. Upper Instruction Reference**".

VMTIR : Transfer from Floating-Point Register to Integer Register

Transfers the lower 16 bits of the field specified by fsf of VF[fs] to VI[it].

Operation Code

32-bit word: MacroOP field type 4

31–26	25	24–23	22–21	20–16	15–11	10–06	05–02	01–00
COP2	co	ftf	fsf	it reg	fs reg	VMTIR		
010010	1	00	--	-----	-----	01111	1111	00
6 bits	1	2 bits	2 bits	5 bits	5 bits	11 bits		

Mnemonic

```
VMTIR it, fs.fsf
```

Operation

Same as the micro instruction MTIR. Refer to "**4.3. Lower Instruction Reference**".

VMUL : Multiply

Multiplies VF[fs] and VF[ft], and stores the product in VF[fd].

Operation Code

32-bit word: MacroOP field type 1

31-26	25	24-21	20-16	15-11	10-06	05-00
COP2 010010	co 1	dest ----	ft reg -----	fs reg -----	fd reg -----	VMUL 101010
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

`**VMUL**.dest fd`$_{dest}$`, fs`$_{dest}$`, ft`$_{dest}$

Operation

Same as the micro instruction MUL. Refer to "**4.2. Upper Instruction Reference**".

VMULi : Multiply; by I Register

Multiplies the values of each field of VF[fs] by the I register, and stores the product in the corresponding field of VF[fd].

Operation Code

32-bit word: MacroOP field type 1

31–26	25	24–21	20–16	15–11	10–06	05–00
COP2 010010	co 1	dest ----	ft reg 00000	fs reg -----	fd reg -----	VMULi 011110
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

```
VMULi.dest fd_dest, fs_dest, I
```

Operation

Same as the micro instruction MULi. Refer to "**4.2. Upper Instruction Reference**".

VMULq : Multiply; by Q Register

Multiplies the values of each field of VF[fs] by the value of the Q register, and stores the product in the corresponding field of VF[fd].

Operation Code

32-bit word: MacroOP field type 1

31–26	25	24–21	20–16	15–11	10–06	05–00
COP2 010010	co 1	dest ----	ft reg 00000	fs reg -----	fd reg -----	VMULq 011100
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

```
VMULq.dest fddest, fsdest, Q
```

Operation

Same as the micro instruction MULq. Refer to "**4.2. Upper Instruction Reference**".

VMULbc : Broadcast Multiply

Multiplies each field of VF[fs] by the specified field of VF[ft], and stores the product in the corresponding field of VF[fd].

Operation Code

32-bit word: MacroOP field type 0

31–26	25	24–21	20–16	15–11	10–06	05–02	01–00
COP2 010010	co 1	dest ----	ft reg -----	fs reg -----	fd reg -----	VMUL? 0110	bc --
6 bits	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Mnemonic

VMULx.dest fd_{dest}, fs_{dest}, $ft_{\mathbf{x}}$
VMULy.dest fd_{dest}, fs_{dest}, $ft_{\mathbf{y}}$
VMULz.dest fd_{dest}, fs_{dest}, $ft_{\mathbf{z}}$
VMULw.dest fd_{dest}, fs_{dest}, $ft_{\mathbf{w}}$

Operation

Same as the micro instruction MULbc. Refer to "**4.2. Upper Instruction Reference**".

VMULA : Multiply; to Accumulator

Multiplies VF[fs] by VF[ft] and stores the product in ACC.

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00		
COP2	co	dest	ft reg	fs reg	VMULA		
010010	1	----	-----	-----	01010	1111	10
6 bits	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

VMULA.dest **ACC**dest, fsdest, ftdest

Operation

Same as the micro instruction MULA. Refer to "**4.2. Upper Instruction Reference**".

VMULAi : Multiply by I Register; to Accumulator

Multiplies each field of VF[fs] by the value of the I register, and stores the product in ACC.

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00
COP2	co	dest	ft reg	fs reg	VMULAi
010010	1	----	00000	-----	00111 1111 10
6 bits	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

VMULAi.dest **ACC**$_{dest}$, fs$_{dest}$, **I**

Operation

Same as the micro instruction MULAi. Refer to "**4.2. Upper Instruction Reference**".

VMULAq : Multiply by Q Register; to Accumulator

Multiplies each field of VF[fs] by value of the Q register, and stores the product in ACC.

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00		
COP2	co	dest	ft reg	fs reg	VMULAq		
010010	1	----	00000	-----	00111	1111	00
6 bits	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

`VMULAq.dest ACCdest, fsdest, Q`

Operation

Same as the micro instruction MULAq. Refer to "**4.2. Upper Instruction Reference**".

VMULAbc : Broadcast Multiply; to Accumulator

Multiplies each field of VF[fs] by the specified field of VF[ft], and stores the product in the corresponding field of ACC.

Operation Code

32-bit word: MacroOP field type 2

31–26	25	24–21	20–16	15–11	10–02		01–00
COP2	co	dest	ft reg	fs reg	VMULA?		bc
010010	1	----	-----	-----	00110	1111	--
6 bits	1	4 bits	5 bits	5 bits	9 bits		2 bits

Mnemonic

```
VMULAx.dest ACCdest, fsdest, ftx
VMULAy.dest ACCdest, fsdest, fty
VMULAz.dest ACCdest, fsdest, ftz
VMULAw.dest ACCdest, fsdest, ftw
```

Operation

Same as the micro instruction MULAbc. Refer to "**4.2. Upper Instruction Reference**".

VNOP : No Operation

No operation is performed.

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00		
COP2	co	dest	ft reg	fs reg	VNOP		
010010	1	0000	00000	00000	01011	1111	11
6 bits	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

`VNOP`

Operation

No operation is performed. The status flag does not change.

Throughput/Latency

1 / 4

VOPMULA : Vector Outer Product

Calculates the first part of the vector outer product of VF[fs] and VF[ft] and stores the result in ACC.

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00		
COP2	co	dest	ft reg	fs reg	VOPMULA		
010010	1	1110	-----	-----	01011	1111	10
6 bits	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

VOPMULA.xyz ACCxyz, fs**xyz**, ft**xyz**

Operation

Same as the micro instruction OPMULA. Refer to "**4.2. Upper Instruction Reference**".

VOPMSUB : Vector Outer Product

Calculates the last part of the vector outer product using VF[fs], VF[ft] and ACC, and stores the result in VF[fd].

Operation Code

32-bit word: MacroOP field type 1

31–26	25	24–21	20–16	15–11	10–06	05–00
COP2 010010	co 1	dest 1110	ft reg -----	fs reg -----	fd reg -----	VOPMSUB 101110
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

VOPMSUB.xyz fd**xyz**, fs**xyz**, ft**xyz**

Operation

Same as the micro instruction OPMSUB. Refer to **"4.2. Upper Instruction Reference"**.

VRGET : Get Random Numbers

Obtains random numbers and stores them in VF[ft].

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00		
COP2	co	dest	ft reg	fs reg	VRGET		
010010	1	----	-----	00000	10000	1111	01
6 bits	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

VRGET.dest ft

Operation

Same as the micro instruction RGET. Refer to "**4.3. Lower Instruction Reference**".

VRINIT : Random Number Initial Set

Sets the R register to the field specified by fsf of VF[fs].

Operation Code

32-bit word: MacroOP field type 4

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

COP2	co	ftf	fsf	ft reg	fs reg	VRINIT
010010	1	00	--	00000	-----	10000 1111 10
6 bits	1	2 bits	2 bits	5 bits	5 bits	11 bits

Mnemonic

VRINIT R, fs_{fsf}

Operation

Same as the micro instruction RINIT. Refer to "**4.3. Lower Instruction Reference**".

VRNEXT : New Random Numbers

Generates new random numbers and stores them in VF[ft].

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00
COP2	co	dest	ft reg	fs reg	VRNEXT
010010	1	----	-----	00000	10000 1111 00
6 bits	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

```
VRNEXT.dest ftdest, R
```

Operation

Same as the micro instruction RNEXT. Refer to "**4.3. Lower Instruction Reference**".

VRSQRT : Square Root Division

Divides the fsf field of VF[fs] by the square root of the ftf field of VF[ft], and stores the result in the Q register.

Operation Code

32-bit word: MacroOP field type 4

31–26	25	24–23	22–21	20–16	15–11	10–00
COP2	co	ftf	fsf	ft reg	fs reg	VRSQRT
010010	1	--	--	-----	-----	01110 1111 10
6 bits	1	2 bits	2 bits	5 bits	5 bits	11 bits

Mnemonic

VRSQRT Q, fs_{fsf}, ft_{ftf}

Operation

Same as the micro instruction RSQRT. Refer to "**4.3. Lower Instruction Reference**".

VRXOR : Random Number Set

Takes the exclusive OR of the field specified with fsf of VF[fs] and the R register, and sets the R register to the result.

Operation Code

32-bit word: MacroOP field type 4

31–26	25	24–23	22–21	20–16	15–11	10–00		
COP2	co	ftf	fsf	ft reg	fs reg	VRXOR		
010010	1	00	--	00000	-----	10000	1111	11
6 bits	1	2 bits	2 bits	5 bits	5 bits	11 bits		

Mnemonic

VRXOR R, fs_{fsf}

Operation

Same as the micro instruction RXOR. Refer to **"4.3. Lower Instruction Reference"**.

VSQD : Store with Pre-Decrement

Stores the contents of VF[fs] at the VU Mem0 address VI[it] - 1.

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00		
COP2	co	dest	it reg	fs reg	VSQD		
010010	1	----	-----	-----	01101	1111	11
6 bits	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

VSQD.dest fs$_{dest}$, **(--**it**)**$_{dest}$

Operation

Same as the micro instruction SQD. Refer to "**4.3. Lower Instruction Reference**".

VSQI : Store with Post-Increment

Stores the contents of VF[fs] in the VU Mem0 address specified by VI[it] and adds 1 to VI[it].

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00
COP2	co	dest	it reg	fs reg	VSQI
010010	1	----	-----	-----	01101 1111 01
6 bits	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

`VSQI.dest fsdest, (it++)dest`

Operation

Same as the micro instruction SQI. Refer to "**4.3. Lower Instruction Reference**".

VSQRT : Square Root

Obtains the square root of the field specified by the ftf field of VF[ft], and stores the result in the Q register.

Operation Code

32-bit word: MacroOP field type 4

31–26	25	24–23	22–21	20–16	15–11	10–00
COP2	co	ftf	fsf	ft reg	fs reg	VSQRT
010010	1	--	00	-----	00000	01110 1111 01
6 bits	1	2 bits	2 bits	5 bits	5 bits	11 bits

Mnemonic

VSQRT Q, ft$_{ftf}$

Operation

Same as the micro instruction SQRT. Refer to "**4.3. Lower Instruction Reference**".

VSUB : Subtract

Subtracts VF[ft] from VF[fs] and stores the result in VF[fd].

Operation Code

32-bit word: MacroOP field type 1

31 30 29 28 27 26	25	24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 09 08 07 06	05 04 03 02 01 00
COP2 010010	co 1	dest ----	ft reg -----	fs reg -----	fd reg -----	VSUB 101100
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

VSUB.dest fd

Operation

Same as the micro instruction SUB. Refer to "**4.2. Upper Instruction Reference**".

VSUBi : Subtract I Register

Subtracts the I register from each field of VF[fs], and stores the result in the corresponding field of VF[fd].

Operation Code

32-bit word: MacroOP field type 1

31–26	25	24–21	20–16	15–11	10–06	05–00
COP2	co	dest	ft reg	fs reg	fd reg	VSUBi
010010	1	----	00000	-----	-----	100110
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

VSUBi.dest fd_{dest}, fs_{dest}, **I**

Operation

Same as the micro instruction SUBi. Refer to "**4.2. Upper Instruction Reference**".

VSUBq : Subtract Q Register

Subtracts the Q register from each field of VF[fs], and stores the result in the corresponding field of VF[ft].

Operation Code

32-bit word: MacroOP field type 1

31–26	25	24–21	20–16	15–11	10–06	05–00
COP2	co	dest	ft reg	fs reg	fd reg	VSUBq
010010	1	----	00000	-----	-----	100100
6 bits	1	4 bits	5 bits	5 bits	5 bits	6 bits

Mnemonic

VSUBq.dest fd_{dest}, fs_{dest}, **Q**

Operation

Same as the micro instruction SUBq. Refer to "**4.2. Upper Instruction Reference**".

VSUBbc : Broadcast Subtract

Subtracts the specified field of VF[ft] from each field of VF[fs] and stores the result in the corresponding field of VF[fd].

Operation Code

32-bit word: MacroOP field type 0

31–26	25	24–21	20–16	15–11	10–06	05–02	01–00
COP2 010010	co 1	dest ----	ft reg -----	fs reg -----	fd reg -----	VSUB? 0001	bc --
6 bits	1	4 bits	5 bits	5 bits	5 bits	4 bits	2 bits

Mnemonic

VSUBx.dest fd$_{dest}$, fs$_{dest}$, ft**x**

VSUBy.dest fd$_{dest}$, fs$_{dest}$, ft**y**

VSUBz.dest fd$_{dest}$, fs$_{dest}$, ft**z**

VSUBw.dest fd$_{dest}$, fs$_{dest}$, ft**w**

Operation

Same as the micro instruction SUBbc. Refer to "**4.2. Upper Instruction Reference**".

VSUBA : Subtract; to Accumulator

Subtracts VF[ft] from VF[fs], and stores the result in ACC.

