]>

IPUV

Reference Manual

Table of Contents

1 Core: IPUV

1.1 Registers

1.1.1 *CC_ZNCO*

Z	N	C	O
3	2	1	0

Description:

Internal Register: The condition coder register:

Fields:

Field	Range	Description
O	[0]	Overflow flag: signed arithmetic's produced an value overflow
C	[1]	Carry flag: unsigned arithmetic's produced an value overflow
N	[2]	Negative flag: result is negative (bit[15] is set)
Z	[3]	Zero flag: result is 0

1.1.2 *CONFAAG*

HALF	SGN
1	0

Reset value: 1

Description:

Core Register: Configuration register for ABS_ANGLE Accelerartor

Fields:

Field	Range	Description

Field	Range	Description
SGN	[0]	Inputs are in signed or unsigned logic • 0: unsigned • 1: signed
HALF	[1]	genrate angle in (0-15) << 12 (0-360�) or (0-8) << 12 (0-180�) • 0: 0-360� • 1: 0-180�

1.1.3 *CONFALU*

OPIX0AUTOINC	ACCSHR	SHR		SAT	SGN
12	11 9	8 4	3 2	1	0

Reset value: 0x2

Description:

Core Register: Configuration register for arithmentic's in ALU

Fields:

Field	Range	Description
SGN	[0]	ALU uses signed or unsigned logic • 0: unsigned • 1: signed
SAT	[1]	ALU results are saturated according to signed/unsigned or overflow • 0: overflow • 1: saturate
SHR	[8,4]	MULH 32 bit result is shifted right by CONFALU.SHR bits to implement fixed point arithmetic's. The result after shift is 16 bit wide. The valid value range is from 0 to 16.
ACCSHR	[11,9]	ACC 24 bit result is shifted right by CONFALU.ACCSHR bits to implement fixed point arithmetic's. The result after shift is 16 bit wide. The valid value range is from 0 to 7.
OPIX0AUTOINC	[12]	Perform no automatic increment of OPIXA0 when OPIX0 is accessed for reading or writing (0=autoinc 1=no autoinc)

1.1.4 *CSE_SET_LOCK*

CSE_SET_LOCK
0

Reset value: 0

Description:

Input Signal: Indicate if the core runs in locked mode (no debug, register read out)

- 1 = Locked / Secure Mode

1.1.5 *HD_BREAKENABLE*

HD_BREAKENABLE

7	0

Reset value: 0

Description:

Host Debug Register: Enable Break Conditions.

1.1.6 *H_CURRPOS*

XPOS	YPOS
31 16	15 0

Description:

Host Register: Position of current line processed (read only)

Fields:

Field	Range	Description
YPOS	[15,0]	YPOS (same value as current core register)
XPOS	[31,16]	XPOS (same value as current core register)

1.1.7 *H_CURRXCFG*

XSIZE		SHIFT
31 16	15 2	1 0

Reset value: 0x04000001

Description:

Host Register: X counter configuration for current line

Fields:

Field	Range	Description
SHIFT	[1,0]	Progress XPOS by SHIFT pixels (constant value of 1)
XSIZE	[31,16]	XSIZE (to be compared with XPOS to detect end of line)

1.1.8 *H_DREGA*

H_DREGA
7 0

Description:

Host Register: Data register address register

1.1.9 *H_DREGD*

H_DREGD	
15	0

Description:

> Host Register: Data register value at address H_DREGA. Any access to H_DREGD will progress H_DREGA to the next register address

1.1.10 *H_IMEMA*

H_IMEMA	
5	0

Description:

> Host Register: IMEM address register

1.1.11 *H_IMEMD*

H_IMEMD	
31	0

Description:

> Host Register: IMEM data value at address H_IMEMA Any access to H_IMEMD will progress H_IMEMA to the next instruction address

1.1.12 *H_INCFG*

CURRCFG									
[31,29]	CURRLINE4[28]	CURRLINE3[27]	CURRLINE2[26]	CURRLINE1[25]	CURRLINE0[24]	[23,19]	CURRNHOOD[18]	CURRSHIFT[17,16]	[15,1
31								16	15

Reset value: 0x00010001

Description:

> Host Register: Input Matrix configuration

Fields:

Field	Range	Description
NEXTCFG	[15,0]	Setup for next line to be processed (becomes valid with next write to H_START)
SHIFT	[1,0]	Next line shifts input matrix by CURRSHIFT pixel positions • 0: shift by 4 pixels • 1: shift by 1 pixel • 2: shift by 2 pixels • 3: shift by 3 pixels
NHOOD	[2]	next line neighborhood type • 0: 2D (5x5) • 1: not allowed
LINE0	[8]	Indicates if line 0 is enabled for next line (for IN[0] to IN[7]) • 1: enabled

Field	Range	Description
LINE1	[9]	Indicates if line 1 is enabled for next line (for IN[8] to IN[15]) • 1: enabled
LINE2	[10]	Indicates if line 2 is enabled for next line (for IN[16] to IN[23]) • 1: enabled
LINE3	[11]	Indicates if line 3 s enabled for next line (for IN[24] to IN[31]) • 1: enabled
LINE4	[12]	Indicates if line 4 s enabled for next line (for IN[32] to IN[39]) • 1: enabled
CURRCFG (read-only)	[31,16]	Status of current line to be processed (read only)
CURRSHIFT	[17,16]	Current line shifts input matrix by CURRSHIFT pixel positions • 0: shift by 4 pixels • 1: shift by 1 pixel • 2: shift by 2 pixels • 3: shift by 3 pixels
CURRNHOOD	[18]	Current line neighborhood type • 0: 2D (5x5) • 1: not allowed
CURRLINE0	[24]	Indicates if currently line 0 is enabled (for IN[0] to IN[7]) • 1: enabled
CURRLINE1	[25]	Indicates if currently line 1 is enabled (for IN[8] to IN[15]) • 1: enabled
CURRLINE2	[26]	Indicates if currently line 2 is enabled (for IN[16] to IN[23]) • 1: enabled
CURRLINE3	[27]	Indicates if currently line 3 is enabled (for IN[24] to IN[31]) • 1: enabled
CURRLINE4	[28]	Indicates if currently line 4 is enabled (for IN[32] to IN[39]) • 1: enabled

1.1.13 *H_OUTCFG*

CURRCFG						NEXTCFG					
[31,27]	CURRLINE2[26]	CURRLINE1[25]	CURRLINE0[24]	[23,18]	CURRSHIFT[17,16]	[15,12]	LINE3[11]	LINE2[10]	LINE1[9]	LINE0[8]	[7
31					16	15					

Description:

Host Register: Output configuration

Fields:

Field	Range	Description
NEXTCFG	[15,0]	Setup for next line to be processed (becomes valid with next write top H_START)

Field	Range	Description
SHIFT	[1,0]	Next line shifts input matrix by SHIFT pixel positions • 0: shift by 4 pixels • 1: shift by 1 pixel • 2: shift by 2 pixels • 3: shift by 3 pixels
LINE0	[8]	Indicates if line 0 is enabled for next line, connected to OUT(0) • 1: enabled
LINE1	[9]	Indicates if line 1 is enabled for next line, connected to OUT(4) • 1: enabled
LINE2	[10]	Indicates if line 2 is enabled for next line, connected to OUT(8) • 1: enabled
LINE3	[11]	Indicates if line 3 is enabled for next line (not used) • 1: enabled
CURRCFG (read-only)	[31,16]	Status of current line to be processed (read only)
CURRSHIFT	[17,16]	Current line shifts input matrix by CURRSHIFT pixel positions • 0: shift by 4 pixels • 1: shift by 1 pixel • 2: shift by 2 pixels • 3: shift by 3 pixels
CURRLINE0	[24]	Indicates if currently line 0 is enabled, connected to OUT(0) • 1: enabled
CURRLINE1	[25]	Indicates if currently line 1 is enabled, connected to OUT(4) • 1: enabled
CURRLINE2	[26]	Indicates if currently line 2 is enabled, connected to OUT(8) • 1: enabled

1.1.14 *H_POS*

XPOS	YPOS
31 16	15 0

Description:

Host Register: Start Position for next line to be processed

Fields:

Field	Range	Description
YPOS	[15,0]	YPOS
XPOS	[31,16]	XPOS

1.1.15 *H_START*

	CURR		NEXT

31 24	23 16	15 8	7 0

Description:

Host Register: Trigger new line

Fields:

Field	Range	Description
NEXT	[7,0]	start address for next line to be processed (read/write)
CURR (read-only)	[23,16]	Instruction start address for current line being processed (read only)

1.1.16 *H_STATUS*

REMAINING		DEBUG	STATUS BUFFERED[9]	STATUS RUN[8]		EVENTS
31 16	15 11	10	9	8	7 2	1 0

Description:

Host Register: Reads the operation status, clear events

Fields:

Field	Range	Description
EVENTS	[1,0]	Number of events, write to decrement
STATUS	[9,8]	Status • 0: idle • 1: running • 3: running and buffered
RUN	[8]	Engine is currently running • 1: running
BUFFERED	[9]	Engine has a buffered setup. Do not add/modify line setup anymore until buffered became active (running). • 1: buffered
DEBUG	[10]	Debug Mode • 0: normal • 1: debug mode
REMAINING	[31,16]	Number of remaining pixels

1.1.17 *H_XCFG*

XSIZE		SHIFT
31 16	15 2	1 0

Reset value: 0x04000001

Description:

Host Register: X counter configuration for next line

Fields:

Field	Range	Description

Field	Range	Description
SHIFT	[1,0]	Progress XPOS by SHIFT pixels (constant value of 1)
XSIZE	[31,16]	XSIZE (to be compared with XPOS to detect end of line)

1.1.18 *IADDR*

IADDR	
7	0

Description:

Core Register: Current instruction byte address. The two LSBs are alway zero.

1.1.19 *I_BUFFERED*

I_BUFFERED
0

Reset value: 0

Description:

Internal Register: Core has buffered setup for next line. Is identical to H_STATUS.BUFFERED

1.1.20 *I_DEBUGMODE*

I_DEBUGMODE
0

Reset value: 0

Description:

Internal Register: Debug Mode. Is identical to H_STATUS.DEBUG

1.1.21 *I_EVENTS*

I_EVENTS	
1	0

Reset value: 0

Description:

Internal Register: Number of processed lines. Is identical to H_STATUS.EVENTS

- counts up per done line
- counts down per cleared event

1.1.22 *I_INCFG*

CURRCFG									
[31,29]	CURRLINE4[28]	CURRLINE3[27]	CURRLINE2[26]	CURRLINE1[25]	CURRLINE0[24]	[23,19]	CURRNHOOD[18]	CURRSHIFT[17,16]	[15,1
31								16	15

Reset value: 0x00010001

Description:

Internal Register: Identical to H_INCFG (Input Matrix configuration)

Fields:

Field	Range	Description
NEXTCFG	[15,0]	Setup for next line to be processed (becomes valid with next write to H_START)
SHIFT	[1,0]	Next line shifts input matrix by CURRSHIFT pixel positions • 0: shift by 4 pixels • 1: shift by 1 pixel • 2: shift by 2 pixels • 3: shift by 3 pixels
NHOOD	[2]	next line neighborhood type • 0: 2D (5x5) • 1: not allowed
LINE0	[8]	Indicates if line 0 is enabled for next line (for IN[0] to IN[7]) • 1: enabled
LINE1	[9]	Indicates if line 1 is enabled for next line (for IN[8] to IN[15]) • 1: enabled
LINE2	[10]	Indicates if line 2 is enabled for next line (for IN[16] to IN[23]) • 1: enabled
LINE3	[11]	Indicates if line 3 s enabled for next line (for IN[24] to IN[31]) • 1: enabled
LINE4	[12]	Indicates if line 4 s enabled for next line (for IN[32] to IN[39]) • 1: enabled
CURRCFG	[31,16]	Status of current line to be processed (read only)
CURRSHIFT	[17,16]	Current line shifts input matrix by CURRSHIFT pixel positions • 0: shift by 4 pixels • 1: shift by 1 pixel • 2: shift by 2 pixels • 3: shift by 3 pixels
CURRNHOOD	[18]	Current line neighborhood type • 0: 2D (5x5) • 1: not allowed
CURRLINE0	[24]	Indicates if currently line 0 is enabled (for IN[0] to IN[7]) • 1: enabled
CURRLINE1	[25]	Indicates if currently line 1 is enabled (for IN[8] to IN[15]) • 1: enabled
CURRLINE2	[26]	Indicates if currently line 2 is enabled (for IN[16] to IN[23]) • 1: enabled
CURRLINE3	[27]	Indicates if currently line 3 is enabled (for IN[24] to IN[31]) • 1: enabled

Field	Range	Description
CURRLINE4	[28]	Indicates if currently line 4 is enabled (for IN[32] to IN[39]) • 1: enabled

1.1.23 *I_OUTCFG*

CURRCFG						NEXTCFG					
[31,27]	CURRLINE2[26]	CURRLINE1[25]	CURRLINE0[24]	[23,18]	CURRSHIFT[17,16]	[15,12]	LINE3[11]	LINE2[10]	LINE1[9]	LINE0[8]	[7
31					16	15					

Description:

Host Register: Output configuration

Fields:

Field	Range	Description
NEXTCFG	[15,0]	Setup for next line to be processed (becomes valid with next write to H_START)
SHIFT	[1,0]	Next line shifts input matrix by SHIFT pixel positions • 0: shift by 4 pixels • 1: shift by 1 pixel • 2: shift by 2 pixels • 3: shift by 3 pixels
LINE0	[8]	Indicates if line 0 is enabled for next line, connected to OUT(0) • 1: enabled
LINE1	[9]	Indicates if line 1 is enabled for next line, connected to OUT(4) • 1: enabled
LINE2	[10]	Indicates if line 2 is enabled for next line, connected to OUT(8) • 1: enabled
LINE3	[11]	Indicates if line 3 is enabled for next line (not used) • 1: enabled
CURRCFG	[31,16]	Status of current line to be processed (read only)
CURRSHIFT	[17,16]	Current line shifts input matrix by CURRSHIFT pixel positions • 0: shift by 4 pixels • 1: shift by 1 pixel • 2: shift by 2 pixels • 3: shift by 3 pixels
CURRLINE0	[24]	Indicates if currently line 0 is enabled, connected to OUT(0) • 1: enabled
CURRLINE1	[25]	Indicates if currently line 1 is enabled, connected to OUT(4) • 1: enabled
CURRLINE2	[26]	Indicates if currently line 2 is enabled, connected to OUT(8) • 1: enabled

1.1.24 *I_RUN*

I_RUN
0

Reset value: 0

Description:

Internal Register: Core is running Is identical to H_STATUS.RUN

1.1.25 *I_SHIFT_X*

I_SHIFT_X
2 0

Reset value: 1

Description:

Internal Register: Identical to H_CURRXCFG.SHIFT. Increment XPOS counter by positions per "done ,*x"

1.1.26 *I_START*

	CURR		NEXT
31 24	23 16	15 8	7 0

Description:

Internal Register: Identical to H_START

Fields:

Field	Range	Description
NEXT	[7,0]	start address for next line to be processed (read/write)
CURR	[23,16]	Instruction start address for current line being processed (read only)

1.1.27 *I_VFLAG_C*

I_VFLAG_C
3 0

Description:

Internal Combinatorial Signal: 4-way SIMD ALU Carry Flags

1.1.28 *I_VFLAG_L*

I_VFLAG_L
3 0

Description:

Internal Combinatorial Signal: 4-way SIMD ALU Clip Lower Boundary Flags

1.1.29 *I_VFLAG_N*

I_VFLAG_N
3 0

Description:

Internal Combinatorial Signal: 4-way SIMD ALU Negative Flags

1.1.30 *I_VFLAG_O*

I_VFLAG_O
3 0

Description:

Internal Combinatorial Signal: 4-way SIMD ALU Overflow Flags

1.1.31 *I_VFLAG_U*

I_VFLAG_U
3 0

Description:

Internal Combinatorial Signal: 4-way SIMD ALU Clip Upper Boundary Flags

1.1.32 *I_VFLAG_Z*

I_VFLAG_Z
3 0

Description:

Internal Combinatorial Signal: 4-way SIMD ALU Zero Flags

1.1.33 *I_XSIZE*

I_XSIZE
15 0

Reset value: 1024

Description:

Internal Register: Identical to H_CURRXCFG.XSIZE. Stop processing when XPOS reaches H_CURRXCFG.XSIZE (line length)

1.1.34 *LOCK*

GPR28TO31	GPR24TO27	GPR20TO23	GPR16TO19	GPR12TO15	GPR8TO11	GPR4TO7	GPR0TO3
15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0

Reset value: 0xffff

Description:

Core Register: Lock Data Registers for read out via Host register interface (and debug). This is a security feature. The idea is that the kernel itself controls which registers contain result global data to be read by the application SW. All bits of this registers are asserted when a new line is started for processing.

Fields:

Field	Range	Description
GPR0TO3	[1,0]	Lock status for GPR0-GPR3

Field	Range	Description
		• 0: allow read access to GPR0-GPR3 • 1: allow read access to GPR2-GPR3 • 2: allow read access to GPR0 • 3: allow read access to none of the registers GPR0-GPR3 (default)
GPR4TO7	[3,2]	Lock status for GPR4-GPR7 • 0: allow read access to GPR4-GPR7 • 1: allow read access to GPR6-GPR7 • 2: allow read access to GPR4 • 3: allow read access to none of the registers GPR4-GPR7 (default)
GPR8TO11	[5,4]	Lock status for GPR8-GPR11 • 0: allow read access to GPR8 -GPR11 • 1: allow read access to GPR10-GPR11 • 2: allow read access to GPR8 • 3: allow read access to none of the registers GPR8-GPR11 (default)
GPR12TO15	[7,6]	Lock status for GPR12-GPR15 • 0: allow read access to GPR12-GPR15 • 1: allow read access to GPR14-GPR15 • 2: allow read access to GPR12 • 3: allow read access to none of the registers GPR4-GPR7 (default)
GPR16TO19	[9,8]	Lock status for GPR16-GPR19 • 0: allow read access to GPR16-GPR19 • 1: allow read access to GPR18-GPR19 • 2: allow read access to GPR16 • 3: allow read access to none of the registers GPR16-GPR19 (default)
GPR20TO23	[11,10]	Lock status for GPR20-GPR23 • 0: allow read access to GPR20-GPR23 • 1: allow read access to GPR22-GPR23 • 2: allow read access to GPR20 • 3: allow read access to none of the registers GPR20-GP23 (default)
GPR24TO27	[13,12]	Lock status for GPR24 -GPR27 • 0: allow read access to GPR24 -GPR27 • 1: allow read access to GPR26-GPR27 • 2: allow read access to GPR24 • 3: allow read access to none of the registers GPR24-GPR27 (default)
GPR28TO31	[15,14]	Lock status for GPR28-GPR31 • 0: allow read access to GPR28-GPR31 • 1: allow read access to GPR30-GPR31 • 2: allow read access to GPR28 • 3: allow read access to none of the registers GPR28-GPR31 (default)

1.1.35 *LOOPCNT*

LOOPCNT	
15	0

Description:

Core Register: Loop counter

1.1.36 *LOOPCNT1*

LOOPCNT1	
15	0

Description:

Core Register: 2nd Loop counter

1.1.37 *MASKV*

MASKV	
3	0

Reset value: 0xf

Description:

Core Register: Mask for Enabling the way of 4-way SIMD ALU (write only)

- MASK(i) = 1: enable storing result for way i of the 4-way SIMD ALU instruction; default is 1 set at the beginning of a line

1.1.38 *MASKVQ*

MASKVQ	
31	0

Description:

Core Register: Mask for enabling the way of 4-way SIMD ALU (write only; inverted MASKV)

- MASK(i) = 0: enable storing result for way i of the 4-way SIMD ALU instruction

1.1.39 *NIA*

NIA	
7	0

Description:

Internal Signal: Next instruction byte address. The two LSBs are alway zero.

1.1.40 *ONE*

ONE	
15	0

Reset value: 0xffff

Description:

Core Register: -1 constant, read only

1.1.41 *OP1*

OP1	
16	0

Description:

Internal Signal: ALU Operand 1

1.1.42 *OP2*

OP2	
16	0

Description:

Internal Signal: ALU Operand 2

1.1.43 *OPIX*

OPIX	
15	0

Description:

Core Register: Indirect operand register.

1.1.44 *OPIXA*

OPIXA	
7	0

Description:

Core Register: Indirect operand address register.

1.1.45 *RES*

RES	
17	0

Description:

Internal Signal: ALU Result value before saturating

1.1.46 *RES32*

RES32	
32	0

Description:

Internal Signal: Result from multiplier

1.1.47 *SAT*

SAT	
15	0

Description:

Internal Signal: ALU Result value after saturating

1.1.48 *SIG_LINE_DONE*

SIG_LINE_DONE
0

Description:

Output Signal: Line Done. Asserts for one cycle when a line is done

1.1.49 *SRES*

SRES	
19	0

Description:

Internal Signal: Scalar result of 4-way SIMD ALU before saturation

1.1.50 *SSAT*

SSAT	
15	0

Description:

Internal Signal: Scalar result of 4-way SIMD ALU after saturation

1.1.51 *VFLAG_C*

VFLAG_C	
3	0

Description:

Core Register: 4-way SIMD ALU Carry Flags

1.1.52 *VFLAG_N*

VFLAG_N	
3	0

Description:

Core Register: 4-way SIMD ALU Negative Flags

1.1.53 *VFLAG_O*

VFLAG_O	
3	0

Description:

Core Register: 4-way SIMD ALU Overflow Flags

1.1.54 *VFLAG_Z*

VFLAG_Z	
3	0

Description:

Core Register: 4-way SIMD ALU Zero Flags

1.1.55 *VONE*

VONE	
63	0

Reset value: 0xFFFFFFFFFFFFFFFF

Description:

Core Register: -1 read vector register

1.1.56 *VOP1*

VOP1	
63	0

Description:

Internal Signal: Vector operand 1 input to 4-way SIMND ALU

1.1.57 *VOP2*

VOP2	
63	0

Description:

Internal Signal: Vector operand 2 input to 4-way SIMND ALU

1.1.58 *VRES*

VRES	
67	0

Description:

Internal Signal: Vector result of 4-way SIMD ALU before saturation

1.1.59 *VSAT*

VSAT	
67	0

Description:

Internal Signal: Vector result of 4-way SIMD ALU after saturation

1.1.60 *VZERO*

VZERO	
63	0

Reset value: 0

Description:

Core Register: Zero read vector register

1.1.61 *XPOS*

XPOS	
15	0

Description:

Core Register: X position counter This is updated by a Done type of instruction

1.1.62 *YPOS*

YPOS	
15	0

Description:

Core and Host Register: Y position register This is set by the host CPU

1.1.63 *ZERO*

ZERO	
15	0

Description:

Core Register: 0 constant, read only

1.2 Register Files

1.2.1 *ABS*

ABS	
15	0

Number of registers in file: 4

Description:

Core registers: Absolute Value Result of the Abs Angle Acceleration (read only)

Approximated with using a max between (S)ACC(ix) and (S)ACC(ix+4)

1.2.2 *ACC*

ACC	
15	0

Number of registers in file: 8

Description:

Core Register: Accumulator registers Writing accumulates to the registers according to CONFALU.SGN. Reading returns the unsigned value according to CONFALU.SAT and CONFALU.ACCSHR.

1.2.3 *ANGLE*

ANGLE	
15	0

Number of registers in file: 4

Description:

Core registers: Angle Value Result of the Abs Angle Acceleration (read only)

Angle approximates arc_tan(ACC(ix),ACC(ix+4)). For the gardients this assumes, e.g., the X-gradient being stored in ACC0 and the Y-gradient in ACC(4). The Angle value is quantized to 16 directions signed mode or 4 diretcion in usnigned mode

1.2.4 *GPR*

GPR	
15	0

Number of registers in file: 32

Description:

Core Register: General purpose registers

1.2.5 *H_IPS*

H_IPS	
31	0

Number of registers in file: 192

Description:

IPS peripheral register (Host Registers) address mapping

Constituent Registers

Index	Register Usage
0	H_START
1	H_STATUS
2	H_POS
3	H_CURRPOS
4	H_XCFG
5	H_CURRXCFG
6	H_INCFG

Index	Register Usage
9	H_OUTCFG
16	H_IMEMA
17	H_IMEMD
18	H_DREGA
19	H_DREGD
57	IADDR
58	CC_ZNCO
65	S_LINELEN_IN0
66	S_CHCFG_IN0
67	S_LINE_PTR_IN0
68	S_LINELEN_IN1
69	S_CHCFG_IN1
70	S_LINE_PTR_IN1
71	S_LINELEN_IN2
72	S_CHCFG_IN2
73	S_LINE_PTR_IN2
81	S_LINELEN_IN3
82	S_CHCFG_IN3
83	S_LINE_PTR_IN3
84	S_LINELEN_IN4
85	S_CHCFG_IN4
86	S_LINE_PTR_IN4
113	S_LINELEN_OUT0
114	S_CHCFG_OUT0
115	S_LINE_PTR_OUT0
116	S_LINELEN_OUT1
117	S_CHCFG_OUT1
118	S_LINE_PTR_OUT1
119	S_LINELEN_OUT2
120	S_CHCFG_OUT2
121	S_LINE_PTR_OUT2
129	S_CURRLINELEN_IN0
130	S_CURRCHCFG_IN0
131	S_CURRLINE_PTR_IN0
132	S_CURRLINELEN_IN1
133	S_CURRCHCFG_IN1
134	S_CURRLINE_PTR_IN1
135	S_CURRLINELEN_IN2
136	S_CURRCHCFG_IN2
137	S_CURRLINE_PTR_IN2
145	S_CURRLINELEN_IN3
146	S_CURRCHCFG_IN3
147	S_CURRLINE_PTR_IN3
148	S_CURRLINELEN_IN4
149	S_CURRCHCFG_IN4
150	S_CURRLINE_PTR_IN4
177	S_CURRLINELEN_OUT0
178	S_CURRCHCFG_OUT0
179	S_CURRLINE_PTR_OUT0
180	S_CURRLINELEN_OUT1
181	S_CURRCHCFG_OUT1
182	S_CURRLINE_PTR_OUT1
183	S_CURRLINELEN_OUT2
184	S_CURRCHCFG_OUT2
185	S_CURRLINE_PTR_OUT2

1.2.6 *IN*

IN	
15	0

Number of registers in file: 40

Description:

Core Register: Input data register for 8x5 pixels

- IN[7] IN[6] IN[5] IN[4] IN[3] IN[2] IN[1] IN[0]
- IN[15] IN[14] IN[13] IN[12] IN[11] IN[10] IN[9] IN[8]
- IN[23] IN[22] IN[21] IN[20] IN[19] IN[18] IN[17] IN[16]
- IN[31] IN[30] IN[29] IN[28] IN[27] IN[26] IN[25] IN[24]
- IN[39] IN[38] IN[37] IN[36] IN[35] IN[34] IN[33] IN[32]
- (numbering for scan from left to right)

1.2.7 *I_POS_IN*

I_POS_IN	
17	0

Number of registers in file: 5

Description:

Internal Register: Current pixel position in the line

1.2.8 *I_POS_OUT*

I_POS_OUT	
17	0

Number of registers in file: 3

Description:

Internal Register: Current byte position in the line

1.2.9 *I_RPTCNT_IN*

I_RPTCNT_IN	
1	0

Number of registers in file: 5

Reset value: 1

Description:

Internal Register: counting the repeat status

On a read

```
func ( unsigned line_no ) {
 if ( I_RPTCNT_IN ( line_no ) == 0 ) return 4 ;
 return I_RPTCNT_IN ( line_no ) ;
}
```

1.2.10 *I_SKIPCNT_OUT*

I_SKIPCNT_OUT	
1	0

Number of registers in file: 5

Reset value: 0

Description:

> Internal Register: counting the skip status

1.2.11 *O*

O	
15	0

Number of registers in file: 3

Description:

> Internal Register: StreamDMA output interface. Once data is moved from OUT, the StreamDMA engine will autonomously write the register back to the memory (buffering 64 bits).