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00		
COP2	co	dest	ft reg	fs reg	VSUBA		
010010	1	----	-----	-----	01011	1111	00
6 bits	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

VSUBA.dest **ACC**$_{dest}$, fs$_{dest}$, ft$_{dest}$

Operation

Same as the micro instruction SUBA. Refer to "**4.2. Upper Instruction Reference**".

VSUBAi : Subtract I Register; to Accumulator

Subtracts the I register from each field of VF[fs], and stores the result in the corresponding field of ACC.

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00		
COP2	co	dest	ft reg	fs reg	VSUBAi		
010010	1	----	00000	-----	01001	1111	10
6 bits	1	4 bits	5 bits	5 bits	11 bits		

Mnemonic

VSUBAi.dest **ACC**$_{dest}$, fs$_{dest}$, **I**

Operation

Same as the micro instruction SUBAi. Refer to "**4.2. Upper Instruction Reference**".

VSUBAq : Subtract Q Register; to Accumulator

Subtracts the Q register from each field of VF[fs], and stores the result in the corresponding field of ACC.

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00
COP2	co	dest	ft reg	fs reg	VSUBAq
010010	1	----	00000	-----	01001 1111 00
6 bits	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

VSUBAq.dest **ACC**dest, fsdest, **Q**

Operation

Same as the micro instruction SUBAq. Refer to "**4.2. Upper Instruction Reference**".

VSUBAbc : Broadcast Subtract; to Accumulator

Subtracts the specified field of VF[ft] from each field of VF[fs], and stores the result in the corresponding field of ACC.

Operation Code

32-bit word: MacroOP field type 2

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00

COP2	co	dest	ft reg	fs reg	VSUBA?	bc
010010	1	----	-----	-----	00001 1111	--
6 bits	1	4 bits	5 bits	5 bits	9 bits	2 bits

Mnemonic

```
VSUBAx.dest fddest, fsdest, ftx
VSUBAy.dest fddest, fsdest, fty
VSUBAz.dest fddest, fsdest, ftz
VSUBAw.dest fddest, fsdest, ftw
```

Operation

Same as the micro instruction SUBAbc. Refer to "**4.2. Upper Instruction Reference**".

VWAITQ : Q Register Synchronize

Stops the VU until the result is written to the Q register.

Operation Code

32-bit word: MacroOP field type 3

31–26	25	24–21	20–16	15–11	10–00
COP2	co	dest	ft reg	fs reg	VWAITQ
010010	1	0000	00000	00000	01110 1111 11
6 bits	1	4 bits	5 bits	5 bits	11 bits

Mnemonic

`VWAITQ`

Operation

Same as the micro instruction WAITQ. Refer to "**4.3. Lower Instruction Reference**".

(This page is left blank intentionally)

7. Appendix

7.1. Sample Micro Programs

Examples of micro programs are shown below. Note that these programs are insufficient as practical programs, since they have been created as samples.
NOP* indicates a NOP that can be deleted. (When deleted, however, the subsequent instruction stalls). There is no NOP in the Lower instruction, so substitute a meaningless instruction such as MOVE VF00, VF00 for NOP.

Register allocation

```
;Floating point register
;       x        y        z       w
;VF00:  0        0        0       1       ; Special constant register
;VF01:  1        1        1       1       ; All 1 register (would be helpful to have)
;VF02:  L0x      L1x      L2x     L3x     ; Normalized light source vector (X)
;VF03:  L0y      L1y      L2y     L3y     ; Normalized light source vector (Y)
;VF04:  L0z      L1z      L2z     L3z     ; Normalized light source vector (Z)
;VF05:  R0       G0       B0      -       ; Light source color RGB 0
;VF06:  R1       G1       B1      -       ; Light source color RGB 1
;VF07:  R2       G2       B2      -       ; Light source color RGB 2
;VF08:  R3       G3       B3      -       ; Light source color RGB 3
;VF09:  Ra       Ga       Ba      r0      ; Peripheral light
;VF10:  M00      M01      M02     M03     ; Coordinate conversion matrix (row 0)
;VF11:  M10      M11      M12     M13     ; Coordinate conversion matrix (row 1)
;VF12:  M20      M21      M22     M23     ; Coordinate conversion matrix (row 2)
;VF13:  M30      M31      M32     M33     ; Coordinate conversion matrix (row 3)
;VF14:  MINI     MAX      SEED    MR      ; MINI=0, MAX=255
;VF15:  -        -        -       -
;VF16:  Locally used by each subroutine
;VF17:  Locally used by each subroutine
;VF18:  Locally used by each subroutine
;VF19:  Locally used by each subroutine
;VF20:  Locally used by each subroutine
;VF21:  Locally used by each subroutine
;VF22:  Locally used by each subroutine
;VF23:  Locally used by each subroutine
;VF24:  Locally used by each subroutine
;VF25:  Locally used by each subroutine
;VF26:  Locally used by each subroutine
;VF27:  Locally used by each subroutine
;VF28:  -        -        -       -
;VF29:  u_0      v_0      -       -  ; Texture coordinate temporary memory
;VF30:  Delta u  Delta v  -       -  ; Texture coordinate Delta
;VF31:  1/32     1/16     1/8     1/4
;
; Integer register
;VI00:  0 register
;VI01:  u counter (0 - 32)
;VI02:  v counter (0 - 32)
;VI03:  -
;VI04:  -
;VI05:  -
;VI06:  -
;VI07:  -
;VI08:  -
```

```
;VI09:   -
;VI10:   -
;VI11:   Base register for Buffer 0
;VI12:   Base register for Buffer 1
;VI13:   Buffer 0, 1 swap temporary, and texture output pointer
;VI14:   Stack pointer recommended
;VI15:   Link register
```

Curve generation

This program generates a Bezier curve. A Bezier curve represented by 16 control points is replaced with a plane polygon divided into 32 x 32 two-dimensional meshes and is output as 33 x 33 vertex columns.

```
;-------------------------------------------------------------------
; SURF subroutine temporary register
; Floating point register
;VF16:           ; BTv load temporary (BTu temp when optimized)
;VF17:           ; BTu load temporary
;VF18:           ; Output register
;VF19:           ; Control point temporary 1
;VF20:           ; Control point temporary 2
;VF21:           ; Control point temporary 3
;VF22:           ; Control point temporary 4
;VF23:           ; SUM (BTu X CP) temporary 1 tmp1
;VF24:           ; SUM (BTu X CP) temporary 2 tmp2
;VF25:           ; SUM (BTu X CP) temporary 3 tmp3
;VF26:           ; SUM (BTu X CP) temporary 4 tmp4
;VF27:           ; Output register (use when optimized)

;Integer register
;VI00:  0 register
;VI01:  u counter (0 - 32)
;VI02:  v counter (0 - 32)
;VI03:  i counter (0 - 3)
;VI04:  pBTu (pointer to u blending table)
;VI05:  pBTv (pointer to v blending table)
;VI06:  pCP (pointer to control point)
;VI07:  Output vertex column pointer (dummy is output right before the pointer when optimized)
;      :
;VI14:  Stack pointer recommended
;VI15:  Link register

; Curve generation subroutine (u one-column 33-vertex)
;--------------------------------------
; Non-optimized version (for understanding the algorithms)
;--------------------------------------
SURF:
   NOP                        LQI VF16, (VI05++) ; load BTv
   NOP                        LQI VF19, (VI06++) ; load CP
   NOP                        LQI VF20, (VI06++) ; load CP
   NOP                        LQI VF21, (VI06++) ; load CP
   NOP                        LQI VF22, (VI06++) ; load CP

   MULAx.xyzw ACC, VF19, VF16x     LQI VF19, (VI06++) ; tmp1=CP*BTv
   MADDAy.xyzw ACC, VF20, VF16y    LQI VF20, (VI06++) ;   +=CP*BTv
   MADDAz.xyzw ACC, VF21, VF16z    LQI VF21, (VI06++) ;   +=CP*BTv
   MADDw.xyzw VF23, VF22, VF16w    LQI VF22, (VI06++) ;   +=CP*BTv

   MULAx.xyzw ACC, VF19, VF16x     LQI VF19, (VI06++) ; tmp2=CP*BTv
   MADDAy.xyzw ACC, VF20, VF16y    LQI VF20, (VI06++) ;   +=CP*BTv
```

```
    MADDAz.xyzw ACC, VF21, VF16z    LQI VF21, (VI06++) ;   +=CP*BTv
    MADDw.xyzw VF24, VF22, VF16w    LQI VF22, (VI06++) ;   +=CP*BTv

    MULAx.xyzw ACC, VF19, VF16x     LQI VF19, (VI06++) ; tmp3=CP*BTv
    MADDAy.xyzw ACC, VF20, VF16y    LQI VF20, (VI06++) ;   +=CP*BTv
    MADDAz.xyzw ACC, VF21, VF16z    LQI VF21, (VI06++) ;   +=CP*BTv
    MADDw.xyzw VF25, VF22, VF16w    LQI VF22, (VI06++) ;   +=CP*BTv

    MULAx.xyzw ACC, VF19, VF16x     LQI VF19, (VI06++) ; tmp4=CP*BTv
    MADDAy.xyzw ACC, VF20, VF16y    LQI VF20, (VI06++) ;   +=CP*BTv
    MADDAz.xyzw ACC, VF21, VF16z    LQI VF21, (VI06++) ;   +=CP*BTv
    MADDw.xyzw VF26, VF22, VF16w    LQI VF22, (VI06++) ;   +=CP*BTv

    NOP                         IADDI VI01, VI00, 33
Lu:
    NOP                         LQI VF17, (VI04++) ; load BTu
    NOP                         NOP
    NOP                         NOP
    NOP                         NOP
    MULAx.xyzw ACC, VF23, VF17x     NOP             ;  = tmp1*BTu
    MADDAy.xyzw ACC, VF24, VF17y    NOP             ; += tmp2*BTu
    MADDAz.xyzw ACC, VF25, VF17z    NOP             ; += tmp3*BTu
    MADDw.xyzw VF18, VF26, VF17w    NOP             ; += tmp4*BTu
    NOP                         NOP
    NOP                         NOP
    NOP                         IADDI VI01, VI01, -1
    NOP                         SQI VF18, (VI07++)   ; store Data
    NOP                         NOP
    NOP                         IBNE VI01, VI00, Lu:  ; loop
    NOP                         NOP         ; BDSlot

    NOP                         JR (VI15)       ; return
    NOP                         NOP         ; BDSlot


;--------------------------------------
; Optimized version
;--------------------------------------
;  Lu: Loop unrolls twice and makes the load delay no longer visible.
;      Therefore, the looping count is not 33 loops, but is instead 16 loops + epilog (for 1 loop).
SURF_O:
    NOP                         LQI VF16, (VI05++) ; load BTv
    NOP                         LQI VF19, (VI06++) ; load CP
    NOP                         LQI VF20, (VI06++) ; load CP
    NOP                         LQI VF21, (VI06++) ; load CP
    NOP                         LQI VF22, (VI06++) ; load CP

    MULAx.xyzw ACC, VF19, VF16x     LQI VF19, (VI06++) ; tmp1=CP*BTv
    MADDAy.xyzw ACC, VF20, VF16y    LQI VF20, (VI06++) ;   +=CP*BTv
    MADDAz.xyzw ACC, VF21, VF16z    LQI VF21, (VI06++) ;   +=CP*BTv
    MADDw.xyzw VF23, VF22, VF16w    LQI VF22, (VI06++) ;   +=CP*BTv

    MULAx.xyzw ACC, VF19, VF16x     LQI VF19, (VI06++) ; tmp2=CP*BTv
    MADDAy.xyzw ACC, VF20, VF16y    LQI VF20, (VI06++) ;   +=CP*BTv
    MADDAz.xyzw ACC, VF21, VF16z    LQI VF21, (VI06++) ;   +=CP*BTv
    MADDw.xyzw VF24, VF22, VF16w    LQI VF22, (VI06++) ;   +=CP*BTv

    MULAx.xyzw ACC, VF19, VF16x     LQI VF19, (VI06++) ; tmp3=CP*BTv
    MADDAy.xyzw ACC, VF20, VF16y    LQI VF20, (VI06++) ;   +=CP*BTv
```

```
    MADDAz.xyzw ACC, VF21, VF16z    LQI VF21, (VI06++) ;   +=CP*BTv
    MADDw.xyzw VF25, VF22, VF16w    LQI VF22, (VI06++) ;   +=CP*BTv

    MULAx.xyzw ACC, VF19, VF16x     LQI VF17, (VI04++) ; tmp4.. ,ld BTu
    MADDAy.xyzw ACC, VF20, VF16y    NOP                ;   +=CP*BTv
    MADDAz.xyzw ACC, VF21, VF16z    IADDI VI07,VI07,-1 ;   +=CP*BTv
    MADDw.xyzw VF26, VF22, VF16w    IADDI VI01,VI00,16 ;   +=CP*BTv

Lu:
    MULAx.xyzw ACC, VF23, VF17x     LQI VF16, (VI04++) ;  = tmp1*BTu
    MADDAy.xyzw ACC, VF24, VF17y    NOP                ; += tmp2*BTu
    MADDAz.xyzw ACC, VF25, VF17z    IADDI VI01,VI01,-1 ; += tmp3*BTu
    MADDw.xyzw VF18, VF26, VF17w    SQI VF27, (VI07++) ; += tmp4*BTu
    MULAx.xyzw ACC, VF23, VF16x     LQI VF17, (VI04++) ;  = tmp1*BTu
    MADDAy.xyzw ACC, VF24, VF16y    NOP                ; += tmp2*BTu
    MADDAz.xyzw ACC, VF25, VF16z    IBNE VI01, VI00, Lu: ; loop
    MADDw.xyzw VF27, VF26, VF16w    SQI VF18, (VI07++) ; BDSlot

    MULAx.xyzw ACC, VF23, VF17x     NOP                ;  = tmp1*BTu
    MADDAy.xyzw ACC, VF24, VF17y    NOP                ; += tmp2*BTu
    MADDAz.xyzw ACC, VF25, VF17z    NOP                ; += tmp3*BTu
    MADDw.xyzw VF18, VF26, VF17w    SQI VF27, (VI07++) ; += tmp4*BTu

    NOP*                            NOP*
    NOP*                            NOP*
    NOP                             JR (VI15)          ; return
    NOP                             SQI VF28, (VI07++) ;  last store
```