1.2.12 *OUT*

OUT	
15	0

Number of registers in file: 12

Description:

> Core Register: Output pixel registers.

1.2.13 *P*

P	
15	0

Number of registers in file: 5

Description:

> Internal Register: Prefetch Register for IN StreamDMA input. Once data is moved to IN, the StreamDMA engine will autonomously fill the register with the next value from the line.

1.2.14 *R*

R	
15	0

Number of registers in file: 256

Description:

> Core Register: All registers

Constituent Registers

Index	**Register Usage**

Index	Register Usage
0	LOOPCNT1
1	LOOPCNT
2	XPOS
3	YPOS
4	CONFALU
8	CONFAAG
16	IN0
17	IN1
18	IN2
19	IN3
20	IN4
21	IN5
22	IN6
23	IN7
24	IN8
25	IN9
26	IN10
27	IN11
28	IN12
29	IN13
30	IN14
31	IN15
32	IN16
33	IN17
34	IN18
35	IN19
36	IN20
37	IN21
38	IN22
39	IN23
40	IN24
41	IN25
42	IN26
43	IN27
44	IN28
45	IN29
46	IN30
47	IN31
48	IN32
49	IN33
50	IN34
51	IN35
52	IN36
53	IN37
54	IN38
55	IN39
56	MASKV
57	MASKVQ
58	VFLAG_O
59	VFLAG_C
60	VFLAG_N
61	VFLAG_Z
64	OUT0
65	OUT1
66	OUT2
67	OUT3
68	OUT4
69	OUT5
70	OUT6
71	OUT7
72	OUT8
73	OUT9
74	OUT10
75	OUT11

Index	Register Usage
80	GPR0
81	GPR1
82	GPR2
83	GPR3
84	GPR4
85	GPR5
86	GPR6
87	GPR7
88	GPR8
89	GPR9
90	GPR10
91	GPR11
92	GPR12
93	GPR13
94	GPR14
95	GPR15
96	GPR16
97	GPR17
98	GPR18
99	GPR19
100	GPR20
101	GPR21
102	GPR22
103	GPR23
104	GPR24
105	GPR25
106	GPR26
107	GPR27
108	GPR28
109	GPR29
110	GPR30
111	GPR31
112	ACC0
113	ACC1
114	ACC2
115	ACC3
116	ACC4
117	ACC5
118	ACC6
119	ACC7
128	SACC0
129	SACC1
130	SACC2
131	SACC3
132	SACC4
133	SACC5
134	SACC6
135	SACC7
144	OPIXA
145	OPIX
160	ABS0
161	ABS1
162	ABS2
163	ABS3
164	ANGLE0
165	ANGLE1
166	ANGLE2
167	ANGLE3
252	LOCK
254	ONE
255	ZERO

1.2.15 *SACC*

SACC	
15	0

Number of registers in file: 8

Description:

> Core Register: Set accumulator registers Writing sets the corresponding ACC registers (according to CONFALU.SGN. Reading returns the unsigned value according to CONFALU.SAT and CONFALU.ACCSHR.

1.2.16 *S_CHCFG_IN*

	BUFFERTYPE		REPLACEWIDTH	REVERSE	REPLACEMODE	RPT	STEP	REPLACEVALUE
31 30	29 28	27	26 24	23	22 20	19 18	17 16	15 0

Number of registers in file: 5

Reset value: 0x50000

Description:

> StreamIF Host Register: Configuration for input streams of next line to be processed

Fields:

Field	Range	Description
REPLACEVALUE	[15,0]	Value to use for pixels outside the valid image area
STEP	[17,16]	Offset between two pixels being presented to the IPU engine • 0: 4 pixels offset • 1: 1 pixel offset (no pixel is skipped, all pixels are exercised) • 2: 2 pixels offset • 3: 3 pixels offset
RPT	[19,18]	How often a pixel is presented to the IPU engine • 0: 4 times • 1: 1 time (increment each pixel) • 2: 2 times • 3: 3 times
REPLACEMODE	[22,20]	Scheme for which pixels are getting the REPLAVEVALUE • 0: outside pixel = REPLACEVALUE, inside pixel = pixel value • 1: outside pixel = copy pixel from boundary, inside pixels = pixel value • 2: reserved • 3: outside pixel = REPLACEVALUE, inside pixels = REPLACEVALUE • 4: outside pixel = 0 , inside pixels = REPLACEVALUE
REVERSE	[23]	Do reverse scan of the line • 0: normal scan (left to right) • 1: reverse scan (right to left)
REPLACEWIDTH	[26,24]	How many pixels are assumed at the beginning of the line outside the image. This might be used for filters with horizontal neighborhood. Example: • 0: the image is not extended at the beginning of the line (first pixels is valid) • 2: will present two more pixels at the beginning of the line. The 3rd pixel is the first valid pixel. The two first pixels are treated according to the REPLACEMODE
BUFFERTYPE	[29,28]	Data type of memory buffer. The IPU will always get 16 bit values. • 0: 16-bit

Field	Range	Description
		• 1: 8-bit, data is put to MSBs • 2: 8-bit, data is put to LSBs

1.2.17 *S_CHCFG_OUT*

	BUFFERTYPE		SKIP	
31 30	29 28	27 18	17 16	15 0

Number of registers in file: 3

Reset value: 0x00000

Description:

StreamIF Host Register: Configuration for output streams

Fields:

Field	Range	Description
SKIP	[17,16]	Skip pixel when writing this channel SKIP times • 0: 0 pixels (store evrey pixel) • 1: 1 pixel (store every second pixel) • 2: 2 pixels (store every 3rd pixel) • 3: 3 pixels (store every 4th pixel)
BUFFERTYPE	[29,28]	Data type of memory buffer. • 0: 16-bit • 1: 8-bit, data from MSBs • 2: 8-bit, data from LSBs

1.2.18 *S_CURRCHCFG_IN*

	BUFFERTYPE		REPLACEWIDTH	REVERSE	REPLACEMODE	RPT	STEP	REPLAVEVALUE
31 30	29 28	27	26 24	23	22 20	19 18	17 16	15 0

Number of registers in file: 5

Description:

StreamIF Host Register: Active configuration status for input streams (read onyl)

Fields:

Field	Range	Description
REPLAVEVALUE	[15,0]	Value to use for pixels outside the valid image area
STEP	[17,16]	Offset between two pixels being presented to the IPU engine • 0: 4 pixels offset • 1: 1 pixel offset (no pixel is skipped, all pixels are exercised) • 2: 2 pixels offset • 3: 3 pixels offset
RPT	[19,18]	How often a pixel is presented to the IPU engine • 0: 4 times • 1: 1 time (increment each pixel) • 2: 2 times • 3: 3 times
REPLACEMODE	[22,20]	Scheme for which pixels are getting the REPLAVEVALUE • 0: outside pixel = REPLACEVALUE, inside pixel = pixel value

Field	Range	Description
		• 1: outside pixel = copy pixel from boundary, inside pixels = pixel value • 2: reserved • 3: outside pixel = REPLACEVALUE, inside pixels = REPLACEVALUE • 4: outside pixel = 0 , inside pixels = REPLACEVALUE
REVERSE	[23]	Do reverse scan of the line • 0: normal scan (left to right) • 1: reverse scan (right to left)
REPLACEWIDTH	[26,24]	How many pixels are assumed at the beginning of the line outside the image. This might be used for filters with horizontal neighborhood. Example: • 0: the image is not extended at the beginning of the line (first pixels is valid) • 2: will present two more pixels at the beginning of the line. The 3rd pixel is the first valid pixel. The two first pixels are treated according to the REPLACEMODE
BUFFERTYPE	[29,28]	Data type of memory buffer. The IPU will always get 16 bit values. • 0: 16-bit • 1: 8-bit, data is put to MSBs • 2: 8-bit, data is put to LSBs

1.2.19 S_CURRCHCFG_OUT

	BUFFERTYPE		SKIP	
31 30	29 28	27 18	17 16	15 0

Number of registers in file: 3

Description:

StreamIF Host Register: Active configuration for output streams

Fields:

Field	Range	Description
SKIP	[17,16]	Skip pixel when writing this channel SKIP times • 0: 0 pixels (store evrey pixel) • 1: 1 pixel (store every second pixel) • 2: 2 pixels (store every 3rd pixel) • 3: 3 pixels (store every 4th pixel)
BUFFERTYPE	[29,28]	Data type of memory buffer. • 0: 16-bit • 1: 8-bit, data from MSBs • 2: 8-bit, data from LSBs

1.2.20 S_CURRLINELEN_IN

S_CURRLINELEN_IN
15 0

Number of registers in file: 5

Reset value: 0xFFFF

Description:

StreamIF Host Register: Active line length in pixels (read only)

1.2.21 *S_CURRLINELEN_OUT*

S_CURRLINELEN_OUT	
15	0

Number of registers in file: 3

Reset value: 0xFFFF

Description:

StreamIF Host Register: Active line length in pixels (read only)

1.2.22 *S_CURRLINE_PTR_IN*

S_CURRLINE_PTR_IN	
31	0

Number of registers in file: 5

Description:

StreamIF Host Register: Active Line Start Addresses (read only)

1.2.23 *S_CURRLINE_PTR_OUT*

S_CURRLINE_PTR_OUT	
31	0

Number of registers in file: 3

Description:

StreamIF Host Register: Active Line Start Addresses

On a write

```
func ( unsigned line_no , bits < 32 > val ) {
 S_CURRLINE_PTR_OUT ( line_no ) = val . uint32 (   ) & 0xfffffff8 ;
 if ( val ( 2 , 0 ) != 0 ) {
     error ( 1 , "\n*******************\n" , "ERROR: unaligned address OUT_PTR(" , line_no , ") = "
 }
}
```

1.2.24 *S_LINELEN_IN*

S_LINELEN_IN	
15	0

Number of registers in file: 5

Reset value: 0xFFFF

Description:

StreamIF Host Register: Line length in pixels

1.2.25 *S_LINELEN_OUT*

S_LINELEN_OUT

15	0

Number of registers in file: 3

Reset value: 0xFFFF

Description:

StreamIF Host Register: Line length in pixels

1.2.26 *S_LINE_PTR_IN*

S_LINE_PTR_IN	
31	0

Number of registers in file: 5

Description:

StreamIF Host Register: Line Start Addresses

1.2.27 *S_LINE_PTR_OUT*

S_LINE_PTR_OUT	
31	0

Number of registers in file: 3

Description:

StreamIF Host Register: Line Start Addresses

1.2.28 *VABS*

VABS	
67	0

Number of registers in file: 1

Description:

Core Register: Absolute vector register pointing to ABS registers

- VABS = (ABS[0] ,ABS[1] ,ABS[2] ,ABS[3])

1.2.29 *VACC*

VACC	
67	0

Number of registers in file: 2

Description:

Core Register: Accumulator vector registers pinting to ACC registers

- VACC[0] = (ACC[0] ,ACC[1] ,ACC[2] ,ACC[3])
- VACC[1] = (ACC[4] ,ACC[5] ,ACC[6] ,ACC[7])

1.2.30 *VANGLE*

VANGLE	
67	0

Number of registers in file: 1

Description:

Core Register: Angle vector register pointing to ANGLE registers (read only)

- VANGLE = (ANGLE[0] ,ANGLE[1] ,ANGLE[2] ,ANGLE[3])

1.2.31 *VGPR*

VGPR	
67	0

Number of registers in file: 8

Description:

Core Register: General purpose vector registers pointing to GPR registers

- VGPR[0] = (GPR[0] ,GPR[1] ,GPR[2] ,GPR[3])
- VGPR[1] = (GPR[4] ,GPR[5] ,GPR[6] ,GPR[7])
- VGPR[2] = (GPR[8] ,GPR[9] ,GPR[10],GPR[11])
- VGPR[3] = (GPR[12],GPR[13],GPR[14],GPR[15])
- VGPR[4] = (GPR[16],GPR[17],GPR[18],GPR[19])
- VGPR[5] = (GPR[20],GPR[21],GPR[22],GPR[23])
- VGPR[6] = (GPR[24],GPR[25],GPR[26],GPR[27])
- VGPR[7] = (GPR[28],GPR[29],GPR[30],GPR[31])

1.2.32 *VH*

VH	
67	0

Number of registers in file: 25

Description:

Core Register: Horizontal vector registers pointing to IN registers

e.g., VH[10] = (IN[19],IN[18],IN[17],IN[18])

- ○ ■ ■ VH[4] VH[3] VH[2] VH[1] VH[0]
- ○ ■ ■ VH[9] VH[8] VH[7] VH[6] VH[5]
- ○ ■ ■ VH[14] VH[13] VH[12] VH[11] VH[10]
- ○ ■ ■ VH[19] VH[18] VH[17] VH[16] VH[15]
- ○ ■ ■ VH[24] VH[23] VH[22] VH[21] VH[20]
- (numbering for scan from left to right)

1.2.33 *VOUT*

VOUT	
67	0

Number of registers in file: 3

Description:

Core Register: Output vector registers pointing to OUT registers

- VOUT[0] = (OUT[0],OUT[1],OUT[2] ,OUT[3])
- VOUT[1] = (OUT[4],OUT[5],OUT[6] ,OUT[7])

- VOUT[2] = (OUT[8],OUT[9],OUT[10],OUT[11])