Perspective conversion

This program performs perspective conversion to each vertex of the mesh.

```
;--------------------------------------------------------------------
; PRSP subroutine temporary register
; Floating point register
;VF16:                        ; Input vertex
;VF17:                        ; Vertex after coordinate conversion
;VF18:                        ; Output vertex
;VF19:                        ; Temporary for optimized version

; Integer register
;VI00:  0 register
;VI01:  u counter (0 - 32)
;VI02:  v counter (0 - 32)
;VI03:  i counter (0 - 3)
;VI04:  Input vertex pointer
;VI05:  Output vertex pointer (Two dummies are written before this pointer in optimized version)
;     :
;VI14:  Stack pointer recommended
;VI15:  Link register

; Coordinate conversion and perspective conversion (for 33 vertices)
;-------------------------------------
; Non-optimized version (for understanding the algorithms)
;-------------------------------------
PRSP:
    MULw.w VF23w, VF01w, VF01w          IADDI VI01, VI00, 33 ; VF23w=1
Lp: NOP                                 LQI VF16, (VI04++)   ; load V
    NOP                                 NOP
    NOP                                 NOP
```

```
    NOP                         NOP
    MULAx.xyzw ACC, VF10, VF16x         NOP
    MADDAy.xyzw ACC, VF11, VF16y        NOP
    MADDAz.xyzw ACC, VF12, VF16z        NOP
    MADDw.xyzw VF17, VF13, VF16w        NOP
    NOP                         NOP
    NOP                         NOP
    NOP                         NOP
    NOP                         DIV Q, VF00w, VF17w
    NOP                         NOP
    NOP                         NOP
    NOP                         NOP
    NOP                         NOP
    NOP                         NOP
    NOP                         NOP
    MULq.xyz VF18, VF17, Q              NOP
    NOP                         NOP
    NOP                         NOP
    NOP                         IADDI VI01, VI01, -1
    NOP                         SQI VF18, (VI05++)
    NOP                         NOP
    NOP                         IBNE VI01, VI00, Lp:
    NOP                         NOP           ; BDSlot
    NOP                         JR (VI15)       ; return
    NOP                         NOP           ; BDSlot

;--------------------------------------
; Optimized version
;--------------------------------------
PRSP_O:
    MULw.w VF23w, VF01w, VF01w          IADDI VI01, VI00, 33 ; VF23w=1
    NOP                         LQI VF16, (VI04++)   ; load V
    NOP                         IADDI VI05, VI05, -2
    NOP                         NOP

Lp: MULq.xyz VF18, VF19, Q              DIV Q, VF00w, VF17w
    MULAx.xyzw ACC, VF10, VF16x         MOVE.xyzw VF19, VF17
    MADDAy.xyzw ACC, VF11, VF16y        NOP
    MADDAz.xyzw ACC, VF12, VF16z        IADDI VI01, VI01, -1
    MADDw.xyzw VF17, VF13, VF16w        LQI VF16, (VI04++)   ; load V
    NOP                         IBNE VI01, VI00, Lp:
    NOP                         SQI VF18, (VI05++)   ; BDSlot

    MULq.xyz VF18, VF19, Q              DIV Q, VF00w, VF17w
    NOP                         MOVE.xyzw VF19, VF17
    NOP*                        NOP*
    NOP*                        NOP*
    NOP                         SQI VF18, (VI05++)
    NOP*                        NOP*
    NOP                         WAITQ
    MULq.xyz VF18, VF19, Q              NOP
    NOP*                        NOP*
    NOP*                        NOP*
    NOP                         JR (VI15)       ; return
    NOP                         SQI VF18, (VI05++) ; BDSlot
```

Normalization

This program obtains the normal for each vertex according to the vector product of the vector to two adjoining vertices, and normalizes it so that the length becomes 1.

```
;-------------------------------------------------------------------
; NORM subroutine temporary register
; Floating point register
;VF16:                    ; Input vertex 1
;VF17:                    ; Input vertex 2
;VF18:                    ; Input vertex 3
;VF19:                    ; Input vector 1
;VF20:                    ; Input vector 2
;VF21:                    ; Vector product results
;VF22:                    ; Vector product calculation temporary
;VF23:                    ; Output normal
;VF24:                    ; Temporary during optimization
;VF25:                    ; Temporary during optimization
;VF26:                    ; Temporary during optimization

; Integer register
;VI00:  0 register
;VI01:  u counter (0 - 32)
;VI02:  v counter (0 - 32)
;VI03:  i counter (0 - 3)
;VI04:  Input vertex pointer (previous column)
;VI05:  Input vertex pointer (current column)
;VI06:  Output normal pointer (Two dummies are written before this pointer in optimized version)
;    :
;VI14:  Stack pointer recommended
;VI15:  Link register

; Normal calculation (for 33 vertices)
;-------------------------------------
; Non-optimized version (for understanding the algorithms)
;-------------------------------------
NORM:
   MULw.w VF23w, VF01w, VF01w            IADDI VI01, VI00, 33 ; VF23w=1
   NOP                             IADDI VI01, VI01, -1
Ln: NOP                            LQ VF17, (VI04 + 1)
   NOP                             LQI VF16, (VI04++)
   NOP                             LQI VF18, (VI05++)
   NOP                             NOP
   NOP                             NOP
   SUB.xyz VF19xyz, VF17xyz, VF16xyz       NOP
   SUB.xyz VF20xyz, VF18xyz, VF16xyz       NOP
   NOP                             NOP
   NOP                             NOP
   NOP                             NOP
   OPMULA.xyz ACCxyz,VF19xyz,VF20xyz       NOP
   OPMSUB.xyz VF21xyz,VF19xyz,VF20xyz      NOP
   NOP                             NOP
   NOP                             NOP
   NOP                             NOP
   MUL.xyz VF22xyz, VF21xyz, VF21xyz       NOP
   NOP                             NOP
   NOP                             NOP
   NOP                             NOP
   ADDy.x VF22x, VF22x, VF22y            NOP
   NOP                             NOP
```

```
NOP                          NOP
NOP                          NOP
ADDz.x VF22x, VF22x, VF22z   NOP
NOP                          NOP
NOP                          NOP
NOP                          NOP
NOP                          RSQRT Q, VF00w, VF22x
NOP                          NOP
NOP                          NOP
NOP                          NOP
NOP                          NOP
NOP                          NOP
NOP                          NOP
NOP                          NOP
NOP                          NOP
NOP                          NOP
NOP                          NOP
NOP                          NOP
NOP                          NOP
MULq.xyz VF23xyz, VF21xyz, Q NOP
NOP                          NOP
NOP                          NOP
NOP                          IADDI VI01, VI01, -1
NOP                          SQI VF23, (VI06++)
NOP                          NOP
NOP                          IBNE VI01, VI00, Ln:
NOP                          NOP         ; BDSlot
NOP                          JR (VI15)   ; return
NOP                          NOP         ; BDSlot
; Vertex that obtains the differential varies in the final normal.
NOP                          LQ VF17, (VI04 - 1)  ;***
NOP                          LQI VF16, (VI04++)
NOP                          LQI VF18, (VI05++)
NOP                          NOP
NOP                          NOP
SUB.xyz VF19xyz, VF17xyz, VF16xyz   NOP
SUB.xyz VF20xyz, VF18xyz, VF16xyz   NOP
NOP                          NOP
NOP                          NOP
NOP                          NOP
OPMULA.xyz ACCxyz,VF20xyz,VF19xyz   NOP   ;***
OPMSUB.xyz VF21xyz,VF20xyz,VF19xyz  NOP   ;***
NOP                          NOP
NOP                          NOP
NOP                          NOP
MUL.xyz VF22xyz, VF21xyz, VF21xyz   NOP
NOP                          NOP
NOP                          NOP
NOP                          NOP
ADDy.x VF22x, VF22x, VF22y   NOP
NOP                          NOP
NOP                          NOP
NOP                          NOP
ADDz.x VF22x, VF22x, VF22z   NOP
NOP                          NOP
NOP                          NOP
NOP                          NOP
NOP                          RSQRT Q, VF00w, VF22x
NOP                          NOP
```

```
    NOP                 NOP
    NOP                 NOP
    NOP                 NOP
    NOP                 NOP
    NOP                 NOP
    NOP                 NOP
    NOP                 NOP
    NOP                 NOP
    NOP                 NOP
    NOP                 NOP
    NOP                 NOP
    MULq.xyz VF23xyz, VF21xyz, Q        NOP
    NOP                 NOP
    NOP                 NOP
    NOP                 IADDI VI01, VI01, -1
    NOP                 SQI VF23, (VI06++)
    NOP                 NOP
    NOP                 IBNE VI01, VI00, Ln:
    NOP                 NOP         ; BDSlot
    NOP                 JR (VI15)       ; return
    NOP                 NOP         ; BDSlot

;--------------------------------------
; Optimized version (also performs texture coordinate calculations)
;--------------------------------------
NORM_TEXD:
MULw.w VF23w, VF01w, VF01w      IADDI VI01, VI00,(33-2); VF23w=1
    NOP                 LQ.xyzw VF17, (VI04 + 1)
    NOP                 LQI.xyzw VF16, (VI04++)
    NOP                 LQI.xyzw VF18, (VI05++)
    NOP                 IADDI VI05, VI05, -3
    NOP                 SQI XY, VF29, VI13

Ln: MULq.xyz VF23xyz, VF24xyz, Q        RSQRT Q, VF00w, VF25x
    SUB.xyz VF19xyz, VF17xyz, VF16xyz   MOVE.xyzw VF24, VF26
    SUB.xyz VF20xyz, VF18xyz, VF16xyz   MOVE.xyzw VF26, VF21
    ADDy.x VF22x, VF22x, VF22y          IADDI VI01, VI01, -1
    ADD.x VF29x, VF29x, VF30x           SQI.xyzw VF23, (VI06++)   ; TEXD
    NOP                 NOP
    OPMULA.xyz ACCxyz,VF19xyz,VF20xyz   LQ.xyzw VF17, (VI04 + 1)
    OPMSUB.xyz VF21xyz,VF19xyz,VF20xyz  LQI.xyzw VF16, (VI04++)
    ADDz.x VF25x, VF22x, VF22z          SQI.xy VF29xy, (VI13++)   ; TEXD
    NOP                 LQI.xyzw VF18, (VI05++)
    NOP                 NOP
    MUL.xyz VF22xyz, VF21xyz, VF21xyz   IBNE VI01, VI00, Ln:
    NOP                 WAITQ       ; BDSlot

; This is a gigantic epilog.  The vertex that obtains the differential varies in the final normal.
; Expels 29 vertices, Q of 30 vertices, normalize of 31 vertices
; Vector product of 32 vertices, texture coordinates of 33 vertices
    MULq.xyz VF23xyz, VF24xyz, Q        RSQRT Q, VF00w, VF25x
    SUB.xyz VF19xyz, VF17xyz, VF16xyz   MOVE.xyzw VF24, VF26
    SUB.xyz VF20xyz, VF18xyz, VF16xyz   MOVE.xyzw VF26, VF21
    ADDy.x VF22x, VF22x, VF22y          NOP
    ADD.x VF29x, VF29x, VF30x           SQI.xyzw VF23, (VI06++)   ; TEXD
    NOP*                NOP*
    OPMULA.xyz ACCxyz,VF19xyz,VF20xyz   LQ.xyzw VF17, (VI04 - 1)
    OPMSUB.xyz VF21xyz,VF19xyz,VF20xyz  LQI.xyzw VF16, (VI04++)
    ADDz.x VF25x, VF22x, VF22z          SQI.xy VF29xy, (VI13++)   ; TEXD
```

```
    NOP                          LQI.xyzw VF18, (VI05++)
    NOP*                         NOP*
    MUL.xyz VF22xyz, VF21xyz, VF21xyz  IADDI VI01, VI01, -1
    NOP                          WAITQ

; Expels 30 vertices, Q of 31 vertices, normalize of 32 vertices
; Vector product of 33 vertices
    MULq.xyz VF23xyz, VF24xyz, Q        RSQRT Q, VF00w, VF25x
    SUB.xyz VF19xyz, VF17xyz, VF16xyz   MOVE.xyzw VF24, VF26
    SUB.xyz VF20xyz, VF18xyz, VF16xyz   MOVE.xyzw VF26, VF21
    ADDy.x VF22x, VF22x, VF22y          NOP
    ADD.x VF29x, VF29x, VF30x           SQI.xyzw VF23, (VI06++)    ; TEXD
    NOP*                         NOP*
    OPMULA.xyz ACCxyz,VF19xyz,VF20xyz   NOP
    OPMSUB.xyz VF21xyz,VF19xyz,VF20xyz  NOP
    ADDz.x VF25x, VF22x, VF22z          NOP
    NOP                          NOP
    NOP*                         NOP*
    NOP*                         NOP*
    MUL.xyz VF22xyz, VF21xyz, VF21xyz   NOP
    NOP                          WAITQ

; Expels 31 vertices, Q of 32 vertices, normalize of 33 vertices
    MULq.xyz VF23xyz, VF24xyz, Q        RSQRT Q, VF00w, VF25x
    NOP                          MOVE.xyzw VF24, VF26
    ADDy.x VF22x, VF22x, VF22y          MOVE.xyzw VF26, VF21
    NOP*                         NOP*
    NOP                          SQI.xyzw VF23, (VI06++)
    NOP*                         NOP*
    ADDz.x VF25x, VF22x, VF22z          NOP
    NOP*                         NOP*
    NOP*                         NOP*
    NOP*                         NOP*
    NOP*                         NOP*
    NOP*                         NOP*
    NOP                          WAITQ

; Expels 32 vertices, Q of 33 vertices
    MULq.xyz VF23xyz, VF24xyz, Q        RSQRT Q, VF00w, VF25x
    NOP                          MOVE.xyzw VF24, VF26
    NOP*                         NOP*
    NOP*                         NOP*
    NOP                          SQI.xyzw VF23, (VI06++)
    NOP*                         NOP*
    NOP*                         NOP*
    NOP*                         NOP*
    NOP*                         NOP*
    NOP*                         NOP*
    NOP*                         NOP*
    NOP*                         NOP*
    NOP                          WAIT

; Expels 33 vertices
    MULq.xyz VF23xyz, VF24xyz, Q        NOP
    NOP*                         NOP*
    NOP*                         NOP*
    NOP                          JR (VI15)       ; return
    NOP                          SQI.xyzw VF23, (VI06++) ; BDSlot
```

Parallel light source

This program obtains the color of each vertex from the light source color, the direction vector of the light source, and the normal.

```
;--------------------------------------------------------------------
; LIGHT subroutine temporary register
; Floating point register
;VF16:                          ; Input normal
;VF17:                          ; cos (theta) 0, cos (theta) 1, cos (theta) 2, cos (theta) 3
;VF18:                          ; cos (theta)^ns temporary
;VF19:                          ; Output R, G, B

; Integer register
;VI00:  0 register
;VI01:  u counter (0 - 32)
;VI02:  v counter (0 - 32)
;VI03:  i counter (0 - 3)
;VI04:  Input normal pointer
;VI05:  Output RGB pointer
;VI06:  NS (=0, 1, 2, 4, ....)
;     :
;VI13 :
;VI14:  Stack pointer recommended
;VI15:  Link register

; Illumination processing of parallel light source (for 33 vertices)
;--------------------------------------
; Non-optimized version (for understanding algorithms)
;--------------------------------------
LIGHT:
   MULw.w VF23w, VF01w, VF01w          IADDI VI01, VI00, 33 ; VF23w=1
Ll: NOP                         LQI VF16, (VI04++)
   NOP                          IADDI VI06, VI00, NS
   NOP                          NOP
   NOP                          NOP
   MULAx.xyzw ACC, VF02, VF16x         NOP     ; lighting
   MADDAy.xyzw ACC, VF03, VF16y        NOP     ; calculate cos (theta)
   MADDz.xyzw VF17, VF04, VF16z        NOP
   NOP                          NOP
   NOP                          NOP
   NOP                          NOP
   MAXx.xyzw VF17, VF17, VF14x         NOP     ; clipping MINI=0
   NOP                          NOP
   NOP                          NOP
   MUL.xyzw VF18, VF01, VF01           NOP     ; VF18=(1,1,1,1)
Lns:
   MUL.xyzw VF18, VF18, VF17           IADDI VI06, VI06, -1     ; cos(theta)^(1,2,4,8,...)
   NOP                          NOP
   NOP                          IBNE VI06, VI00, Lns:
   NOP                          NOP     ; BDSlot

   MULAy.xyzw ACC, VF09, VF00w         NOP     ; Ra, Ga, Ba
   MADDAx.xyzw ACC, VF05, VF18x        NOP     ; R0,G0,B0*cos(theta)^NS
   MADDAy.xyzw ACC, VF06, VF18y        NOP     ; R1,G1,B1*cos(theta)^NS
   MADDAz.xyzw ACC, VF07, VF18z        NOP     ; R2,G2,B2*cos(theta)^NS
   MADDw.xyzw VF19, VF08, VF18w        NOP     ; R3,G3,B3*cos(theta)^NS
   NOP                          NOP
   NOP                          NOP
   NOP                          NOP
```

```
    MINIy.xyzw VF19, VF19, VF14y        NOP     ; clipping MAX=255
    NOP                       NOP
    NOP                       NOP
    NOP                       IADDI VI01, VI01, -1
    NOP                       SQI VF19, (VI05++)
    NOP                       NOP
    NOP                       IBNE VI01, VI00, Ll:
    NOP                       NOP               ; BDSlot
    NOP                       JR (VI15)         ; return
    NOP                       NOP               ; BDSlot

;--------------------------------------
; Optimized version
;--------------------------------------
LIGHT_O:
    MULw.w VF23w, VF01w, VF01w         IADDI VI01, VI00, 33 ; VF23w=1
    NOP                       IADDI VI05  VI05  -1
Ll:
    MULAy.xyzw ACC, VF09, VF00w        LQI VF16, (VI04++) ; Ra, Ga, Ba
    MADDAx.xyzw ACC, VF05, VF18x       NOP     ; R0,G0,B0*cos(theta)^NS
    MADDAy.xyzw ACC, VF06, VF18y       NOP     ; R1,G1,B1*cos(theta)^NS
    MADDAz.xyzw ACC, VF07, VF18z       NOP     ; R2,G2,B2*cos(theta)^NS
    MADDw.xyzw VF19, VF08, VF18w       NOP     ; R3,G3,B3*cos(theta)^NS

    MULAx.xyzw ACC, VF02, VF16x        NOP     ; lighting
    MADDAy.xyzw ACC, VF03, VF16y       NOP     ; calculate cos (theta)
    MADDz.xyzw VF17, VF04, VF16z       NOP

    MINIy.xyzw VF19, VF19, VF14y       NOP     ; clipping MAX=255
    NOP*                      NOP*
    NOP*                      NOP*
    MAXx.xyzw VF17, VF17, VF14x        NOP     ; clipping MINI=0
    NOP                       SQI VF19, (VI05++)
    NOP*                      NOP*
    MUL.xyzw VF18, VF01, VF01          IADDI VI01, VI01, -1 ; VF18=(1,1,1,1)
Lns:
    MUL.xyzw VF18, VF18, VF17          IADDI VI06, VI06, -1     ; cos(theta)^(1,2,4,8,...)
    NOP                       NOP
    NOP                       IBNE VI06, VI00, Lns:
    NOP                       NOP     ; BDSlot

    NOP                       IBNE VI01, VI00, Ll:
    NOP                       NOP               ; BDSlot

    MULAy.xyzw ACC, VF09, VF00w        NOP     ; Ra, Ga, Ba
    MADDAx.xyzw ACC, VF05, VF18x       NOP     ; R0,G0,B0*cos (theta)^NS
    MADDAy.xyzw ACC, VF06, VF18y       NOP     ; R1,G1,B1*cos (theta)^NS
    MADDAz.xyzw ACC, VF07, VF18z       NOP     ; R2,G2,B2*cos (theta)^NS
    MADDw.xyzw VF19, VF08, VF18w       NOP     ; R3,G3,B3*cos (theta)^NS
    NOP*                      NOP*
    NOP*                      NOP*
    NOP*                      NOP*
    MINIy.xyzw VF19, VF19, VF14y       NOP     ; clipping MAX=255
    NOP*                      NOP*
    NOP*                      NOP*
    NOP                       JR (VI15)         ; return
    NOP                       SQI VF19, (VI05++)      ; BDSlot
```