1.2.34 *VR*

VR	
67	0

Number of registers in file: 256

Description:

Core Register: All vector registers

Constituent Registers

Index	Register Usage
0	VV0
1	VV1
2	VV2
3	VV3
4	VV4
5	VV5
6	VV6
7	VV7
8	VV8
9	VV9
10	VV10
11	VV11
12	VV12
13	VV13
14	VV14
15	VV15
16	VH0
17	VH1
18	VH2
19	VH3
20	VH4
21	VH5
22	VH6
23	VH7
24	VH8
25	VH9
26	VH10
27	VH11
28	VH12
29	VH13
30	VH14
31	VH15
32	VH16
33	VH17
34	VH18
35	VH19
36	VH20
37	VH21
38	VH22
39	VH23
40	VH24
45	VOUT0
46	VOUT1
47	VOUT2
48	VGPR0
49	VGPR1
50	VGPR2
51	VGPR3
52	VGPR4

Index	Register Usage
53	VGPR5
54	VGPR6
55	VGPR7
56	VACC0
57	VACC1
58	VSACC0
59	VSACC1
60	VABS0
61	VANGLE0
254	VONE
255	VZERO

1.2.35 *VSACC*

VSACC	
67	0

Number of registers in file: 2

Description:

Core Register: Set accumulator vector register pointing to ACC registers

- VSACC[0] = (SACC[0] ,SACC[1] ,SACC[2] ,SACC[3])
- VSACC[1] = (SACC[4] ,SACC[5] ,SACC[6] ,SACC[7])

1.2.36 *VV*

VV	
67	0

Number of registers in file: 16

Description:

Core Register: Vertical vector registers point to IN registers

e.g., VV[11] = (IN[11],IN[19],IN[27],IN[35])

- VV[7] VV[6] VV[5] VV[4] VV[3] VV[2] VV[1] VV[0]
- VV[15] VV[14] VV[13] VV[12] VV[11] VV[10] VV[9] VV[8]
- ○ ■ ■ ■ ■ ■ ■ ■
- ○ ■ ■ ■ ■ ■ ■ ■
- ○ ■ ■ ■ ■ ■ ■ ■
- (numbering for scan from left to right)

1.3 Instruction Fields

C D I L O R T V

1.3.1 *CLZBITS*

Size	3
Bits	[6,4]

Description:

CLZ vs. LSL bit

1.3.2 *COND*

Size	5
Bits	[4,0]

Description:

Branch Condition

Enumeration	**Value**
AL	0
NV	1
EQ	2
NE	3
CC	4
CS	5
MI	6
PL	7
OS	8
OC	9
GE	10
LT	11
HI	12
LS	13
GT	14

1.3.3 *DEST*

Size	8
Bits	[23,16]

Description:

Scalar destination register

Enumeration	**Value**
loopcntb	0
loopcnt	1
xpos	2
ypos	3
confalu	4
reserved	5
reserved	6
reserved	7
confaag	8
reserved	9
reserved	10
reserved	11
reserved	12
reserved	13
reserved	14
reserved	15
in0	16
in1	17
in2	18
in3	19
in4	20
in5	21
in6	22
in7	23
in8	24
in9	25
in10	26
in11	27
in12	28

in13	29
in14	30
in15	31
in16	32
in17	33
in18	34
in19	35
in20	36
in21	37
in22	38
in23	39
in24	40
in25	41
in26	42
in27	43
in28	44
in29	45
in30	46
in31	47
in32	48
in33	49
in34	50
in35	51
in36	52
in37	53
in38	54
in39	55
maskv	56
maskvq	57
vflag_o	58
vflag_c	59
vflag_n	60
vflag_z	61
reserved	62
reserved	63
out0	64
out1	65
out2	66
out3	67
out4	68
out5	69
out6	70
out7	71
out8	72
out9	73
out10	74
out11	75
reserved	76
reserved	77
reserved	78
reserved	79
gpr0	80
gpr1	81
gpr2	82
gpr3	83
gpr4	84
gpr5	85
gpr6	86
gpr7	87
gpr8	88
gpr9	89
gpr10	90
gpr11	91
gpr12	92
gpr13	93
gpr14	94
gpr15	95
gpr16	96

gpr17	97
gpr18	98
gpr19	99
gpr20	100
gpr21	101
gpr22	102
gpr23	103
gpr24	104
gpr25	105
gpr26	106
gpr27	107
gpr28	108
gpr29	109
gpr30	110
gpr31	111
acc0	112
acc1	113
acc2	114
acc3	115
acc4	116
acc5	117
acc6	118
acc7	119
reserved	120
reserved	121
reserved	122
reserved	123
reserved	124
reserved	125
reserved	126
reserved	127
sacc0	128
sacc1	129
sacc2	130
sacc3	131
sacc4	132
sacc5	133
sacc6	134
sacc7	135
reserved	136
reserved	137
reserved	138
reserved	139
reserved	140
reserved	141
reserved	142
reserved	143
opixa	144
opix	145
reserved	146
reserved	147
reserved	148
reserved	149
reserved	150
reserved	151
reserved	152
reserved	153
reserved	154
reserved	155
reserved	156
reserved	157
reserved	158
reserved	159
abs0	160
abs1	161
abs2	162
abs3	163
angle0	164

angle1	165
angle2	166
angle3	167
reserved	168
reserved	169
reserved	170
reserved	171
reserved	172
reserved	173
reserved	174
reserved	175
reserved	176
reserved	177
reserved	178
reserved	179
reserved	180
reserved	181
reserved	182
reserved	183
reserved	184
reserved	185
reserved	186
reserved	187
reserved	188
reserved	189
reserved	190
reserved	191
reserved	192
reserved	193
reserved	194
reserved	195
reserved	196
reserved	197
reserved	198
reserved	199
reserved	200
reserved	201
reserved	202
reserved	203
reserved	204
reserved	205
reserved	206
reserved	207
reserved	208
reserved	209
reserved	210
reserved	211
reserved	212
reserved	213
reserved	214
reserved	215
reserved	216
reserved	217
reserved	218
reserved	219
reserved	220
reserved	221
reserved	222
reserved	223
reserved	224
reserved	225
reserved	226
reserved	227
reserved	228
reserved	229
reserved	230
reserved	231
reserved	232

reserved	233
reserved	234
reserved	235
reserved	236
reserved	237
reserved	238
reserved	239
reserved	240
reserved	241
reserved	242
reserved	243
reserved	244
reserved	245
reserved	246
reserved	247
reserved	248
reserved	249
reserved	250
reserved	251
lock	252
reserved	253
one	254
zero	255

1.3.4 *DONE_IN*

Size	8
Bits	[23,16]

Description:

Internal Signal: Done operator register

Enumeration	Value
loopcntb	0
loopcnt	1
xpos	2
ypos	3
confalu	4
reserved	5
reserved	6
reserved	7
confaag	8
reserved	9
reserved	10
reserved	11
reserved	12
reserved	13
reserved	14
reserved	15
in0	16
in1	17
in2	18
in3	19
in4	20
in5	21
in6	22
in7	23
in8	24
in9	25
in10	26
in11	27
in12	28
in13	29
in14	30
in15	31
in16	32

in17	33
in18	34
in19	35
in20	36
in21	37
in22	38
in23	39
in24	40
in25	41
in26	42
in27	43
in28	44
in29	45
in30	46
in31	47
in32	48
in33	49
in34	50
in35	51
in36	52
in37	53
in38	54
in39	55
maskv	56
maskvq	57
vflag_o	58
vflag_c	59
vflag_n	60
vflag_z	61
reserved	62
reserved	63
out0	64
out1	65
out2	66
out3	67
out4	68
out5	69
out6	70
out7	71
out8	72
out9	73
out10	74
out11	75
reserved	76
reserved	77
reserved	78
reserved	79
gpr0	80
gpr1	81
gpr2	82
gpr3	83
gpr4	84
gpr5	85
gpr6	86
gpr7	87
gpr8	88
gpr9	89
gpr10	90
gpr11	91
gpr12	92
gpr13	93
gpr14	94
gpr15	95
gpr16	96
gpr17	97
gpr18	98
gpr19	99
gpr20	100

gpr21	101
gpr22	102
gpr23	103
gpr24	104
gpr25	105
gpr26	106
gpr27	107
gpr28	108
gpr29	109
gpr30	110
gpr31	111
acc0	112
acc1	113
acc2	114
acc3	115
acc4	116
acc5	117
acc6	118
acc7	119
reserved	120
reserved	121
reserved	122
reserved	123
reserved	124
reserved	125
reserved	126
reserved	127
sacc0	128
sacc1	129
sacc2	130
sacc3	131
sacc4	132
sacc5	133
sacc6	134
sacc7	135
reserved	136
reserved	137
reserved	138
reserved	139
reserved	140
reserved	141
reserved	142
reserved	143
opixa	144
opix	145
reserved	146
reserved	147
reserved	148
reserved	149
reserved	150
reserved	151
reserved	152
reserved	153
reserved	154
reserved	155
reserved	156
reserved	157
reserved	158
reserved	159
abs0	160
abs1	161
abs2	162
abs3	163
angle0	164
angle1	165
angle2	166
angle3	167
reserved	168

reserved	169
reserved	170
reserved	171
reserved	172
reserved	173
reserved	174
reserved	175
reserved	176
reserved	177
reserved	178
reserved	179
reserved	180
reserved	181
reserved	182
reserved	183
reserved	184
reserved	185
reserved	186
reserved	187
reserved	188
reserved	189
reserved	190
reserved	191
reserved	192
reserved	193
reserved	194
reserved	195
reserved	196
reserved	197
reserved	198
reserved	199
reserved	200
reserved	201
reserved	202
reserved	203
reserved	204
reserved	205
reserved	206
reserved	207
reserved	208
reserved	209
reserved	210
reserved	211
reserved	212
reserved	213
reserved	214
reserved	215
reserved	216
reserved	217
reserved	218
reserved	219
reserved	220
reserved	221
reserved	222
reserved	223
reserved	224
reserved	225
reserved	226
reserved	227
reserved	228
reserved	229
reserved	230
reserved	231
reserved	232
reserved	233
reserved	234
reserved	235
reserved	236

reserved	237
reserved	238
reserved	239
reserved	240
reserved	241
reserved	242
reserved	243
reserved	244
reserved	245
reserved	246
reserved	247
reserved	248
reserved	249
reserved	250
reserved	251
lock	252
reserved	253
one	254
zero	255

1.3.5 *DONE_VIN*

Size	8
Bits	[23,16]

Description:

Internal Signal: Done vector operator register

Enumeration	Value
vv0	0
vv1	1
vv2	2
vv3	3
vv4	4
vv5	5
vv6	6
vv7	7
vv8	8
vv9	9
vv10	10
vv11	11
vv12	12
vv13	13
vv14	14
vv15	15
vh0	16
vh1	17
vh2	18
vh3	19
vh4	20
vh5	21
vh6	22
vh7	23
vh8	24
vh9	25
vh10	26
vh11	27
vh12	28
vh13	29
vh14	30
vh15	31
vh16	32
vh17	33
vh18	34
vh19	35
vh20	36

vh21	37
vh22	38
vh23	39
vh24	40
reserved	41
reserved	42
reserved	43
reserved	44
vout0	45
vout1	46
vout2	47
vgpr0	48
vgpr1	49
vgpr2	50
vgpr3	51
vgpr4	52
vgpr5	53
vgpr6	54
vgpr7	55
vacc0	56
vacc1	57
vsacc0	58
vsacc1	59
vabs	60
vangle	61
reserved	62
reserved	63
reserved	64
reserved	65
reserved	66
reserved	67
reserved	68
reserved	69
reserved	70
reserved	71
reserved	72
reserved	73
reserved	74
reserved	75
reserved	76
reserved	77
reserved	78
reserved	79
reserved	80
reserved	81
reserved	82
reserved	83
reserved	84
reserved	85
reserved	86
reserved	87
reserved	88
reserved	89
reserved	90
reserved	91
reserved	92
reserved	93
reserved	94
reserved	95
reserved	96
reserved	97
reserved	98
reserved	99
reserved	100
reserved	101
reserved	102
reserved	103
reserved	104

reserved	105
reserved	106
reserved	107
reserved	108
reserved	109
reserved	110
reserved	111
reserved	112
reserved	113
reserved	114
reserved	115
reserved	116
reserved	117
reserved	118
reserved	119
reserved	120
reserved	121
reserved	122
reserved	123
reserved	124
reserved	125
reserved	126
reserved	127
reserved	128
reserved	129
reserved	130
reserved	131
reserved	132
reserved	133
reserved	134
reserved	135
reserved	136
reserved	137
reserved	138
reserved	139
reserved	140
reserved	141
reserved	142
reserved	143
reserved	144
reserved	145
reserved	146
reserved	147
reserved	148
reserved	149
reserved	150
reserved	151
reserved	152
reserved	153
reserved	154
reserved	155
reserved	156
reserved	157
reserved	158
reserved	159
reserved	160
reserved	161
reserved	162
reserved	163
reserved	164
reserved	165
reserved	166
reserved	167
reserved	168
reserved	169
reserved	170
reserved	171
reserved	172

reserved	173
reserved	174
reserved	175
reserved	176
reserved	177
reserved	178
reserved	179
reserved	180
reserved	181
reserved	182
reserved	183
reserved	184
reserved	185
reserved	186
reserved	187
reserved	188
reserved	189
reserved	190
reserved	191
reserved	192
reserved	193
reserved	194
reserved	195
reserved	196
reserved	197
reserved	198
reserved	199
reserved	200
reserved	201
reserved	202
reserved	203
reserved	204
reserved	205
reserved	206
reserved	207
reserved	208
reserved	209
reserved	210
reserved	211
reserved	212
reserved	213
reserved	214
reserved	215
reserved	216
reserved	217
reserved	218
reserved	219
reserved	220
reserved	221
reserved	222
reserved	223
reserved	224
reserved	225
reserved	226
reserved	227
reserved	228
reserved	229
reserved	230
reserved	231
reserved	232
reserved	233
reserved	234
reserved	235
reserved	236
reserved	237
reserved	238
reserved	239
reserved	240

reserved	241
reserved	242
reserved	243
reserved	244
reserved	245
reserved	246
reserved	247
reserved	248
reserved	249
reserved	250
reserved	251
reserved	252
reserved	253
vone	254
vzero	255