Point light source

This program obtains the color of each vertex from the light source color, light source position, and the normal.

```
;--------------------------------------------------------------------
; LIGHTP subroutine temporary register

In the case of point light source, there are coordinates for four point light sources:
VF02:  L0x   L0y   L0z   -   ; Light source coordinate 0
VF03:  L1x   L1y   L1z   -   ; Light source coordinate 1
VF04:  L2x   L2y   L2z   -   ; Light source coordinate 2
VF15:  L3x   L3y   L3z   -   ; Light source coordinate 3

; Floating point register
;VF16:              ; Input normal
;VF17:              ; cos (theta) 0, cos (theta) 1, cos (theta) 2, cos (theta) 3
;VF18:              ; cos(theta)^ns temporary
;VF19:              ; Output R, G, B
;VF20:              ; Input vetex coordinates
;VF21:              ; Light source vector temporary
;VF22:              ; Light source vector temporary
;VF23:              ; Light source vector temporary
;VF24:              ; Light source vector temporary
;VF25:              ; Light source vector temporary 2
;VF26:              ; Light source vector temporary 2
;VF27:              ; Light source vector temporary 2
;VF28:              ; Light source vector temporary 2
;VF29:              ; 1/ri

; Integer register
;VI00:  0 register
;VI01:  u counter (0 - 32)
;VI02:  v counter (0 - 32)
;VI03:  i counter (0 - 3)
;VI04:  Input normal pointer
;VI05:  Output RGB pointer
;VI06:  NS (=0, 1, 2, 4, ....)
;VI07:  Input vertex pointer
;    :
;VI13:
;VI14:  Stack pointer recommended
;VI15:  Link register

; Illumination processing of point light source (for 33 vertices)
;--------------------------------------
; Non-optimized version (for understanding algorithms)
;--------------------------------------
LIGHTP:
MULw.w VF23w, VF01w, VF01w        IADDI VI01, VI00, 33 ; VF23w=1
Ll:
   NOP                        LQI VF20, (VI07++)
   NOP                        IADDI, VI06, VI00, NS
   NOP                        NOP
   NOP                        NOP
   SUB.xyzw VF21, VF02, VF20        NOP ; L0: calc lx, ly, lz
   SUB.xyzw VF22, VF03, VF20        NOP ; L1: calc lx, ly, lz
   SUB.xyzw VF23, VF04, VF20        NOP ; L2: calc lx, ly, lz
   SUB.xyzw VF24, VF15, VF20        NOP ; L3: calc lx, ly, lz
   MUL.xyzw VF25, VF21, VF21        NOP ; L0: lx^2, ly^2, lz^2
   MUL.xyzw VF26, VF22, VF22        NOP ; L1: lx^2, ly^2, lz^2
```

```
MUL.xyzw VF27, VF23, VF23         NOP ; L2: lx^2, ly^2, lz^2
MUL.xyzw VF28, VF24, VF24         NOP ; L3: lx^2, ly^2, lz^2

MULAx.x ACCx, VF21x, VF21x        NOP ; L0: lx^2
MADDAy.x ACCx, VF01x, VF25y       NOP ; L0: lx^2+ly^2
MADDz.x VF25x, VF01x, VF25z       NOP ; L0: lx^2+ly^2+lz^2

MULAx.x ACCx, VF22x, VF22x        NOP ; L1: lx^2
MADDAy.x ACCx, VF01x, VF26y       NOP ; L1: lx^2+ly^2
MADDz.x VF26x, VF01x, VF26z       NOP ; L1: lx^2+ly^2+lz^2

MULAx.x ACCx, VF23x, VF23x        NOP ; L2: lx^2
MADDAy.x ACCx, VF01x, VF27y       NOP ; L2: lx^2+ly^2
MADDz.x VF27x, VF01x, VF27z       NOP ; L2: lx^2+ly^2+lz^2

MULAx.x ACCx, VF24x, VF24x        NOP ; L3: lx^2
MADDAy.x ACCx, VF01x, VF28y       NOP ; L3: lx^2+ly^2
MADDz.x VF28x, VF01x, VF28z       NOP ; L3: lx^2+ly^2+lz^2

NOP                    RSQRT Q, VF25x  ; L0
NOP                    NOP
NOP                    NOP
NOP                    NOP
NOP                    NOP
NOP                    NOP
NOP                    NOP
NOP                    NOP
NOP                    NOP
NOP                    NOP
NOP                    NOP
NOP                    NOP
NOP                    NOP
MULq.x VF29x, VF01x, Q            NOP ; L0: move Q
NOP                    NOP
NOP                    NOP
NOP                    NOP
MULz.xyz VF23xyz, VF23xyz, VF29z    NOP ; L2: normalize
NOP                    NOP
NOP                    NOP
NOP                    NOP
MULx.z VF25z, VF01z, VF23x        NOP ; L2: transpose
MULy.z VF26z, VF01z, VF23y        NOP ; L2: transpose
MULz.z VF27z, VF01z, VF23z        NOP ; L2: transpose
NOP                    NOP
NOP                    NOP

NOP                    RSQRT Q, VF26x  ; L1
NOP                    NOP
NOP                    NOP
NOP                    NOP
NOP                    NOP
NOP                    NOP
NOP                    NOP
NOP                    NOP
NOP                    NOP
NOP                    NOP
NOP                    NOP
NOP                    NOP
NOP                    NOP
```

```
MULq.y VF29y, VF01y, Q            NOP ; L1: move Q
NOP                               NOP
NOP                               NOP
NOP                               NOP
MULy.xyz VF22xyz, VF22xyz, VF29y  NOP ; L1: normalize
NOP                               NOP
NOP                               NOP
NOP                               NOP
MULx.y VF25y, VF01y, VF22x        NOP ; L1: transpose
MULy.y VF26y, VF01y, VF22y        NOP ; L1: transpose
MULz.y VF27y, VF01y, VF22z        NOP ; L1: transpose
NOP                               NOP
NOP                               NOP

NOP                               RSQRT Q, VF27x  ; L2
NOP                               NOP
NOP                               NOP
NOP                               NOP
NOP                               NOP
NOP                               NOP
NOP                               NOP
NOP                               NOP
NOP                               NOP
NOP                               NOP
NOP                               NOP
NOP                               NOP
NOP                               NOP
MULq.z VF29z, VF01z, Q            NOP ; L2: move Q
NOP                               NOP
NOP                               NOP
NOP                               NOP
MULx.xyz VF21xyz, VF21xyz, VF29x  NOP ; L0: normalize
NOP                               NOP
NOP                               NOP
NOP                               NOP
MULx.x VF25x, VF01x, VF21x        NOP ; L0: transpose
MULy.x VF26x, VF01x, VF21y        NOP ; L0: transpose
MULz.x VF27x, VF01x, VF21z        NOP ; L0: transpose
NOP                               NOP
NOP                               NOP

NOP                               RSQRT Q, VF28x  ; L3
NOP                               NOP
NOP                               NOP
NOP                               NOP
NOP                               NOP
NOP                               NOP
NOP                               NOP
NOP                               NOP
NOP                               NOP
NOP                               NOP
NOP                               NOP
NOP                               NOP
NOP                               NOP
MULq.w VF29w, VF01w, Q            NOP ; L3: move Q
NOP                               NOP
NOP                               NOP
NOP                               NOP
MULw.xyz VF24xyz, VF24xyz, VF29w  NOP ; L3: normalize
```

```
    NOP                     NOP
    NOP                     NOP
    NOP                     NOP
    MULx.w VF25w, VF01w, VF24x       NOP ; L3: transpose
    MULy.w VF26w, VF01w, VF24y       NOP ; L3: transpose
    MULz.w VF27w, VF01w, VF24z       NOP ; L3: transpose

    MULw.xyzw VF29, VF29, VF09w      NOP ; 1.ri * r0
    NOP                     NOP
    NOP                     NOP
    NOP                     NOP
    MULw.xyzw VF29, VF29, VF29       NOP; (r0/ri)^2
    NOP                     NOP
    NOP                     NOP
    NOP                     NOP
    MINIy.xyzw VF29, VF29, VF00w     NOP; (r0/ri)^2>1 ? 1:  (r0/ri)^2
    NOP                     NOP
    NOP                     NOP
    NOP                     NOP

    MULx.xyzw VF21, VF05, VF29x      NOP; R0,G0,B0*(r0/ri)^2
    MULx.xyzw VF22, VF06, VF29y      NOP ;R1,G1,B1*(r0/ri)^2
    MULx.xyzw VF23, VF07, VF29z      NOP; R2,G2,B2*(r0/ri)^2
    MULx.xyzw VF24, VF08, VF29w      NOP; R3,G3,B3*(r0/ri)^2

    NOP                     LQI VF16, (VI04++)
    NOP                     NOP
    NOP                     NOP
    NOP                     NOP
    MULAx.xyzw ACC, VF25, VF16x      NOP    ; lighting
    MADDAy.xyzw ACC, VF26, VF16y     NOP    ; calculate cos(theta)
    MADDz.xyzw VF17, VF27, VF16z     NOP
    NOP                     NOP
    NOP                     NOP
    NOP                     NOP
    MAXx.xyzw VF17, VF17, VF14y      NOP    ; clipping MINI=0
    NOP                     NOP
    NOP                     NOP
    MUL.xyzw VF18, VF01, VF01        NOP    ; VF18=(1,1,1,1)
Lns:
    MUL.xyzw VF18, VF18, VF17        IADDI VI06, VI06, -1    ; cos(theta)^(1,2,4,8,...)
    NOP                     NOP
    NOP                     IBNE VI06, VI00, Lns:
    NOP                     NOP    ; BDSlot

    MULAy.xyzw ACC, VF09, VF00w      NOP    ; Ra, Ga, Ba
    MADDAx.xyzw ACC, VF21, VF18x     NOP    ; R0,G0,B0*cos(theta)^NS
    MADDAy.xyzw ACC, VF22, VF18y     NOP    ; R1,G1,B1*cos(theta)^NS
    MADDAz.xyzw ACC, VF23, VF18z     NOP    ; R2,G2,B2*cos(theta)^NS
    MADDw.xyzw VF19, VF24, VF18w     NOP    ; R3,G3,B3*cos(theta)^NS
    NOP                     NOP
    NOP                     NOP
    NOP                     NOP
    MINIy.xyzw VF19, VF19, VF14x     NOP    ; clipping MAX=255
    NOP                     NOP
    NOP                     NOP
    NOP                     IADDI VI01, VI01, -1
    NOP                     SQI VF19, (VI05++)
    NOP                     NOP
```

```
    NOP                       IBNE VI01, VI00, Ll:
    NOP                       NOP              ; BDSlot
    NOP                       JR (VI15)        ; return
    NOP                       NOP              ; BDSlot

;Illumination processing of point light source (for 33 vertices)
;----------------
;Optimized version
;----------------
LIGHTP_O:
    NOP                       LQI VF20, (VI07++)
    MULw.w VF23w, VF01w, VF01w        IADDI VI01, VI00, 33; VF23w=1
    NOP                       IADDI VI05, VI05, -1
    NOP                       NOP
Ll:
    SUB.xyzw VF21, VF02, VF20         IADDI VI06,VI00,NS; L0: calc lx,ly,lz
    SUB.xyzw VF22, VF03, VF20         NOP;  L1: calc lx, ly, lz
    SUB.xyzw VF23, VF04, VF20         NOP;  L2: calc lx, ly, lz
    SUB.xyzw VF24, VF15, VF20         NOP;  L3: calc lx, ly, lz
    MUL.xyzw VF25, VF21, VF21         NOP; L0: lx^2, ly^2, lz^2
    MULAx.x ACCx, VF21x, VF21x        NOP; L0: lx^2
    MADDAy.x ACCx, VF01x, VF25y       NOP; L0: lx^2+ly^2
    MADDz.x VF25x, VF01x, VF25z       NOP; L0: lx^2+ly^2+lz^2
    MUL.xyzw VF26, VF22, VF22         NOP;  L1: lx^2, ly^2, lz^2
    MUL.xyzw VF27, VF23, VF23         NOP;  L2: lx^2, ly^2, lz^2
    MUL.xyzw VF28, VF24, VF24         NOP;  L3: lx^2, ly^2, lz^2

    MULAx.x ACCx, VF22x, VF22x        RSQRT Q, VF25x;  L1: lx^2,L0: 1/square root
    MADDAy.x ACCx, VF01x, VF26y       NOP;  L1: lx^2+ly^2
    MADDz.x VF26x, VF01x, VF26z       NOP;  L1: lx^2+ly^2+lz^2

    MULAx.x ACCx, VF23x, VF23x        NOP;  L2: lx^2
    MADDAy.x ACCx, VF01x, VF27y       NOP;  L2: lx^2+ly^2
    MADDz.x VF27x, VF01x, VF27z       NOP;  L2: lx^2+ly^2+lz^2

    MULAx.x ACCx, VF24x, VF24x        NOP;  L3: lx^2
    MADDAy.x ACCx, VF01x, VF28y       NOP;  L3: lx^2+ly^2
    MADDz.x VF28x, VF01x, VF28z       NOP;  L3: lx^2+ly^2+lz^2

    MINIy.xyzw VF19, VF19, VF14x      NOP; result: clipping MAX=255
    NOP*                      NOP*
    NOP*                      NOP*
    NOP*                      NOP*

    MULq.x VF29x, VF01x, Q            RSQRT Q, VF26x; L0: moveQ,L1: 1/square root
    NOP                       SQI VF19, (VI05++);  store result***
    NOP*                      NOP*
    NOP*                      NOP*
    MULx.xyz VF21xyz, VF21xyz, VF29x    NOP; L0 normalize
    NOP*                      NOP*
    NOP*                      NOP*
    NOP*                      NOP*
    MULx.x VF25x, VF01x, VF21x        NOP; L0: transpose
    MULy.x VF26x, VF01x, VF21y        NOP; L0: transpose
    MULz.x VF27x, VF01x, VF21z        NOP; L0: transpose
    NOP*                      NOP*
    NOP*                      NOP*

    MULq.y VF29y, VF01y, Q            RSQRT Q, VF27x;  L1: moveQ,L2: 1/squre root
```

```
NOP*                    NOP*
NOP*                    NOP*
NOP*                    NOP*
MULy.xyz VF22xyz, VF22xyz, VF29y    NOP;  L1: normalize
NOP*                    NOP*
NOP*                    NOP*
NOP*                    NOP*
MULx.y VF25y, VF01y, VF22x          NOP;  L1: transpose
MULy.y VF26y, VF01y, VF22y          NOP;  L1: transpose
MULz.y VF27y, VF01y, VF22z          NOP;  L1: transpose
NOP*                    NOP*
NOP*                    NOP*

MULq.z VF29z, VF01z, Q              RSQRT Q, VF28x;  L2: moveQ,L3: 1/square root
NOP*                    NOP*
NOP*                    NOP*
NOP*                    NOP*
MULz.xyz VF23xyz, VF23xyz, VF29z    NOP;  L2: normalize
NOP*                    NOP*
NOP*                    NOP*
NOP*                    NOP*
MULx.z VF25z, VF01z, VF23x          NOP;  L2: transpose
MULy.z VF26z, VF01z, VF23y          NOP;  L2: transpose
MULz.z VF27z, VF01z, VF23z          NOP;  L2: transpose
NOP*                    NOP*
NOP*                    NOP*

MULq.w VF29w, VF01w, Q              NOP;  L3: move Q
NOP*                    NOP*
NOP*                    NOP*
NOP*                    NOP*
MULw.xyzw VF29, VF29, VF09w         NOP; 1/ri * r0
MULq.xyz VF24xyz, VF24xyz, Q        NOP;  L3: normalize
NOP*                    NOP*
NOP*                    NOP*
MULw.xyzw VF29, VF29, VF29          NOP; (r0/ri)^2
MULx.w VF25w, VF01w, VF24x          NOP;  L3: transpose
MULy.w VF26w, VF01w, VF24y          NOP;  L3: transpose
MULz.w VF27w, VF01w, VF24z          LQI VF16, (VI04++);  L3: transpose

MINIy.xyzw VF29, VF29, VF00y        NOP; (r0/ri)^2>1 ? 1: (r0/ri)^2
NOP*                    NOP*
NOP*                    NOP*
MUL.xyzw VF18, VF01, VF01           NOP    ; VF18=(1,1,1,1)

MULAx.xyzw ACC, VF25, VF16x         NOP    ; lighting
MADDAy.xyzw ACC, VF26, VF16y        NOP    ; calculate cos (theta)
MADDz.xyzw VF17, VF27, VF16z        NOP
MULx.xyzw VF21, VF05, VF29x         NOP; R0,G0,B0*(r0/ri)^2
MULx.xyzw VF22, VF06, VF29y         NOP; R1,G1,B1*(r0/ri)^2
MULx.xyzw VF23, VF07, VF29z         NOP; R2,G2,B2*(r0/ri)^2
MAXx.xyzw VF17, VF17, VF14y         NOP    ; clipping MINI=0
MULx.xyzw VF24, VF08, VF29w         NOP; R3,G3,B3*(r0/ri)^2
NOP                     LQI VF20, (VI07++)
NOP                     IADDI VI01,VI01,-1
Lns:
MUL.xyzw VF18, VF18, VF17           IADDI VI06,VI06,-1;
NOP                                 NOP
NOP                                 IBNE VI06,VI00,Lns;
```

```
MULAw.xyzw ACC, VF09, VF00w        NOP
MADDAx.xyzw ACC, VF21, VF18x       NOP
MADDAy.xyzw ACC, VF22, VF18y       NOP
MADDAz.xyzw ACC, VF23, VF18z       IBNE VI01,VI00,Ll;
MADDw.xyzw VF19, VF24, VF18w       NOP

NOP*                     NOP*
NOP*                     NOP*
NOP*                     NOP*
MINIy.xyzw VF19, VF19, VF14x       NOP    ; clipping MAX=255
NOP*                     NOP*
NOP*                     NOP*
NOP                      JR (VI15)            ; return
NOP                      SQI VF19, (VI05++)     ; BDSlot
```

Texture coordinates partitioning

This program obtains the coordinate values of the texture for each vertex.

```
;---------------------------------------------------------------------
;TEXD subroutine temporary register
;Floating point register
VF29:   u_0     v_0     -       -     ;  Texture coordinates temporary memory
VF30:   Delta u     Delta v     -       -; Texture coordinates Delta

;Integer register
;VI00:  0 register
;VI01:  u counter (0 - 32)
;VI02:  v counter (0 - 32)
;VI03:  i counter (0 - 3)
;     :
;VI13:  Output RGB pointer
;VI14:  Stack pointer recommended
;VI15:  Link register

;Texture coordinates partitioning (for 33 vertices)
;--------------------------------------
;Non-optimized version (for understanding algorithms)
;--------------------------------------
TEXD:
   NOP                     IADDI VI01, VI00, 33
Lt: ADD.x VF29x, VF29x, VF30x          NOP
   NOP                     NOP
   NOP                     NOP
   NOP                     IADDI VI01, VI01, -1
   NOP                     SQI.xy VF29xy, (VI13++)
   NOP                     NOP
   NOP                     IBNE VI01, VI00, Lt:
   NOP                     NOP              ; BDSlot
   ADD.y VF29y, VF29y, VF30y          LQ.x VF29x, (VI00, #pTEX)
   NOP                     JR (VI15)          ; return
   NOP                     NOP              ; BDSlot

;--------------------------------------
;Optimized version
;--------------------------------------
;Is positioned so there is no conflict with the register and is embedded inside the NORM routine.
```

```
;--------------------------------------------------------------------
;OUTP subroutine temporary register
;Floating point register
;VF16:  -
;VF17:  -
;VF18:  -
;VF19:  -
;VF20:  -
;VF21:  -
;VF22:  -
;VF23:  -
;VF24:          ; Input vertex coordinate 0
;VF25:          ; Input texture coordinate 0
;VF26:          ; Input RGB 0
;VF27:          ; Input vertex coordinate 1
;VF28:          ; Input texture coordinate 1
;VF29:          ; Input RGB 1

;Integer register
;VI00:  0 register
;VI01:  u counter (0 - 32)
;VI02:  v counter (0 - 32)
;VI03:  i counter (0 - 3)
;VI04:  Input vertex coordinate 0 pointer
;VI05:  Input vertex coordinate 1 pointer
;VI06:  Input texture coordinate 0 pointer
;VI07:  Input texture coordinate 1 pointer
;VI08:  Input RGB0 pointer
;VI09:  Input RGB1 pointer
;VI10:  Output pointer (output by strip triangle)
;VI11:  -
;VI12:  -
;VI13:  -
;VI14:  Stack pointer recommended
;VI15:  Link register

;Consecutive polygon/strip triangle output (33 vertices)
;--------------------------------------
;Non-optimized version (for understanding algorithms)
;--------------------------------------
OUTP:
   NOP              IADDI VI01, VI00, 33
Lo: NOP              IADDI VI01, VI01, -1
   NOP              LQI VF24, (VI04++)
   NOP              LQI VF25, (VI06++)
   NOP              LQI VF26, (VI08++)
   NOP              LQI VF27, (VI05++)
   NOP              LQI VF28, (VI07++)
   NOP              LQI VF29, (VI09++)
   NOP              SQI VF24, (VI10++)
   NOP              SQI VF25, (VI10++)
   NOP              SQI VF26, (VI10++)
   NOP              SQI VF27, (VI10++)
   NOP              SQI VF28, (VI10++)
   NOP              IBNE VI01, VI00, Lo:
   NOP              SQI VF29, (VI10++)    ; BDSlot
   NOP              JR (VI15)         ; return
   NOP              NOP               ; BDSlot
```

```
;--------------------------------------
;Optimized version
;--------------------------------------
; Same as above
```

Displacement Mapping

This program adds offsets, which vary from vertex to vertex, to the coordinate values of respective vertices. The arrangement of the above offset values is called displacement, and is given by input displacement table.

```
;------------------------------------------------------------------
;DISP subroutine temporary register
;Floating point register
;VF16:                  ; Input displacement data
;VF17:                  ; Input vertex coordinate 0
;VF18:                  ; Input normal vector 0
;VF19:                  ; Input vertex coordinate 1
;VF20:                  ; Input normal vector 1
;VF21:                  ; Output vertex coordinate 0
;VF22:                  ; Output vertex coordinate 1

;Integer register
;VI00:  0 register
;VI01:  u counter (0 - 32)
;VI02:  v counter (0 - 32)
;VI03:  i counter (0 - 3)
;VI04:  Input vertex coordinate pointer
;VI05:  Input normal pointer
;VI06:  Input displacement table pointer
;VI07:  Output vertex coordinate pointer
;     :
;VI13 :
;VI14:  Stack pointer recommended
;VI15:  Link register

;Table displacement mapping (33 vertices)
;  It is assumed that the displacement data is continuous in the x, y, z, w fields.
;  Eight loops are invoked by 32/4.  The final vertex is calculated outside the loop.
;--------------------------------------
;Non-optimized version (for understanding the algorithms)
;--------------------------------------
DISP:
   NOP                          IADDI VI01, VI00, 8
Ld: NOP                         LQI VF16, (VI06++)
   NOP                          LQI VF17, (VI04++)
   NOP                          LQI VF18, (VI05++)
   NOP                          NOP
   NOP                          NOP
   MULA.xyzw ACC, VF01, VF17            NOP
   MADDx.xyzw VF21, VF18, VF16x         NOP
   NOP                          NOP
   NOP                          NOP
   NOP                          NOP
   NOP                          SQI VF21, (VI07++)

   NOP                          LQI VF19, (VI04++)
   NOP                          LQI VF20, (VI05++)
   NOP                          NOP
   NOP                          NOP
   MULA.xyzw ACC, VF01, VF19            NOP
```

```
    MADDy.xyzw VF22, VF20, VF16y        NOP
    NOP                         NOP
    NOP                         NOP
    NOP                         NOP
    NOP                         SQI VF22, (VI07++)

    NOP                         LQI VF17, (VI04++)
    NOP                         LQI VF18, (VI05++)
    NOP                         NOP
    NOP                         NOP
    MULA.xyzw ACC, VF01, VF17           NOP
    MADDz.xyzw VF21, VF18, VF16z        NOP
    NOP                         NOP
    NOP                         NOP
    NOP                         NOP
    NOP                         SQI VF21, (VI07++)

    NOP                         LQI VF19, (VI04++)
    NOP                         LQI VF20, (VI05++)
    NOP                         NOP
    NOP                         NOP
    MULA.xyzw ACC, VF01, VF19           NOP
    MADDw.xyzw VF22, VF20, VF16w        NOP
    NOP                         NOP
    NOP                         NOP
    NOP                         IADDI VI01, VI01, -1
    NOP                         SQI VF22, (VI07++)

    NOP                         NOP
    NOP                         IBNE VI01, VI00, Ld:
    NOP                         NOP             ; BDSlot

    NOP                         LQI VF16, (VI06++)
    NOP                         LQI VF17, (VI04++)
    NOP                         LQI VF18, (VI05++)
    NOP                         NOP
    NOP                         NOP
    MULA.xyzw ACC, VF01, VF17           NOP
    MADDx.xyzw VF21, VF18, VF16x        NOP
    NOP                         NOP
    NOP                         NOP
    NOP                         NOP
    NOP                         SQI VF21, (VI07++)

    NOP                         JR (VI15)       ; return
    NOP                         NOP             ; BDSlot

;-------------------------------------
;Optimized version
;-------------------------------------
DISP_O:
    NOP                         IADDI VI01, VI00, 8
    NOP                         LQI VF16, (VI06++)

Ld: NOP                         LQI VF17, (VI04++)
    NOP                         LQI VF18, (VI05++)
    NOP                         LQI VF19, (VI04++)
    NOP                         LQI VF20, (VI05++)
    MULA.xyzw ACC, VF01, VF17           LQI VF17, (VI04++)
```

```
MADDx.xyzw VF21, VF18, VF16x        LQI VF18, (VI05++)
MULA.xyzw ACC, VF01, VF19           LQI VF19, (VI04++)
MADDy.xyzw VF22, VF20, VF16y        LQI VF20, (VI05++)
MULA.xyzw ACC, VF01, VF17           NOP
MADDz.xyzw VF21, VF18, VF16z        SQI VF21, (VI07++)
MULA.xyzw ACC, VF01, VF19           IADDI VI01, VI01, -1
MADDw.xyzw VF22, VF20, VF16w        SQI VF22, (VI07++)
NOP                                 LQI VF16, (VI06++)
NOP                                 SQI VF21, (VI07++)
NOP                                 IBNE VI01, VI00, Ld:
NOP                                 SQI VF22, (VI07++)  ; BDSlot

NOP                                 LQI VF17, (VI04++)
NOP                                 LQI VF18, (VI05++)
NOP*                                NOP*
NOP*                                NOP*
MULA.xyzw ACC, VF01, VF17           NOP
MADDx.xyzw VF21, VF18, VF16x        NOP
NOP*                                NOP*
NOP*                                NOP*
NOP                                 JR (VI15)           ; return
NOP                                 SQI VF21, (VI07++)  ; BDSlot
```

Random number displacement mapping

This subroutine obtains the displacement value from random numbers, not from the table.

```
;--------------------------------------------------------------------
;DISPR subroutine temporary register
;Floating point register
;VF16:                ; Input displacement data
;VF17:                ; Input vertex coordinate 0
;VF18:                ; Input normal vector 0
;VF19:                ; Input vertex coordinate 1
;VF20:                ; Input normal vector 1
;VF21:                ; Output vertex coordinate 0
;VF22:                ; Output vertex coordinate 1

;Integer register
;VI00:  0 register
;VI01:  u counter (0 - 32)
;VI02:  v counter (0 - 32)
;VI03:  i counter (0 - 3)
;VI04:  Input vertex coordinate pointer
;VI05:  Input normal pointer
;VI06:  Input displacement table pointer
;VI07:  Output vertex coordinate pointer
;     :
;VI13:
;VI14: Stack pointer recommended
;VI15: Link register

;Random number displacement mapping (33 vertices)
;  Random numbers are created by the RNEXT instruction within the range +1.0 to +2.0.
;  Scaling to 0 - MR by RNEXT(1 - 2) x MR - MR.
;  Eight loops are invoked by 32/4 and the final vertex is calculated outside the loop.

;Non-optimized version (for understanding algorithms)
;--------------------------------------
DISPR:
```

```
;--------------------------------------
    NOP                         IADDI VI01, VI00, 8

Lr:
    NOP                         RNEXT VF16x
    NOP                         NOP
    NOP                         NOP
    NOP                         NOP
    MULAw.x ACCx, VF16x, VF14w          LQI VF17, (VI04++); Rnad x MR
    MSUBw.x VF16x, VF01x, VF14w         LQI VF18, (VI05++); RNEXT x MR – MR
    NOP                         NOP
    NOP                         NOP
    MULA.xyzw ACC, VF01, VF17           NOP
    MADDx.xyzw VF21, VF18, VF16x        NOP
    NOP                         NOP
    NOP                         NOP
    NOP                         NOP
    NOP                         SQI VF21, (VI07++)

    NOP                         RNEXT VF16y
    NOP                         NOP
    NOP                         NOP
    NOP                         NOP
    MULAw.y ACCy, VF16y, VF14w          LQI VF19, (VI04++); Rnad x MR
    MSUBw.y VF16y, VF01y, VF14w         LQI VF20, (VI05++); RNEXT x MR – MR
    NOP                         NOP
    NOP                         NOP
    MULA.xyzw ACC, VF01, VF19           NOP
    MADDy.xyzw VF22, VF20, VF16y        NOP
    NOP                         NOP
    NOP                         NOP
    NOP                         NOP
    NOP                         SQI VF22, (VI07++)

    NOP                         RNEXT VF16z
    NOP                         NOP
    NOP                         NOP
    NOP                         NOP
    MULAw.z ACCz, VF16z, VF14w          LQI VF17, (VI04++); Rnad x MR
    MSUBw.z VF16z, VF01z, VF14w         LQI VF18, (VI05++); RNEXT x MR – MR
    NOP                         NOP
    NOP                         NOP
    MULA.xyzw ACC, VF01, VF17           NOP
    MADDz.xyzw VF21, VF18, VF16z        NOP
    NOP                         NOP
    NOP                         NOP
    NOP                         NOP
    NOP                         SQI VF21, (VI07++)

    NOP                         RNEXT VF16w
    NOP                         NOP
    NOP                         NOP
    NOP                         NOP
    MULAw.w ACCw, VF16w, VF14w      LQI VF19, (VI04++); Rnad x MR
    MSUBw.w VF16w, VF01w, VF14w     LQI VF20, (VI05++); RNEXT x MR – MR
    NOP                         NOP
    NOP                         NOP
    MULA.xyzw ACC, VF01, VF19           NOP
    MADDw.xyzw VF22, VF20, VF16w        NOP
```

```
    NOP                      NOP
    NOP                      NOP
    NOP                      IADDI VI01, VI01, -1
    NOP                      SQI VF22, (VI07++)

    NOP                      NOP
    NOP                      IBNE VI01, VI00, Lr:
    NOP                      NOP            ; BDSlot

    NOP                      RNEXT VF16x
    NOP                      NOP
    NOP                      NOP
    NOP                      NOP
    MULAw.x ACCx, VF16x, VF14w        LQI VF17, (VI04++)
    MSUBw.x VF16x, VF01x, VF14w       LQI VF18, (VI05++)
    NOP                      NOP
    NOP                      NOP
    MULA.xyzw ACC, VF01, VF17         NOP
    MADDx.xyzw VF21, VF18, VF16x      NOP
    NOP                      NOP
    NOP                      NOP
    NOP                      NOP
    NOP                      SQI VF21, (VI07++)

    NOP                      JR (VI15)      ; return
    NOP                      NOP            ; BDSlot

;--------------------------------------
;Optimized version
;--------------------------------------
DISPR_O:
    NOP                      IADDI VI01, VI00, 8
    NOP                      LQI VF16, (VI06++)

    NOP                      RNEXT VF16x
    NOP*                     NOP*
    NOP                      RNEXT VF16y
    NOP*                     NOP*

lr: MULAw.x ACCx, VF16x, VF14w        LQI VF17, (VI04++)
    MSUBw.x VF16x, VF01x, VF14w       LQI VF18, (VI05++)
    MULAw.y ACCy, VF16y, VF14w        LQI VF19, (VI04++)
    MSUBw.y VF16y, VF01y, VF14w       LQI VF20, (VI05++)
    MULA.xyzw ACC, VF01, VF17         RNEXT VF16z
    MADDx.xyzw VF21, VF18, VF16x      NOP
    MULA.xyzw ACC, VF01, VF19         RNEXT VF16w
    MADDy.xyzw VF22, VF20, VF16y      NOP
    MULAw.z ACCz, VF16z, VF14w        LQI VF17, (VI04++)
    MSUBw.z VF16z, VF01z, VF14w       LQI VF18, (VI05++)
    MULAw.w ACCw, VF16w, VF14w        LQI VF19, (VI04++)
    MSUBw.w VF16w, VF01w, VF14w       LQI VF20, (VI05++)
    MULA.xyzw ACC, VF01, VF17         SQI VF21, (VI07++)
    MADDz.xyzw VF21, VF18, VF16z      SQI VF22, (VI07++)
    MULA.xyzw ACC, VF01, VF19         RNEXT VF16x
    MADDw.xyzw VF22, VF20, VF16w      RNEXT VF16y
    NOP                      IADDI VI01, VI01, -1
    NOP                      SQI VF21, (VI07++)
    NOP                      IBNE VI01, VI00, Lr:
    NOP                      SQI VF22, (VI07++)  ; BDSlot
```

```
    NOP                        RNEXT VF16x
    NOP*                       NOP*
    NOP*                       NOP*
    NOP*                       NOP*
    MULAw.x ACCx, VF16x, VF14w        LQI VF17, (VI04++)
    MSUBw.x VF16x, VF01x, VF14w       LQI VF18, (VI05++)
    NOP*                       NOP*
    NOP*                       NOP*
    MULA.xyzw ACC, VF01, VF17         NOP
    MADDx.xyzw VF21, VF18, VF16x      NOP
    NOP*                       NOP*
    NOP*                       NOP*
    NOP                        JR (VI15)          ; return
    NOP                        SQI VF21, (VI07++)  ; BDSlot
```

Main routine

Processing such as initialization or polygon generation should be performed by the EE Core originally. This program, however, is described as a sample using micro mode instructions.

```
;--------------------------------------------------------------------
;main routine
;--------------------------------------------------------------------
; This was originally a EE Core program.

; pBASE0 Base address 0 for temporary buffer
; pBASE1 Base address 1 for temporary buffer

; pTEX(VI00) is the address in which the texture coordinates reside.
; pBT32(VI00) is the address in which the 32-part partition blending table resides.
; pCO_P0(VI00) is the address in which the patch 0 control point resides.

; The following buffers take each of VI11 and VI12 as the base address.
; pQP(VI11) and pQP(VI12) are the output buffer addresses for partition coordinates of curved
surface partitioning
; pOP(VI11) and pOP(VI12) are the output buffer addresses for coordinate conversion +
perspective converted coordinates
; pOtex(VI11) and pOtex(VI11) are the output buffer addresses for partitioned texture coordinates
; pN(VI11) and pN(VI12) are normal output buffer addresses
; pRGB(VI11) and pRGB(VI12) are RGB output buffer addresses


; Main routine
; Main routine initialization processing
    NOP                        IADDI VI01, VI00, SEED
    NOP                        RINIT VI01          ; SEED set
    NOP                        IADDI VI11, VI00, #pBASE0
    NOP                        IADDI VI12, VI00, #pBASE1
; Texture partitioning initialization processing
    NOP                        IADDI VI04, VI00, #pTEX
    NOP                        LQI.xy VF29xy, (VI04++)
    NOP                        LQI.xy VF30xy, (VI04++)
    NOP                        NOP
    NOP                        NOP
    NOP                        NOP
    SUB.xy VF30xy, VF30xy, VF29xy  NOP
    NOP                        NOP
    NOP                        NOP
    NOP                        NOP
```

```
    MULx.xy VF30xy, VF30xy, VF31x  NOP      ; (x,y) * 1/32

    NOP                    IADDI VI02, VI00, 33

Lmain:
; Curved surface generation (Generates nth column, 33 vertices)
    NOP                    IADDI VI04, VI00, #pBT32
    NOP                    IADDI VI05, VI00, #pBT32+1
    NOP                    IADDI VI06, VI00, #pCP_P0
    NOP                    BAL VI15, SURF_OUTP: ; subroutine call
    NOP                    IADDI VI07, VI12, #pQP; BDSlot
; Coordinate conversion + perspective conversion
    NOP                    IADDI VI04, VI12, #pQP
    NOP                    BAL VI15, PRSP_O:    ; subroutine call
    NOP                    IADDI VI05, VI12, #pOP; BDSlot

; Using the vertices for two columns, the vector product is calculated, normal is derived, and the light source is calculated.
; Illumination processing is performed by the vertex in the modeling coordinate system prior to coordinate conversion + perspective conversion.
;Normal calculation + texture coordinate partitioning
; Setting parameters for texture coordinate partitioning
    NOP                    IADDI VI13, VI12, #pOtex; BDSlot
;Setting parameters for normal (vector product) calculation
    NOP                    IADDI VI04, VI11, #pQP
    NOP                    IADDI VI05, VI12, #pQP
    NOP                    BAL VI15, NORM_TEXD: ; subroutine call
    NOP                    IADDI VI06, VI12, #pN; BDSlot
;Processing illumination using four parallel light sources
    NOP                    IADDI VI04, VI12, #pN
    NOP                    IADDI VI06, VI00, NS
    NOP                    BAL VI15, LIGHT_O:   ; subroutine call
    NOP                    IADDI VI05, VI12, #pRGB; BDSlot

; Outputting strip triangles
    NOP                    IADDI VI04, VI12, #pOP
    NOP                    IADDI VI05, VI11, #pOP
    NOP                    IADDI VI06, VI12, #pOtex
    NOP                    IADDI VI07, VI11, #pOtex
    NOP                    IADDI VI08, VI12, #pRGB
    NOP                    BAL VI15, OUTP:   ; subroutine call
    NOP                    IADDI VI09, VI11, #pRGB
;* * * * * * * * * * * * * * * * * * *

;swap VI11<-->VI12
    NOP                    IADD VI13, VI00, VI11
    NOP                    IADD VI11, VI00, VI12
    NOP                    IADD VI12, VI00, VI13
; v loop
    NOP                    IADDI VI02, VI02, -1
    NOP                    IBNE VI02, VI00, Lmain:
    NOP                    NOP              ; BDSlot

; Outputting final strip triangles
    NOP                    IADDI VI08, VI11, #pOP
    NOP                    IADDI VI09, VI12, #pOP
    NOP                    IADDI VI10, VI11, #pOtex
    NOP                    IADDI VI11, VI12, #pOtex
    NOP                    IADDI VI12, VI11, #pRGB
    NOP                    BAL VI15, OUTP:      ; subroutine call
```

```
NOP                     IADDI VI13, VI12, #pRGB; BDSlot
```

7.2. EFU Processing

The processing of the elementary function calculation unit, EFU, built into VU1, is shown as pseudo microinstructions for reference purpose.

Registers

Input registers

VN00: 32 bits: x reg

VN01: 32 bits: y reg (Can be used with tmp3)

VN02: 32 bits: z reg (Can be used with tmp2)

VN03: 32 bits: w reg (Can be used with tmp1)

Output register

VN04: 32 bits: p reg

tmp registers

VN05: 32 bits: tmp1

VN06: 32 bits: tmp2

VN07: 32 bits: tmp3

Constant registers

VN08: 32 bits: 1

VN09: 32 bits: S1

VN10: 32 bits: S2

VN11: 32 bits: S3

VN12: 32 bits: S4

VN13: 32 bits: S5

VN14: 32 bits: T1

VN15: 32 bits: T2

VN16: 32 bits: T3

VN17: 32 bits: T4

VN18: 32 bits: T5

VN19: 32 bits: T6

VN20: 32 bits: T7

VN21: 32 bits: T8

VN22: 32 bits: PI/4

VN23: 32 bits: E1

VN24: 32 bits: E2

VN25: 32 bits: E3

VN26: 32 bits: E4

VN27: 32 bits: E5

VN28: 32 bits: E6

VN29: 32 bits: ----

VN30: 32 bits: ----

VN31: 32 bits: ----

Others

ACC

The input copies the four fields (128 bits) of the source register to the x, y, z, and w registers or copies an arbitrary field (32 bits) of the source register to the x register.
The output register is p.

EATAN

[description]
x <- VF[fs]fsf
p = arctan(x) (0 =< x =< 1)
p = (T1 x Y + T2 x Y^3 + T3 x Y^5 + T4 x Y^7
+ T5 x Y^9 + T6 x Y^11 + T7 x Y^13 + T8 x Y^15)+ PI/4
Y = (X - 1) / (X + 1)

[nano code]

```
ADD     tmp1, x, 1
SUB     tmp2, x, 1
nop                 ; Arctans are the same after this.
nop
nop
DIV     x, tmp2, tmp1
nop
nop
nop
nop
nop
nop
nop
MUL     tmp3, x, x        ; x^2
MULA    ACC, T1, x
nop
nop
MUL     tmp1, tmp3, x     ; x^3
nop
nop
nop
MUL     tmp2, tmp1, tmp3 ; x^5
MADDA   ACC, tmp1, T2
nop
nop
MUL     tmp1, tmp2, tmp3 ; x^7
MADDA   ACC, tmp2, T3
nop
nop
MUL     tmp2, tmp1, tmp3 ; x^9
MADDA   ACC, tmp1, T4
nop
nop
MUL     tmp1, tmp2, tmp3 ; x^11
MADDA   ACC, tmp2, T5
nop
nop
MUL     tmp2, tmp1, tmp3 ; x^13
MADDA   ACC, tmp1, T6
nop
nop
MUL     tmp1, tmp2, tmp3 ; x^15
MADDA   ACC, tmp2, T7
MADDA   ACC, 1, PI/4
nop
MADD    p, tmp1, T8
nop
nop
nop
Outputting p.
```

EATANxy

[description]
x <- VF[ft]x
y <- VF[ft]y
z <- VF[ft]z
w <- VF[ft]w
p = arctan(y/x) (0 =< y =< x)

[nano code]

```
ADD   tmp1, y, x
SUB   tmp2, y, x
nop           ; Same processing as arctan follows.
......
```

EATANxz

[description]
x <- VF[ft]x
y <- VF[ft]y
z <- VF[ft]z
w <- VF[ft]w
p = arctan(z/x) (0 =< z =< x)

[nano code]

```
ADD   tmp1, z, x
SUB   tmp2, z, x
nop           ; Same processing as arctan follows.
......
```

EEXP

[description]

x <- VF[fs]fsf

p = exp(-x) (0 =< x =< +MAX)

[nano code]

```
MUL     tmp1, x, x       ; x^2
MULA    ACC, x, E1       ; E1 * x
nop
nop
MUL     tmp2, tmp1, x    ; x^3
MADDA   ACC, tmp1, E2    ; E2 * x^2
nop
nop
MUL     tmp1, tmp2, x    ; x^4
MADDA   ACC, tmp2, E3    ; E3 * x^3
nop
nop
MUL     tmp2, tmp1, x    ; x^5
MADDA   ACC, tmp1, E4    ; E4 * x^4
nop
nop
MUL     tmp1, tmp2, x    ; x^6
MADDA   ACC, tmp2, E5    ; E5 * x^5
MADDA   ACC, 1, 1        ; + 1
nop
MADD    p, tmp1, E6      ; E6 * x^6
nop
nop
nop
MUL     p, p, p          ; p^2
nop
nop
nop
MUL     p, p, p          ; p^4
nop
nop
nop
DIV     p, 1, p
nop
nop
nop
nop
nop
nop
nop
```

Outputting p.

ELENG

[description]
x <- VF[ft]x
y <- VF[ft]y
z <- VF[ft]z
w <- VF[ft]w
$p = \sqrt{(x*x+y*y+z*z)}$

[nano code]
MULA ACC, x, x
MADDA ACC, y, y
MADD p, z, z
nop
nop
nop
SQRT p, 1, p
nop
nop
nop
nop
nop
nop
nop
Outputting p.

ERCPR

[description]
x <- VF[fs]fsf
p = 1 / x

[nano code]
DIV p, 1, x
nop
nop
nop
nop
nop
nop
nop
Outputting p.

ERLENG

[description]
x <- VF[ft]x
y <- VF[ft]y
z <- VF[ft]z
w <- VF[ft]w
$p = \frac{1}{\sqrt{(x*x+y*y+z*z)}}$

[nano code]
MULA ACC, x, x
MADDA ACC, y, y
MADD p, z, z
nop
nop
nop
RSQRT p, 1, p
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
Outputting p.

ERSADD

[description]
x <- VF[ft]x
y <- VF[ft]y
z <- VF[ft]z
w <- VF[ft]w
p = 1 / (x*x + y*y + z*z)

[nano code]
MULA ACC, x, x
MADDA ACC, y, y
MADD p, z, z
nop
nop
nop
DIV p, 1, p
nop
nop
nop
nop
nop
nop
nop
Outputting p.

ERSQRT

[description]
x <- VF[fs]dest
$p = \frac{1}{\sqrt{x}}$

[nano code]
RSQRT p, 1, x
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
nop
Outputting p.

ESADD

[description]
x <- VF[ft]x
y <- VF[ft]y
z <- VF[ft]z
w <- VF[ft]w
p = x*x + y*y + z*z

[nano code]
MULA ACC, x, x
MADDA ACC, y, y
MADD p, z, z
nop
nop
nop
Outputting p.

ESIN

[description]
x <- VF[fs]fsf
p = sin(x) (-π/2 =< x =< +π/2)
p = S1 x X + S2 x X^3 + S3 x X^5 + S4 x X^7 + S5 x X^9

[nano code]
```
MUL    tmp3, x, x        ; x^2
MULA   ACC, x, S1
nop
nop
MUL    tmp1, tmp3, x     ; x^3
nop
nop
nop
MUL    tmp2, tmp1, tmp3 ; x^5
MADDA  ACC, tmp1, S2
nop
nop
MUL    tmp1, tmp2, tmp3 ; x^7
MADDA  ACC, tmp2, S3
nop
nop
MUL    tmp2, tmp1, tmp3 ; x^9
MADDA  ACC, tmp1, S4
nop
nop
MADD   p, tmp2, S5
nop
nop
nop
```
Outputting p.

ESQRT

[description]

x <- VF[fs]fsf
$p = \sqrt{x}$

[nano code]

```
SQRT    p, 1, x
nop
nop
nop
nop
nop
nop
nop
```

Outputting p.

ESQUR

[description]

x <- VF[fs]fsf
p = x*x

[nano code]

```
MUL     p, x, x
nop
nop
nop
```

Outputting p.

ESUM

[description]

x <- VF[ft]x
y <- VF[ft]y
z <- VF[ft]z
w <- VF[ft]w
p = x + y + z + w

[nano code]

```
MULA    ACC, x, 1
MADDA   ACC, y, 1
MADDA   ACC, z, 1
MADD    p,   w, 1
nop
nop
nop
```

Outputting p.

7.3. Micro Subroutine Debugging

7.3.1. Debug Flow

Micro subroutines can be debugged as follows: Stop the micro subroutine with the D bit or the T bit, examine the VU status by using CFC2 or other instructions, and restart execution referring to the TPC.

Stop of Micro Subroutine

When the D bit or the T bit is set to 1 in a microinstruction, the VU enters the Stop state after executing the instruction, an interrupt is sent to the EE Core, and the address after the last instruction executed is stored in the TPC. Unlike when setting the E bit, there is no delay slot, and the following microinstruction is canceled during its execution.

However, if the D bit or T bit is set in a branch instruction, the following instruction is the last instruction executed, and the branch destination address is stored in the TPC.

The D bit and the T bit function similarly, but use the T bit in the application since the debugger uses the D bit.

Forcible Stop of Micro Subroutine

To forcibly stop execution of a micro subroutine externally, write 1 to the FB bit of the FBRST register using the CTC2 instruction. This generates a ForceBreak, and the VU enters the Stop state.

However, correct re-execution of the micro subroutine after the forcible stop is not guaranteed.

Access to VU Resources

The methods of accessing the VU registers and other resources are shown in the table below. For details, refer to "**5. Macro Mode**", and other corresponding macro instruction descriptions.

VU0 Resource	Access Method
Floating-point register	COP2 data register (CPR[2,00] - CPR[2,31])
Integer register	COP2 control register (CCR[2,00] - CCR[2,15])
Special register	COP2 control register (CCR[2,20] - CCR[2,22])
Flag	COP2 control register (CCR[2,16] - CCR[2,18])
VU Mem0	Main memory 0x1100_4000 - 0x1100_4ff0 (or VLQD/VLQI/VSQD/VSQI instruction)
MicroMem0	Main memory 0x1100_0000 - 0x1100_0ff0

Accesses to VU Mem0 and MicroMem0 are possible only when VU0 is stopped.

VU1 Resource	Access Method
Floating-point register	VU Mem0 addresses 0x4000 - 0x41ff
Integer register	VU Mem0 addresses 0x4200 - 0x42ff
Special register	VU Mem0 addresses 0x4340 - 0x437f
Flag	VU Mem0 addresses 0x4300 - 0x432f
VU Mem1	Main memory 0x1100_c000 - 0x1100_fff0
MicroMem1	Main memory 0x1100_8000 - 0x1100_bff0

Accesses to VU Mem1 and MicroMem1 are possible only when VU1 is stopped. Since the above methods of accessing VU Mem1 and MicroMem1 are strictly for the purpose of debugging with a considerable speed penalty involved, transfer data via VIF in ordinary cases.

Execution Restart

To restart execution of a VU0 micro subroutine, read the TPC with an instruction such as CFC2, transfer the value to CMSAR0 with the CTC2 instruction, and restart execution with VCALLMSR.

In VU1, read the TPC with an instruction such as VILWR or CFC2 and transfer the value to CMSAR1 with the CTC2 instruction to restart execution.

Control of Debugging Function

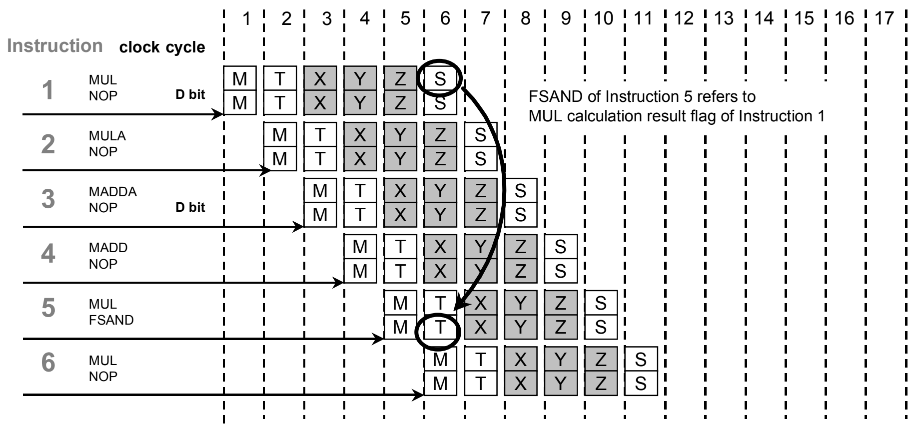
Stopping micro programs with the D bit/T bit can be controlled with the DE bit/TE bit of the control register. Stopping a microprogram with the D bit is permitted only when the DE bit is set to 1; similarly for the T bit and TE bit.

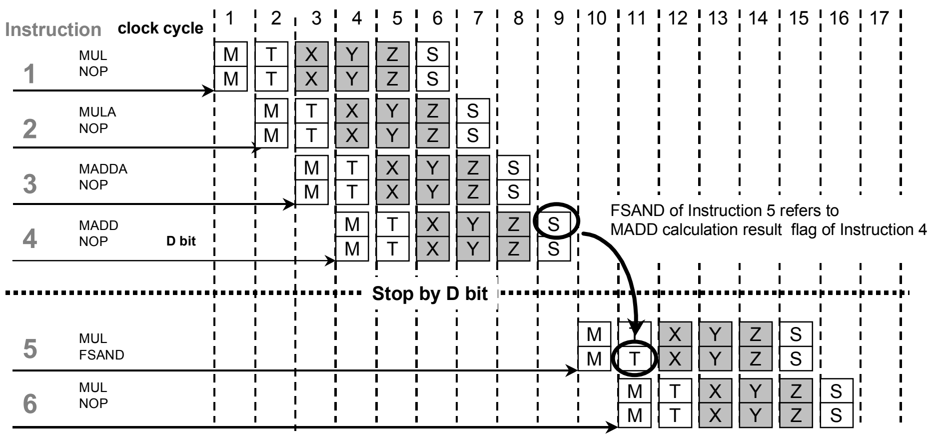
Debugging efficiency can be improved by setting the DE bit only when a routine that becomes a problem is executed while debugging a large-scale program.

7.3.2. Notes on Re-execution

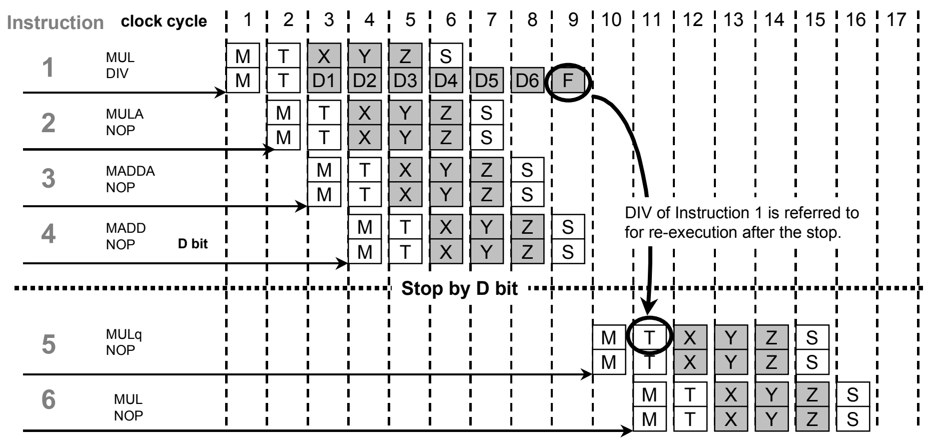
Since the VU performs pipeline operation, continuous execution of a micro subroutine and re-execution after stopping during its execution may sometimes cause different results. It is recommended to stop at a place where the re-execution result is guaranteed by paying attention to the instructions before and after the stop by the D bit.

The figures below show how results can differ between continuous execution of a micro subroutine and re-execution after stopping during its execution with the D bit.





A flag is referred to by the FSAND instruction in Instruction 5. Although it is the calculation result of Instruction 1 in continuous execution, the calculation result of Instruction 4 is referred to in re-execution after stopping.
The following figure illustrates an example in which the value of the Q register (to be referred to when the micro subroutine is re-executed after the stop) is changed.



7.4. Throughput / Latency List

Micro Instruction	Macro Instruction	Throughput	Latency
ABS	VABS	1	4
ADD	VADD	1	4
ADDi ADDq	VADDi VADDq	1	4
ADDbc	VADDbc	1	4
ADDA	VADDA	1	4
ADDAi ADDAq	VADDAi VADDAq	1	4
ADDAbc	VADDAbc	1	4
B	-----	2	2
BAL	-----	2	2
-----	BC2F	2	2
-----	BC2FL	2	2
-----	BC2T	2	2
-----	BC2TL	2	2
-----	VCALLMS	-----	-----
-----	VCALLMSR	-----	-----
CLIP	VCLIP	1	4
-----	CFC2	1	1
-----	CTC2	1	1
DIV	VDIV	7	7
EATAN	-----	53	54
EATANxy	-----	53	54
EATANxz	-----	53	54
EEXP	-----	43	44
ELENG	-----	17	18
ERCPR	-----	11	12
ERLENG	-----	23	24
ERSADD	-----	17	18
ERSQRT	-----	17	18
ESADD	-----	10	11
ESIN	-----	28	29
ESQRT	-----	11	12
ESUM	-----	11	12
FCAND	-----	1	1
FCEQ	-----	1	1
FCGET	-----	1	1
FCOR	-----	1	1
FCSET	-----	1	4
FMAND	-----	1	1
FMEQ	-----	1	1
FMOR	-----	1	1
FSAND	-----	1	1
FSEQ	-----	1	1
FSOR	-----	1	1
FSSET	-----	1	4
FTOI0 FTOI4 FTOI12 FTOI15	VFTOI0 VFTOI4 VFTOI12 VFTOI15	1	4
IADD	VIADD	1	1
IADDI	VIADDI	1	1
IAND	VIAND	1	1

Micro Instruction	Macro Instruction	Throughput	Latency
IADDIU	-----	1	1
IBEQ	-----	2	2
IBGEZ	-----	2	2
IBGTZ	-----	2	2
IBLEZ	-----	2	2
IBLTZ	-----	2	2
IBNE	-----	2	2
ILW	-----	1	4
ILWR	VILWR	1	4
IOR	VIOR	1	1
ISUB	VISUB	1	1
ISUBIU	-----	1	1
ISW	-----	1	4
ISWR	VISWR	1	4
ITOF0 ITOF4 ITOF12 ITOF15	VITOF0 VITOF4 VITOF12 VITOF15	1	4
JALR	-----	2	2
JR	-----	2	2
LQ	-----	1	4
-----	LQC2	1	1
LQD	VLQD	1	4
LQI	VLQI	1	4
MADD	VMADD	1	4
MADDi MADDq	VMADDi VMADDq	1	4
MADDbc	VMADDbc	1	4
MADDA	VMADDA	1	4
MADDAi MADDAq	VMADDAi VMADDAq	1	4
MADDAbc	VMADDAbc	1	4
MAX	VMAX	1	4
MAXi	VMAXi	1	4
MAXbc	VMAXbc	1	4
MFIR	VMFIR	1	4
MFP	-----	1	4
MINI	VMINI	1	4
MINIi	VMINIi	1	4
MINIbc	VMINIbc	1	4
MOVE	VMOVE	1	4
MR32	VMR32	1	4
MSUB	VMSUB	1	4
MSUBi MSUBq	VMSUBi VMSUBq	1	4
MSUBbc	VMSUBbc	1	4
MSUBA	VMSUBA	1	4
MSUBAi MSUBAq	VMSUBAi MSUBAq	1	4
MSUBAbc	VMSUBAbc	1	4
MTIR	VMTIR	1	1
MUL	VMUL	1	4
MULi MULq	VMULi VMULq	1	4
MULbc	VMULbc	1	4
MULA	VMULA	1	4
MULAi MULAq	VMULAi VMULAq	1	4
MULAbc	VMULAbc	1	4
NOP	VNOP	1	4

Micro Instruction	Macro Instruction	Throughput	Latency
OPMULA	VOPMULA	1	4
OPMSUB	VOPMSUB	1	4
-----	QMFC2	1	1
-----	QMTC2	1	1
RGET	VRGET	1	4
RINIT	VRINIT	1	1
RNEXT	VRNEXT	1	4
RSQRT	VRSQRT	13	13
RXOR	VRXOR	1	1
SUB	VSUB	1	4
SUBi SUBq	VSUBi VSUBq	1	4
SUBbc	VSUBbc	1	4
SUBA	VSUBA	1	4
SUBAi SUBAq	VSUBAi VSUBAq	1	4
SUBAbc	VSUBAbc	1	4
SQ	-----	1	4
-----	SQC2	1	1
SQD	VSQD	1	4
SQI	VSQI	1	4
SQRT	VSQRT	7	7
WAITP	-----	1~54	1~54
WAITQ	VWAITQ	1~13	1~13
XGKICK	-----	1~	1~
XITOP	-----	1	1
XTOP	-----	1	1

(This page is left blank intentionally)