1.3.6 *IM16*

Size	16
Bits	[15,0]

Description:

Operator 2 immediate 16 bits

1.3.7 *IM4*

Size	4
Bits	[3,0]

Description:

Operator 2 immediate 4 bits

1.3.8 *IN1*

Size	8
Bits	[15,8]

Description:

Operator 1 register

Enumeration	Value
loopcntb	0
loopcnt	1
xpos	2
ypos	3
confalu	4
reserved	5
reserved	6
reserved	7
confaag	8
reserved	9
reserved	10
reserved	11
reserved	12
reserved	13
reserved	14
reserved	15
in0	16
in1	17
in2	18

in3	19
in4	20
in5	21
in6	22
in7	23
in8	24
in9	25
in10	26
in11	27
in12	28
in13	29
in14	30
in15	31
in16	32
in17	33
in18	34
in19	35
in20	36
in21	37
in22	38
in23	39
in24	40
in25	41
in26	42
in27	43
in28	44
in29	45
in30	46
in31	47
in32	48
in33	49
in34	50
in35	51
in36	52
in37	53
in38	54
in39	55
maskv	56
maskvq	57
vflag_o	58
vflag_c	59
vflag_n	60
vflag_z	61
reserved	62
reserved	63
out0	64
out1	65
out2	66
out3	67
out4	68
out5	69
out6	70
out7	71
out8	72
out9	73
out10	74
out11	75
reserved	76
reserved	77
reserved	78
reserved	79
gpr0	80
gpr1	81
gpr2	82
gpr3	83
gpr4	84
gpr5	85
gpr6	86

gpr7	87
gpr8	88
gpr9	89
gpr10	90
gpr11	91
gpr12	92
gpr13	93
gpr14	94
gpr15	95
gpr16	96
gpr17	97
gpr18	98
gpr19	99
gpr20	100
gpr21	101
gpr22	102
gpr23	103
gpr24	104
gpr25	105
gpr26	106
gpr27	107
gpr28	108
gpr29	109
gpr30	110
gpr31	111
acc0	112
acc1	113
acc2	114
acc3	115
acc4	116
acc5	117
acc6	118
acc7	119
reserved	120
reserved	121
reserved	122
reserved	123
reserved	124
reserved	125
reserved	126
reserved	127
sacc0	128
sacc1	129
sacc2	130
sacc3	131
sacc4	132
sacc5	133
sacc6	134
sacc7	135
reserved	136
reserved	137
reserved	138
reserved	139
reserved	140
reserved	141
reserved	142
reserved	143
opixa	144
opix	145
reserved	146
reserved	147
reserved	148
reserved	149
reserved	150
reserved	151
reserved	152
reserved	153
reserved	154

reserved	155
reserved	156
reserved	157
reserved	158
reserved	159
abs0	160
abs1	161
abs2	162
abs3	163
angle0	164
angle1	165
angle2	166
angle3	167
reserved	168
reserved	169
reserved	170
reserved	171
reserved	172
reserved	173
reserved	174
reserved	175
reserved	176
reserved	177
reserved	178
reserved	179
reserved	180
reserved	181
reserved	182
reserved	183
reserved	184
reserved	185
reserved	186
reserved	187
reserved	188
reserved	189
reserved	190
reserved	191
reserved	192
reserved	193
reserved	194
reserved	195
reserved	196
reserved	197
reserved	198
reserved	199
reserved	200
reserved	201
reserved	202
reserved	203
reserved	204
reserved	205
reserved	206
reserved	207
reserved	208
reserved	209
reserved	210
reserved	211
reserved	212
reserved	213
reserved	214
reserved	215
reserved	216
reserved	217
reserved	218
reserved	219
reserved	220
reserved	221
reserved	222

reserved	223
reserved	224
reserved	225
reserved	226
reserved	227
reserved	228
reserved	229
reserved	230
reserved	231
reserved	232
reserved	233
reserved	234
reserved	235
reserved	236
reserved	237
reserved	238
reserved	239
reserved	240
reserved	241
reserved	242
reserved	243
reserved	244
reserved	245
reserved	246
reserved	247
reserved	248
reserved	249
reserved	250
reserved	251
lock	252
reserved	253
one	254
zero	255

1.3.9 *IN2*

Size	8
Bits	[7,0]

Description:

Operator 2 register

Enumeration	Value
loopcntb	0
loopcnt	1
xpos	2
ypos	3
confalu	4
reserved	5
reserved	6
reserved	7
confaag	8
reserved	9
reserved	10
reserved	11
reserved	12
reserved	13
reserved	14
reserved	15
in0	16
in1	17
in2	18
in3	19
in4	20
in5	21
in6	22

in7	23
in8	24
in9	25
in10	26
in11	27
in12	28
in13	29
in14	30
in15	31
in16	32
in17	33
in18	34
in19	35
in20	36
in21	37
in22	38
in23	39
in24	40
in25	41
in26	42
in27	43
in28	44
in29	45
in30	46
in31	47
in32	48
in33	49
in34	50
in35	51
in36	52
in37	53
in38	54
in39	55
maskv	56
maskvq	57
vflag_o	58
vflag_c	59
vflag_n	60
vflag_z	61
reserved	62
reserved	63
out0	64
out1	65
out2	66
out3	67
out4	68
out5	69
out6	70
out7	71
out8	72
out9	73
out10	74
out11	75
reserved	76
reserved	77
reserved	78
reserved	79
gpr0	80
gpr1	81
gpr2	82
gpr3	83
gpr4	84
gpr5	85
gpr6	86
gpr7	87
gpr8	88
gpr9	89
gpr10	90

gpr11	91
gpr12	92
gpr13	93
gpr14	94
gpr15	95
gpr16	96
gpr17	97
gpr18	98
gpr19	99
gpr20	100
gpr21	101
gpr22	102
gpr23	103
gpr24	104
gpr25	105
gpr26	106
gpr27	107
gpr28	108
gpr29	109
gpr30	110
gpr31	111
acc0	112
acc1	113
acc2	114
acc3	115
acc4	116
acc5	117
acc6	118
acc7	119
reserved	120
reserved	121
reserved	122
reserved	123
reserved	124
reserved	125
reserved	126
reserved	127
sacc0	128
sacc1	129
sacc2	130
sacc3	131
sacc4	132
sacc5	133
sacc6	134
sacc7	135
reserved	136
reserved	137
reserved	138
reserved	139
reserved	140
reserved	141
reserved	142
reserved	143
opixa	144
opix	145
reserved	146
reserved	147
reserved	148
reserved	149
reserved	150
reserved	151
reserved	152
reserved	153
reserved	154
reserved	155
reserved	156
reserved	157
reserved	158

reserved	159
abs0	160
abs1	161
abs2	162
abs3	163
angle0	164
angle1	165
angle2	166
angle3	167
reserved	168
reserved	169
reserved	170
reserved	171
reserved	172
reserved	173
reserved	174
reserved	175
reserved	176
reserved	177
reserved	178
reserved	179
reserved	180
reserved	181
reserved	182
reserved	183
reserved	184
reserved	185
reserved	186
reserved	187
reserved	188
reserved	189
reserved	190
reserved	191
reserved	192
reserved	193
reserved	194
reserved	195
reserved	196
reserved	197
reserved	198
reserved	199
reserved	200
reserved	201
reserved	202
reserved	203
reserved	204
reserved	205
reserved	206
reserved	207
reserved	208
reserved	209
reserved	210
reserved	211
reserved	212
reserved	213
reserved	214
reserved	215
reserved	216
reserved	217
reserved	218
reserved	219
reserved	220
reserved	221
reserved	222
reserved	223
reserved	224
reserved	225
reserved	226

reserved	227
reserved	228
reserved	229
reserved	230
reserved	231
reserved	232
reserved	233
reserved	234
reserved	235
reserved	236
reserved	237
reserved	238
reserved	239
reserved	240
reserved	241
reserved	242
reserved	243
reserved	244
reserved	245
reserved	246
reserved	247
reserved	248
reserved	249
reserved	250
reserved	251
lock	252
reserved	253
one	254
zero	255

1.3.10 *IXO*

Size	3
Bits	[2,0]

Description:

Increment input, xposition, output

Enumeration	Value
-	0
o	1
x	2
xo	3
i	4
io	5
ix	6
ixo	7

1.3.11 *LCNT*

Size	1
Bits	[3,3]

Description:

Loop counter selection

Enumeration	Value
loopcnt1	0
loopcnt	1

1.3.12 *OPCD*

Size	4
Bits	[27,24]

Description:

Extended opcode

1.3.13 *RELADDR*

Size	8
Shift	2
Bits	[15,8]
Addressing	PC-Relative
Signed	True

Description:

Relative branch target address

Semantics: **IADDR + (SignExtend(RELADDR << 2))**

1.3.14 *TYPE*

Size	4
Bits	[31,28]

Description:

Primary instruction type

Enumeration	Value
Reserved8	0
Reserved7	1
Reserved6	2
Reserved5	3
Reserved4	4
Reserved3	5
Reserved2	6
Reserved1	7
Flow	8
S=SxI	9
S=SxS	10
Reserved	11
S=VxS	12
S=VxV	13
V=VxS	14
V=VxV	15

1.3.15 *VDEST*

Size	8
Bits	[23,16]

Description:

Vector destination register

Enumeration	Value
vv0	0
vv1	1
vv2	2
vv3	3

vv4	4
vv5	5
vv6	6
vv7	7
vv8	8
vv9	9
vv10	10
vv11	11
vv12	12
vv13	13
vv14	14
vv15	15
vh0	16
vh1	17
vh2	18
vh3	19
vh4	20
vh5	21
vh6	22
vh7	23
vh8	24
vh9	25
vh10	26
vh11	27
vh12	28
vh13	29
vh14	30
vh15	31
vh16	32
vh17	33
vh18	34
vh19	35
vh20	36
vh21	37
vh22	38
vh23	39
vh24	40
reserved	41
reserved	42
reserved	43
reserved	44
vout0	45
vout1	46
vout2	47
vgpr0	48
vgpr1	49
vgpr2	50
vgpr3	51
vgpr4	52
vgpr5	53
vgpr6	54
vgpr7	55
vacc0	56
vacc1	57
vsacc0	58
vsacc1	59
vabs	60
vangle	61
reserved	62
reserved	63
reserved	64
reserved	65
reserved	66
reserved	67
reserved	68
reserved	69
reserved	70
reserved	71

reserved	72
reserved	73
reserved	74
reserved	75
reserved	76
reserved	77
reserved	78
reserved	79
reserved	80
reserved	81
reserved	82
reserved	83
reserved	84
reserved	85
reserved	86
reserved	87
reserved	88
reserved	89
reserved	90
reserved	91
reserved	92
reserved	93
reserved	94
reserved	95
reserved	96
reserved	97
reserved	98
reserved	99
reserved	100
reserved	101
reserved	102
reserved	103
reserved	104
reserved	105
reserved	106
reserved	107
reserved	108
reserved	109
reserved	110
reserved	111
reserved	112
reserved	113
reserved	114
reserved	115
reserved	116
reserved	117
reserved	118
reserved	119
reserved	120
reserved	121
reserved	122
reserved	123
reserved	124
reserved	125
reserved	126
reserved	127
reserved	128
reserved	129
reserved	130
reserved	131
reserved	132
reserved	133
reserved	134
reserved	135
reserved	136
reserved	137
reserved	138
reserved	139

reserved	140
reserved	141
reserved	142
reserved	143
reserved	144
reserved	145
reserved	146
reserved	147
reserved	148
reserved	149
reserved	150
reserved	151
reserved	152
reserved	153
reserved	154
reserved	155
reserved	156
reserved	157
reserved	158
reserved	159
reserved	160
reserved	161
reserved	162
reserved	163
reserved	164
reserved	165
reserved	166
reserved	167
reserved	168
reserved	169
reserved	170
reserved	171
reserved	172
reserved	173
reserved	174
reserved	175
reserved	176
reserved	177
reserved	178
reserved	179
reserved	180
reserved	181
reserved	182
reserved	183
reserved	184
reserved	185
reserved	186
reserved	187
reserved	188
reserved	189
reserved	190
reserved	191
reserved	192
reserved	193
reserved	194
reserved	195
reserved	196
reserved	197
reserved	198
reserved	199
reserved	200
reserved	201
reserved	202
reserved	203
reserved	204
reserved	205
reserved	206
reserved	207

reserved	208
reserved	209
reserved	210
reserved	211
reserved	212
reserved	213
reserved	214
reserved	215
reserved	216
reserved	217
reserved	218
reserved	219
reserved	220
reserved	221
reserved	222
reserved	223
reserved	224
reserved	225
reserved	226
reserved	227
reserved	228
reserved	229
reserved	230
reserved	231
reserved	232
reserved	233
reserved	234
reserved	235
reserved	236
reserved	237
reserved	238
reserved	239
reserved	240
reserved	241
reserved	242
reserved	243
reserved	244
reserved	245
reserved	246
reserved	247
reserved	248
reserved	249
reserved	250
reserved	251
reserved	252
reserved	253
vone	254
vzero	255

1.3.16 *VIN1*

Size	8
Bits	[15,8]

Description:

Vector Operator 1 register

Enumeration	Value
vv0	0
vv1	1
vv2	2
vv3	3
vv4	4
vv5	5
vv6	6
vv7	7

vv8	8
vv9	9
vv10	10
vv11	11
vv12	12
vv13	13
vv14	14
vv15	15
vh0	16
vh1	17
vh2	18
vh3	19
vh4	20
vh5	21
vh6	22
vh7	23
vh8	24
vh9	25
vh10	26
vh11	27
vh12	28
vh13	29
vh14	30
vh15	31
vh16	32
vh17	33
vh18	34
vh19	35
vh20	36
vh21	37
vh22	38
vh23	39
vh24	40
reserved	41
reserved	42
reserved	43
reserved	44
vout0	45
vout1	46
vout2	47
vgpr0	48
vgpr1	49
vgpr2	50
vgpr3	51
vgpr4	52
vgpr5	53
vgpr6	54
vgpr7	55
vacc0	56
vacc1	57
vsacc0	58
vsacc1	59
vabs	60
vangle	61
reserved	62
reserved	63
reserved	64
reserved	65
reserved	66
reserved	67
reserved	68
reserved	69
reserved	70
reserved	71
reserved	72
reserved	73
reserved	74
reserved	75

reserved	76
reserved	77
reserved	78
reserved	79
reserved	80
reserved	81
reserved	82
reserved	83
reserved	84
reserved	85
reserved	86
reserved	87
reserved	88
reserved	89
reserved	90
reserved	91
reserved	92
reserved	93
reserved	94
reserved	95
reserved	96
reserved	97
reserved	98
reserved	99
reserved	100
reserved	101
reserved	102
reserved	103
reserved	104
reserved	105
reserved	106
reserved	107
reserved	108
reserved	109
reserved	110
reserved	111
reserved	112
reserved	113
reserved	114
reserved	115
reserved	116
reserved	117
reserved	118
reserved	119
reserved	120
reserved	121
reserved	122
reserved	123
reserved	124
reserved	125
reserved	126
reserved	127
reserved	128
reserved	129
reserved	130
reserved	131
reserved	132
reserved	133
reserved	134
reserved	135
reserved	136
reserved	137
reserved	138
reserved	139
reserved	140
reserved	141
reserved	142
reserved	143

reserved	144
reserved	145
reserved	146
reserved	147
reserved	148
reserved	149
reserved	150
reserved	151
reserved	152
reserved	153
reserved	154
reserved	155
reserved	156
reserved	157
reserved	158
reserved	159
reserved	160
reserved	161
reserved	162
reserved	163
reserved	164
reserved	165
reserved	166
reserved	167
reserved	168
reserved	169
reserved	170
reserved	171
reserved	172
reserved	173
reserved	174
reserved	175
reserved	176
reserved	177
reserved	178
reserved	179
reserved	180
reserved	181
reserved	182
reserved	183
reserved	184
reserved	185
reserved	186
reserved	187
reserved	188
reserved	189
reserved	190
reserved	191
reserved	192
reserved	193
reserved	194
reserved	195
reserved	196
reserved	197
reserved	198
reserved	199
reserved	200
reserved	201
reserved	202
reserved	203
reserved	204
reserved	205
reserved	206
reserved	207
reserved	208
reserved	209
reserved	210
reserved	211

reserved	212
reserved	213
reserved	214
reserved	215
reserved	216
reserved	217
reserved	218
reserved	219
reserved	220
reserved	221
reserved	222
reserved	223
reserved	224
reserved	225
reserved	226
reserved	227
reserved	228
reserved	229
reserved	230
reserved	231
reserved	232
reserved	233
reserved	234
reserved	235
reserved	236
reserved	237
reserved	238
reserved	239
reserved	240
reserved	241
reserved	242
reserved	243
reserved	244
reserved	245
reserved	246
reserved	247
reserved	248
reserved	249
reserved	250
reserved	251
reserved	252
reserved	253
vone	254
vzero	255

1.3.17 *VIN2*

Size	8
Bits	[7,0]

Description:

Vector Operator 2 register

Enumeration	Value
vv0	0
vv1	1
vv2	2
vv3	3
vv4	4
vv5	5
vv6	6
vv7	7
vv8	8
vv9	9
vv10	10
vv11	11

vv12	12
vv13	13
vv14	14
vv15	15
vh0	16
vh1	17
vh2	18
vh3	19
vh4	20
vh5	21
vh6	22
vh7	23
vh8	24
vh9	25
vh10	26
vh11	27
vh12	28
vh13	29
vh14	30
vh15	31
vh16	32
vh17	33
vh18	34
vh19	35
vh20	36
vh21	37
vh22	38
vh23	39
vh24	40
reserved	41
reserved	42
reserved	43
reserved	44
vout0	45
vout1	46
vout2	47
vgpr0	48
vgpr1	49
vgpr2	50
vgpr3	51
vgpr4	52
vgpr5	53
vgpr6	54
vgpr7	55
vacc0	56
vacc1	57
vsacc0	58
vsacc1	59
vabs	60
vangle	61
reserved	62
reserved	63
reserved	64
reserved	65
reserved	66
reserved	67
reserved	68
reserved	69
reserved	70
reserved	71
reserved	72
reserved	73
reserved	74
reserved	75
reserved	76
reserved	77
reserved	78
reserved	79

reserved	80
reserved	81
reserved	82
reserved	83
reserved	84
reserved	85
reserved	86
reserved	87
reserved	88
reserved	89
reserved	90
reserved	91
reserved	92
reserved	93
reserved	94
reserved	95
reserved	96
reserved	97
reserved	98
reserved	99
reserved	100
reserved	101
reserved	102
reserved	103
reserved	104
reserved	105
reserved	106
reserved	107
reserved	108
reserved	109
reserved	110
reserved	111
reserved	112
reserved	113
reserved	114
reserved	115
reserved	116
reserved	117
reserved	118
reserved	119
reserved	120
reserved	121
reserved	122
reserved	123
reserved	124
reserved	125
reserved	126
reserved	127
reserved	128
reserved	129
reserved	130
reserved	131
reserved	132
reserved	133
reserved	134
reserved	135
reserved	136
reserved	137
reserved	138
reserved	139
reserved	140
reserved	141
reserved	142
reserved	143
reserved	144
reserved	145
reserved	146
reserved	147

reserved	148
reserved	149
reserved	150
reserved	151
reserved	152
reserved	153
reserved	154
reserved	155
reserved	156
reserved	157
reserved	158
reserved	159
reserved	160
reserved	161
reserved	162
reserved	163
reserved	164
reserved	165
reserved	166
reserved	167
reserved	168
reserved	169
reserved	170
reserved	171
reserved	172
reserved	173
reserved	174
reserved	175
reserved	176
reserved	177
reserved	178
reserved	179
reserved	180
reserved	181
reserved	182
reserved	183
reserved	184
reserved	185
reserved	186
reserved	187
reserved	188
reserved	189
reserved	190
reserved	191
reserved	192
reserved	193
reserved	194
reserved	195
reserved	196
reserved	197
reserved	198
reserved	199
reserved	200
reserved	201
reserved	202
reserved	203
reserved	204
reserved	205
reserved	206
reserved	207
reserved	208
reserved	209
reserved	210
reserved	211
reserved	212
reserved	213
reserved	214
reserved	215

reserved	216
reserved	217
reserved	218
reserved	219
reserved	220
reserved	221
reserved	222
reserved	223
reserved	224
reserved	225
reserved	226
reserved	227
reserved	228
reserved	229
reserved	230
reserved	231
reserved	232
reserved	233
reserved	234
reserved	235
reserved	236
reserved	237
reserved	238
reserved	239
reserved	240
reserved	241
reserved	242
reserved	243
reserved	244
reserved	245
reserved	246
reserved	247
reserved	248
reserved	249
reserved	250
reserved	251
reserved	252
reserved	253
vone	254
vzero	255

1.4 Instructions

A B C D H L M N O S X

1.4.1 *abd DEST,IM16*

Description:

Absolute difference of value and a registers

R[DEST] = abs(R[DEST] - IM16)

Condition code flags modified: negative, zero

Action:

```
{
 salu ( SALU_OP_ABSDIFF , R ( DEST ) , IM16 ) ;
 R ( DEST ) = RES ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZ

Affect instruction:

Register	Fields	Details

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (DEST)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

1 TYPE	3 OPCD	DEST	IM16
31 28	27 24	23 16	15 0

Attributes:

isp1, isp2

1.4.2 *abd DEST,IN1,IN2*

Description:

Absolute difference between two registers

R[DEST] = abs (R[IN1] - R[IN2])

Condition code flags modified: negative, zero

Action:

```
{
 salu ( SALU_OP_ABSDIFF , R ( IN1 ) , R ( IN2 ) ) ;
 R ( DEST ) = RES ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZ

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (IN1)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

2 TYPE	3 OPCD	DEST	IN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.3 *abd DEST,VIN1,IN2*

Description:

Absolute difference between a register value and a vector register values and store the sum

VOP1 = VR[VIN1]

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

R[DEST] = SUM(ABSDIFF(VOP1 - VOP2));

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_ABSDIFF , 1 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) )
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 R ( DEST ) = SSAT ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

4 TYPE	3 OPCD	DEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.4 *abd DEST,VIN1,VIN2*

Description:

Absolute difference between two vector register value and store the sum

R[DEST] = SUM(ABSDIFF(VOP1 - VOP2));

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_ABSDIFF , 1 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 R ( DEST ) = SSAT ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	
VR (VIN2)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

5 TYPE	3 OPCD	DEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.5 *abd VDEST,VIN1,IN2*

Description:

Absolute difference between a register value and a vector register value

VOP1 = VR[VIN1];

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

VR[VDEST] = ABSDIFF(VOP1 - VOP2));

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_ABSDIFF , 1 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) )
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VR ( VDEST ) = VRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

6 TYPE	3 OPCD	VDEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.6 *abd VDEST,VIN1,VIN2*

Description:

Absolute difference between two vector register value

VR[VDEST] = ABSDIFF(VOP1 - VOP2));

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_ABSDIFF , 1 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VR ( VDEST ) = VRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	
VR (VIN2)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional

Register	Fields	Details
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

7 TYPE	3 OPCD	VDEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.7 *add DEST,IM16*

Description:

Adds the two source operands using binary addition and stores the result in in the destination register.

R[DEST] = Saturate(R[DEST] + IM16)

Condition code flags modified: carry, overflow, negative, zero

Saturate depends on result, CONFALU.SAT and CONFALU.SGN

Action:

```
{
 salu ( SALU_OP_ADD , R ( DEST ) , IM16 ) ;
 R ( DEST ) = SAT ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZCO

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
OP1	Mask: 0x10000	
OP2	Mask: 0x10000	
R (DEST)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

1 TYPE	1 OPCD	DEST	IM16
31 28	27 24	23 16	15 0

Attributes:

isp1, isp2

1.4.8 *add DEST,IN1,IN2*

Description:

Add two register values

R[DEST] = Saturate(R[IN1] + R[IN2])

Condition code flags modified: carry, overflow, negative, zero

Saturate depends on result, CONFALU.SAT and CONFALU.SGN

Action:

```
{
 salu ( SALU_OP_ADD , R ( IN1 ) , R ( IN2 ) ) ;
 R ( DEST ) = SAT ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZCO

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
OP1	Mask: 0x10000	
OP2	Mask: 0x10000	
R (IN1)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

2 TYPE	1 OPCD	DEST	IN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.9 *add DEST,VIN1,IN2*

Description:

Adds a register value to vector register values and store the sum

VOP1 = VR[VIN1]

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

R[DEST] = SUM(VOP1 + VOP2);

Vector condition code flags modified: carry, overflow, negative, zero

Action:

```
{
 valu ( VALU_OP_ADD , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VFLAG_C = flag_ways ( VFLAG_C , I_VFLAG_C , MASKV ) ;
 VFLAG_O = flag_ways ( VFLAG_O , I_VFLAG_O , MASKV ) ;
 R ( DEST ) = SSAT ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_C	Entire Register	
VFLAG_N	Entire Register	
VFLAG_O	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional

Register	Fields	Details
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_C	Entire Register	
VFLAG_N	Entire Register	
VFLAG_O	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

4 TYPE	1 OPCD	DEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.10 *add DEST,VIN1,VIN2*

Description:

Adds two vector register value and store the sum

R[DEST] = SUM(VOP1 + VOP2);

Vector condition code flags modified: carry, overflow, negative, zero

Action:

```
{
 valu ( VALU_OP_ADD , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VFLAG_C = flag_ways ( VFLAG_C , I_VFLAG_C , MASKV ) ;
 VFLAG_O = flag_ways ( VFLAG_O , I_VFLAG_O , MASKV ) ;
 R ( DEST ) = SSAT ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_C	Entire Register	
VFLAG_N	Entire Register	
VFLAG_O	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	
VR (VIN2)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	

Register	Fields	Details
SSAT	Entire Register	
VFLAG_C	Entire Register	
VFLAG_N	Entire Register	
VFLAG_O	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

5 TYPE	1 OPCD	DEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.11 *add VDEST,VIN1,IN2*

Description:

Adds a register value to a vector register value

VOP1 = VR[VIN1];

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

VR[VDEST] = (VOP1 + VOP2);

Vector condition code flags modified: carry, overflow, negative, zero

Action:

```
{
 valu ( VALU_OP_ADD , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VFLAG_C = flag_ways ( VFLAG_C , I_VFLAG_C , MASKV ) ;
 VFLAG_O = flag_ways ( VFLAG_O , I_VFLAG_O , MASKV ) ;
 VR ( VDEST ) = VSAT ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_C	Entire Register	
VFLAG_N	Entire Register	
VFLAG_O	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	

Register	Fields	Details
VFLAG_C	Entire Register	
VFLAG_N	Entire Register	
VFLAG_O	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

6 TYPE	1 OPCD	VDEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.12 *add VDEST,VIN1,VIN2*

Description:

Adds two vector register value

VR[VDEST] = (VOP1 + VOP2);

Vector condition code flags modified: carry, overflow, negative, zero

Action:

```
{
 valu ( VALU_OP_ADD , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VFLAG_C = flag_ways ( VFLAG_C , I_VFLAG_C , MASKV ) ;
 VFLAG_O = flag_ways ( VFLAG_O , I_VFLAG_O , MASKV ) ;
 VR ( VDEST ) = VSAT ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_C	Entire Register	
VFLAG_N	Entire Register	
VFLAG_O	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	
VR (VIN2)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_C	Entire Register	
VFLAG_N	Entire Register	
VFLAG_O	Entire Register	
VFLAG_Z	Entire Register	

Register	Fields	Details
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

7 TYPE	1 OPCD	VDEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.13 *and DEST,IM16*

Description:

Logic AND of a value to a registers

R[DEST] = R[DEST] & IM16

Condition code flags modified: negative, zero

Action:

```
{
 salu ( SALU_OP_AND , R ( DEST ) , IM16 ) ;
 R ( DEST ) = RES ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZ

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (DEST)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

1 TYPE	7 OPCD	DEST	IM16
31 28	27 24	23 16	15 0

Attributes:
isp1, isp2

1.4.14 *and DEST,IN1,IN2*

Description:

Logic AND of two register values

R[DEST] = R[IN1] & R[IN2]

Condition code flags modified: negative, zero

Action:

```
{
 salu ( SALU_OP_AND , R ( IN1 ) , R ( IN2 ) ) ;
 R ( DEST ) = RES ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZ

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (IN1)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

2 TYPE	7 OPCD	DEST	IN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.15 *and DEST,VIN1,IN2*

Description:

Logical AND between a register value and vector register values,

VOP1 = VR[VIN1]

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

R[DEST] = (VOP1 & VOP2),

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_AND , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 R ( DEST ) = SRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional

Register	Fields	Details
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

4 TYPE	7 OPCD	DEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.16 *and DEST,VIN1,VIN2*

Description:

Logical AND between two vector register value,

R[DEST] = (VOP1 & VOP2)

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_AND , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 R ( DEST ) = SRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	
VR (VIN2)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	

Register	Fields	Details
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

5 TYPE	7 OPCD	DEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.17 *and VDEST,VIN1,IN2*

Description:

Logical AND between a register value and vector register values

VOP1 = VR[VIN1];

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

VR[VDEST] = (VOP1 & VOP2)

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_AND , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VR ( VDEST ) = VRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

6 TYPE	7 OPCD	VDEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.18 *and VDEST,VIN1,VIN2*

Description:

Logical AND between two vector register value

VR[VDEST] = (VOP1 & VOP2)

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_AND , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VR ( VDEST ) = VRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	
VR (VIN2)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

7 TYPE	7 OPCD	VDEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.19 *asl DEST,IN1,IM4*

Description:

Shift a register bit wise up by a value positions. Depending on CONFSAT and CONFSGN the result is saturated

R[DEST] = Saturate(R[IN1] << IM4)

Condition code flags modified: negative, zero

Saturate depends on result, CONFALU.SAT and CONFALU.SGN

Action:

```
{
 salu ( SALU_OP_ASL , R ( IN1 ) , concat ( ( bits < 12 >  ) 0 , IM4 ) ) ;
 R ( DEST ) = SAT ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZ

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (IN1)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
RES32	Entire Register	
SAT	Entire Register	

Encoding:

1 TYPE	14 OPCD	DEST	IN1	0000	IM4
31 28	27 24	23 16	15 8	7 4	3 0

Attributes:
isp1, isp2

1.4.20 *asl DEST,IN1,IN2*

Description:

Shift a register bit wise up by another register value positions. Depending on CONFSAT and CONFSGN the result is saturated

R[DEST] = Saturate(R[IN1] << (R[IN2] & 0xF))

Condition code flags modified: negative, zero

Saturate depends on result, CONFALU.SAT and CONFALU.SGN

Action:

```
{
 salu ( SALU_OP_ASL , R ( IN1 ) , R ( IN2 ) ) ;
 R ( DEST ) = SAT ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZ

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (IN1)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
RES32	Entire Register	
SAT	Entire Register	

Encoding:

2 TYPE	14 OPCD	DEST	IN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.21 *asl DEST,VIN1,IN2*

Description:

Shift up vector register values by a scalar register value and store the sum to a scalar register

VOP1 = VR[VIN1]

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

R[DEST] = sum(VOP1 << (VOP2 & 0xF))

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_ASL , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 R ( DEST ) = SSAT ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

4 TYPE	14 OPCD	DEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.22 asl DEST,VIN1,VIN2

Description:

Shift up vector register values by vector register values and store the sum to a scalar register

R[DEST] = sum(VOP1 << (VOP2 & 0xF))

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_ASL , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 R ( DEST ) = SSAT ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	
VR (VIN2)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

5 TYPE	14 OPCD	DEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.23 asl VDEST,VIN1,IN2

Description:

Shift up vector register values by a scalar register value

VOP1 = VR[VIN1];

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

VR[VDEST] = VOP1 << (VOP2 & 0xF)

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_ASL , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VR ( VDEST ) = VSAT ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

6 TYPE	14 OPCD	VDEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.24 asl VDEST,VIN1,VIN2

Description:

Shift up vector register values by vector register values and

VR[VDEST] = VOP1 << (VOP2 & 0xF)

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_ASL , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VR ( VDEST ) = VSAT ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	
VR (VIN2)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

7 TYPE	14 OPCD	VDEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.25 *asr DEST,IN1,IM4*

Description:

Shift a register value down by a value positions

R[DEST] = ((signed)R[IN1]) >> IM4

Action:

```
{
 salu ( SALU_OP_ASR , R ( IN1 ) , concat ( ( bits < 12 >  ) 0 , IM4 ) ) ;
 R ( DEST ) = RES ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZ

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (IN1)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

1 TYPE	12 OPCD	DEST	IN1	0000	IM4
31 28	27 24	23 16	15 8	7 4	3 0

Attributes:
isp1, isp2

1.4.26 *asr DEST,IN1,IN2*

Description:

Shift a register value down by another register value positions

R[DEST] = ((signed)R[IN1]) >> (R[IN2] & 0xF)

Condition code flags modified: negative, zero

Action:

```
{
 salu ( SALU_OP_ASR , R ( IN1 ) , R ( IN2 ) ) ;
 R ( DEST ) = RES ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZ

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (IN1)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

2 TYPE	12 OPCD	DEST	IN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.27 *asr DEST,VIN1,IN2*

Description:

Shift down vector register values by a scalar register value and store the sum to a scalar register

VOP1 = VR[VIN1]

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

R[DEST] = sum((signed)VOP1 >> (VOP2 & 0xF))

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_ASR , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 R ( DEST ) = SRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

4 TYPE	12 OPCD	DEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.28 *asr DEST,VIN1,VIN2*

Description:

Shift down vector register values by vector register values and store the sum to a scalar register

R[DEST] = sum((signed)VOP1 >> (VOP2 & 0xF))

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_ASR , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
```

```
  R ( DEST ) = SRES ;
 }
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	
VR (VIN2)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

5 TYPE	12 OPCD	DEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.29 *asr VDEST,VIN1,IN2*

Description:

Shift down vector register values by a scalar register value

VOP1 = VR[VIN1];

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

VR[VDEST] = (signed)VOP1 >> (VOP2 & 0xF)

Vector condition code flags modified: negative, zero

Action:

```
 {
  valu ( VALU_OP_ASR , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
  VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
  VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
  VR ( VDEST ) = VRES ;
 }
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

6 TYPE	12 OPCD	VDEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.30 *asr VDEST,VIN1,VIN2*

Description:

Shift down vector register values by vector register values and

VR[VDEST] = (signed)VOP1 >> (VOP2 & 0xF)

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_ASR , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VR ( VDEST ) = VRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	
VR (VIN2)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional

Register	Fields	Details
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

7 TYPE	12 OPCD	VDEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.31 *bal RELADDR*

Description:

Unconditional branch instruction.

Action:

```
{
 NIA = RELADDR ;
}
```

Affect instruction:

Register	Fields	Details
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	

Encoding:

0 TYPE	7 OPCD	0000 / 0000	RELADDR	000	0 COND
31 28	27 24	23 16	15 8	7 5	4 0

Attributes:

isp1, isp2

1.4.32 *bcc RELADDR*

Description:

Branch if carry flag is cleared.

Action:

```
{
 if ( ( CC_ZNCO . C == 0 ) ) NIA = RELADDR ;
}
```

Affect instruction:

Register	Fields	Details
CC_ZNCO	C	Partial
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	Conditional

Encoding:

0 TYPE	7 OPCD	0000 / 0000	RELADDR	000	4 COND
31 28	27 24	23 16	15 8	7 5	4 0

Attributes:

isp1, isp2

1.4.33 *bcs RELADDR*

Description:

Branch if if carry flag is set.

Action:

```
{
 if ( ( CC_ZNCO . C == 1 ) ) NIA = RELADDR ;
}
```

Affect instruction:

Register	Fields	Details
CC_ZNCO	C	Partial
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	Conditional

Encoding:

0 TYPE	7 OPCD	0000 / 0000	RELADDR	000	5 COND
31 28	27 24	23 16	15 8	7 5	4 0

Attributes:

isp1, isp2

1.4.34 *beq RELADDR*

Description:

Branch if equal instruction.

Action:

```
{
 if ( ( CC_ZNCO . Z == 1 ) ) NIA = RELADDR ;
}
```

Affect instruction:

Register	Fields	Details
CC_ZNCO	Z	Partial
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	Conditional

Encoding:

0 TYPE	7 OPCD	0000 / 0000	RELADDR	000	2 COND
31 28	27 24	23 16	15 8	7 5	4 0

Attributes:
isp1, isp2

1.4.35 *bge RELADDR*

Description:

Branch if greater or equal (signed).

Action:

```
{
 if ( ( ( CC_ZNCO . N == 1 ) && ( CC_ZNCO . O == 1 ) ) || ( ( CC_ZNCO . N == 0 ) && ( CC_ZNCO . O ==
}
```

Affect instruction:

Register	Fields	Details
CC_ZNCO	N, O	Partial
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	Conditional

Encoding:

0 TYPE	7 OPCD	0000 / 0000	RELADDR	000	10 COND
31 28	27 24	23 16	15 8	7 5	4 0

Attributes:
isp1, isp2

1.4.36 *bgt RELADDR*

Description:

Branch if greater than (signed).

Action:

```
{
 if ( ( ( CC_ZNCO . N == 1 ) && ( CC_ZNCO . O == 1 ) && ( CC_ZNCO . Z == 0 ) ) || ( ( CC_ZNCO . N ==
}
```

Affect instruction:

Register	Fields	Details
CC_ZNCO	N, O, Z	Partial
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	Conditional

Encoding:

0 TYPE	7 OPCD	0000 / 0000	RELADDR	000	14 COND
31 28	27 24	23 16	15 8	7 5	4 0

Attributes:
isp1, isp2

1.4.37 *bhi RELADDR*

Description:

Branch if higher (unsigned).

Action:

```
{
 if ( ( CC_ZNCO . C == 0 ) && ( CC_ZNCO . Z == 0 ) ) NIA = RELADDR ;
}
```

Affect instruction:

Register	Fields	Details
CC_ZNCO	C, Z	Partial
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	Conditional

Encoding:

0 TYPE	7 OPCD	0000 / 0000	RELADDR	000	12 COND
31 28	27 24	23 16	15 8	7 5	4 0

Attributes:
isp1, isp2

1.4.38 *bls RELADDR*

Description:

Branch if lower or same (unsigned).

Action:

```
{
 if ( ( CC_ZNCO . C == 1 ) || ( CC_ZNCO . Z == 1 ) ) NIA = RELADDR ;
}
```

Affect instruction:

Register	Fields	Details
CC_ZNCO	C, Z	Partial
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	Conditional

Encoding:

0 TYPE	7 OPCD	0000 / 0000	RELADDR	000	13 COND
31 28	27 24	23 16	15 8	7 5	4 0

Attributes:
isp1, isp2

1.4.39 *blt RELADDR*

Description:

Branch if lower than (signed).

Action:

```
{
 if ( ( ( CC_ZNCO . N & ~ CC_ZNCO . O ) | ( ~ CC_ZNCO . N & CC_ZNCO . O ) ) == 1 ) NIA = RELADDR ;
}
```

Affect instruction:

Register	Fields	Details
CC_ZNCO	N, O	Partial
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	Conditional

Encoding:

0 TYPE	7 OPCD	0000 / 0000	RELADDR	000	11 COND
31 28	27 24	23 16	15 8	7 5	4 0

Attributes:

isp1, isp2

1.4.40 *bmi RELADDR*

Description:

Branch if negative flag is set.

Action:

```
{
 if ( ( CC_ZNCO . N == 1 ) ) NIA = RELADDR ;
}
```

Affect instruction:

Register	Fields	Details
CC_ZNCO	N	Partial
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	Conditional

Encoding:

0 TYPE	7 OPCD	0000 / 0000	RELADDR	000	6 COND
31 28	27 24	23 16	15 8	7 5	4 0

Attributes:

isp1, isp2

1.4.41 *bne RELADDR*

Description:

Branch if not equal instruction.

Action:

```
{
 if ( ( CC_ZNCO . Z == 0 ) ) NIA = RELADDR ;
}
```

Affect instruction:

Register	Fields	Details
CC_ZNCO	Z	Partial
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	Conditional

Encoding:

0 TYPE	7 OPCD	0000 / 0000	RELADDR	000	3 COND
31 28	27 24	23 16	15 8	7 5	4 0

Attributes:
isp1, isp2

1.4.42 *bnv RELADDR*

Description:

Branch never instruction. This is equivalent to a NOP.

Action:

```
{
}
```

Affect instruction:

Register	Fields	Details
IADDR	Entire Register	

Encoding:

0 TYPE	7 OPCD	0000 / 0000	RELADDR	000	1 COND
31 28	27 24	23 16	15 8	7 5	4 0

Attributes:
isp1, isp2

1.4.43 *boc RELADDR*

Description:

Branch if overflow flag is cleared.

Action:

```
{
 if ( ( CC_ZNCO . O == 0 ) ) NIA = RELADDR ;
}
```

Affect instruction:

Register	Fields	Details
CC_ZNCO	O	Partial
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	Conditional

Encoding:

0 TYPE	7 OPCD	0000 / 0000	RELADDR	000	9 COND
31 28	27 24	23 16	15 8	7 5	4 0

Attributes:
isp1, isp2

1.4.44 *bos RELADDR*

Description:

Branch if overflow flag is set.

Action:

```
{
 if ( ( CC_ZNCO . O == 1 ) ) NIA = RELADDR ;
}
```

Affect instruction:

Register	Fields	Details
CC_ZNCO	O	Partial
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	Conditional

Encoding:

0 TYPE	7 OPCD	0000 / 0000	RELADDR	000	8 COND
31 28	27 24	23 16	15 8	7 5	4 0

Attributes:
isp1, isp2

1.4.45 *bpl RELADDR*

Description:

Branch if negative flag is cleared.

Action:

```
{
 if ( ( CC_ZNCO . N == 0 ) ) NIA = RELADDR ;
}
```

Affect instruction:

Register	Fields	Details
CC_ZNCO	N	Partial
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	Conditional

Encoding:

0 TYPE	7 OPCD	0000 / 0000	RELADDR	000	7 COND
31 28	27 24	23 16	15 8	7 5	4 0

Attributes:
isp1, isp2

1.4.46 *clz DEST,IN1*

Description:

Count leading of a register value

R[DEST] = clz (R[IN1])

Condition code flags modified: negative, zero

Action:

```
{
 salu ( SALU_OP_CLZ , R ( IN1 ) , R ( IN1 ) ) ;
 R ( DEST ) = RES ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZ

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (IN1)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

1 TYPE	15 OPCD	DEST	IN1	0	1 CLZBITS	0000
31 28	27 24	23 16	15 8	7	6 4	3 0

Attributes:
isp1, isp2

1.4.47 *clz DEST,VIN1*

Description:

Count leading zeros of vector register values and store the minmum to a scalar register

R[DEST] = min (clz (VOP2))

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_CLZ , 0 , VR ( VIN1 ) , VR ( VIN1 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 R ( DEST ) = SRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional

Register	Fields	Details
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

1 TYPE	15 OPCD	DEST	VIN1	0	4 CLZBITS	0000
31 28	27 24	23 16	15 8	7	6 4	3 0

Attributes:
isp1, isp2

1.4.48 *clz VDEST,VIN1*

Description:

Count leading zeros of vector register values

VR[VDEST] = clz (VOP2)

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_CLZ , 0 , VR ( VIN1 ) , VR ( VIN1 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VR ( VDEST ) = VRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	

Register	Fields	Details
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

1 TYPE	15 OPCD	VDEST	VIN1	0	6 CLZBITS	0000
31 28	27 24	23 16	15 8	7	6 4	3 0

Attributes:
isp1, isp2

1.4.49 *dbg*

Description:

dbg: enter debug mode

Action:

```
{
 halt (  ) ;
}
```

Encoding:

0 TYPE	0 OPCD	0000 / 0000	2 IM16
31 28	27 24	23 16	15 0

Attributes:
isp1, isp2

1.4.50 *done RELADDR,IXO*

Description:

Close a pixel cycle and go to the next

- If I is set then the input data in IN is shifted by H_SHIFT_IN positions
- If X is set then the x-position counter XPOS is incremented by H_SHIFT_X
- If O is set then the output data in OUT is shifted out by H_SHIFT_OUT positions

The program execution will continue at the jump target address

Action:

```
{
 NIA = RELADDR ;
}
```

Affect instruction:

Register	Fields	Details
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	

Encoding:

0 TYPE	2 OPCD	0000 / 0000	RELADDR	0000 / 0	IXO
31 28	27 24	23 16	15 8	7 3	2 0

Attributes:
isp1, isp2

1.4.51 *dout DONE_IN,RELADDR,IXO*

Description:

Close a pixel cycle and go to the next

OUT[0] = R[DONE_IN]

- If I is set then the input data in IN is shifted by H_SHIFT_IN positions
- If X is set then the x-position counter XPOS is incremented by H_SHIFT_X
- If O is set then the output data in OUT is shifted out by H_SHIFT_OUT positions

The program execution will continue at the jump target address

Action:

```
{
 OP1 = R ( DONE_IN ) ;
 OUT ( 0 ) = OP1 ;
 NIA = RELADDR ;
}
```

Affect instruction:

Register	Fields	Details
IADDR	Entire Register	
R (DONE_IN)	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	
OP1	Entire Register	
OUT (0)	Entire Register	

Encoding:

0 TYPE	3 OPCD	DONE_IN	RELADDR	0000 / 0	IXO
31 28	27 24	23 16	15 8	7 3	2 0

Attributes:
isp1, isp2

1.4.52 *dvot DONE_VIN,RELADDR,IXO*

Description:

Close a pixel cycle and go to the next

VOUT[0] = V[DONE_VIN]

- If I is set then the input data in IN is shifted by H_SHIFT_IN positions
- If X is set then the x-position counter XPOS is incremented by H_SHIFT_X
- If O is set then the output data in OUT is shifted out by H_SHIFT_OUT positions

The program execution will continue at the jump target address

Action:

```
{
 VOP1 = VR ( DONE_VIN ) ;
 VOUT ( 0 ) = concat ( ( bits < 4 >  ) 0xf , VR ( DONE_VIN ) ( 63 , 0 ) ) ;
 NIA = RELADDR ;
}
```

Affect instruction:

Register	Fields	Details
IADDR	Entire Register	
VR (DONE_VIN)	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	
VOP1	Entire Register	
VOUT (0)	Entire Register	

Encoding:

0 TYPE	4 OPCD	DONE_VIN	RELADDR	0000 / 0	IXO
31 28	27 24	23 16	15 8	7 3	2 0

Attributes:
isp1, isp2

1.4.53 *halt*

Description:

halt: terminate line immediatly

Action:

```
{
 End_Of_Line (  ) ;
}
```

The code above uses the following routines (directly or indirectly): End_Of_Line, StartLine, StreamDMAStartLine

Affect instruction:

Register	Fields	Details
H_POS	Entire Register	Conditional
H_XCFG	Entire Register	Conditional
I_BUFFERED	Entire Register	
I_INCFG	NEXTCFG	Partial, Conditional
I_OUTCFG	NEXTCFG	Partial, Conditional
I_RUN	Entire Register	Conditional
I_START	NEXT	Partial, Conditional
S_CHCFG_IN (0, 4)	Entire Register	Conditional
S_CHCFG_OUT (0, 2)	Entire Register	Conditional
S_LINELEN_IN (0, 4)	Entire Register	Conditional
S_LINELEN_OUT (0, 2)	Entire Register	Conditional
S_LINE_PTR_IN (0, 4)	Entire Register	Conditional
S_LINE_PTR_OUT (0, 2)	Entire Register	Conditional

Affected by instruction:

Register	Fields	Details
H_CURRXCFG	Entire Register	Conditional
I_BUFFERED	Entire Register	Conditional
I_EVENTS	Entire Register	
I_INCFG	CURRCFG	Partial, Conditional
I_OUTCFG	CURRCFG	Partial, Conditional
I_POS_IN (0, 4)	Entire Register	Conditional
I_POS_OUT (0, 2)	Entire Register	Conditional
I_RPTCNT_IN (0, 4)	Entire Register	Conditional
I_RUN	Entire Register	
I_SKIPCNT_OUT (0, 2)	Entire Register	Conditional
I_START	CURR	Partial, Conditional
LOCK	Entire Register	Conditional
MASKV	Entire Register	Conditional
NIA	Entire Register	Conditional
SIG_LINE_DONE	Entire Register	
S_CURRCHCFG_IN (0, 4)	Entire Register	Conditional
S_CURRCHCFG_OUT (0, 2)	Entire Register	Conditional
S_CURRLINELEN_IN (0, 4)	Entire Register	Conditional
S_CURRLINELEN_OUT (0, 2)	Entire Register	Conditional

Register	Fields	Details
S_CURRLINE_PTR_IN (0, 4)	Entire Register	Conditional
S_CURRLINE_PTR_OUT (0, 2)	Entire Register	Conditional
XPOS	Entire Register	Conditional
YPOS	Entire Register	Conditional

Encoding:

0 TYPE	0 OPCD	0000 / 0000	1 IM16
31 28	27 24	23 16	15 0

Attributes:
isp1, isp2

1.4.54 *ldon RELADDR,IXO*

Description:

Close a pixel cycle and go to the next

- If I is set then the input data in IN is shifted by H_SHIFT_IN positions
- If X is set then the x-position counter XPOS is incremented by H_SHIFT_X
- If O is set then the output data in OUT is shifted out by H_SHIFT_OUT positions

If the LOOPCNT is >1 the program execution will continue at the jump target address, and the LOOPCNT is decremented. Otherwise the execution will continue with the next address

Action:

```
{
 if ( LOOPCNT > 1 ) {
     NIA = RELADDR ;
 }
 if ( LOOPCNT != 0 ) {
     LOOPCNT = LOOPCNT - 1 ;
 }
}
```

Affect instruction:

Register	Fields	Details
IADDR	Entire Register	
LOOPCNT	Entire Register	

Affected by instruction:

Register	Fields	Details
LOOPCNT	Entire Register	Conditional
NIA	Entire Register	Conditional

Encoding:

0 TYPE	10 OPCD	0000 / 0000	RELADDR	0000	0 LCNT	IXO
31 28	27 24	23 16	15 8	7 4	3	2 0

Attributes:
isp1, isp2

1.4.55 *ldon1 RELADDR,IXO*

Description:

Close a pixel cycle and go to the next

- If I is set then the input data in IN* is shifted by H_SHIFT_IN positions
- If X is set then the x-position counter XPOS is incremented by H_SHIFT_X
- If O is set then the output data in OUT is shifted out by H_SHIFT_OUT positions

If the LOOPCNT1 is >1 the program execution will continue at the jump target address, and the LOOPCNT1 is decremented. Otherwise the execution will continue with the next address

Action:

```
{
 if ( LOOPCNT1 > 1 ) {
     NIA = RELADDR ;
 }
 if ( LOOPCNT1 != 0 ) {
     LOOPCNT1 = LOOPCNT1 - 1 ;
 }
}
```

Affect instruction:

Register	Fields	Details
IADDR	Entire Register	
LOOPCNT1	Entire Register	

Affected by instruction:

Register	Fields	Details
LOOPCNT1	Entire Register	Conditional
NIA	Entire Register	Conditional

Encoding:

0 TYPE	10 OPCD	0000 / 0000	RELADDR	0000	1 LCNT	IXO
31 28	27 24	23 16	15 8	7 4	3	2 0

Attributes:

isp2

1.4.56 *ldot DONE_IN,RELADDR,IXO*

Description:

Close a pixel cycle and go to the next

OUT[0] = R[DONE_IN]

- If I is set then the input data in IN* is shifted by H_SHIFT_IN positions
- If X is set then the x-position counter XPOS is incremented by H_SHIFT_X
- If O is set then the output data in OUT is shifted out by H_SHIFT_OUT positions

If the LOOPCNT is >1 the program execution will continue at the jump target address, and the LOOPCNT is decremented. Otherwise the execution will continue with the next address

Action:

```
{
 OP1 = R ( DONE_IN ) ;
 OUT ( 0 ) = OP1 ;
 if ( LOOPCNT > 1 ) {
     NIA = RELADDR ;
 }
 if ( LOOPCNT != 0 ) {
     LOOPCNT = LOOPCNT - 1 ;
 }
}
```

Affect instruction:

Register	Fields	Details
IADDR	Entire Register	
LOOPCNT	Entire Register	
R (DONE_IN)	Entire Register	

Affected by instruction:

Register	Fields	Details
LOOPCNT	Entire Register	Conditional

Register	Fields	Details
NIA	Entire Register	Conditional
OP1	Entire Register	
OUT (0)	Entire Register	

Encoding:

0 TYPE	11 OPCD	DONE_IN	RELADDR	0000	0 LCNT	IXO
31 28	27 24	23 16	15 8	7 4	3	2 0

Attributes:
isp1, isp2

1.4.57 *ldot1 DONE_IN,RELADDR,IXO*

Description:

Close a pixel cycle and go to the next

OUT[0] = R[DONE_IN]

- If I is set then the input data in IN* is shifted by H_SHIFT_IN positions
- If X is set then the x-position counter XPOS is incremented by H_SHIFT_X
- If O is set then the output data in OUT is shifted out by H_SHIFT_OUT positions

If the LOOPCNT1 is >1 the program execution will continue at the jump target address, and the LOOPCNT1 is decremented. Otherwise the execution will continue with the next address

Action:

```
{
 OP1 = R ( DONE_IN ) ;
 OUT ( 0 ) = OP1 ;
 if ( LOOPCNT1 > 1 ) {
     NIA = RELADDR ;
 }
 if ( LOOPCNT1 != 0 ) {
     LOOPCNT1 = LOOPCNT1 - 1 ;
 }
}
```

Affect instruction:

Register	Fields	Details
IADDR	Entire Register	
LOOPCNT1	Entire Register	
R (DONE_IN)	Entire Register	

Affected by instruction:

Register	Fields	Details
LOOPCNT1	Entire Register	Conditional
NIA	Entire Register	Conditional
OP1	Entire Register	
OUT (0)	Entire Register	

Encoding:

0 TYPE	11 OPCD	DONE_IN	RELADDR	0000	1 LCNT	IXO
31 28	27 24	23 16	15 8	7 4	3	2 0

Attributes:
isp2

1.4.58 *ldvo DONE_VIN,RELADDR,IXO*

Description:

Close a pixel cycle and go to the next

VOUT[0] = V[DONE_VIN]

- If I is set then the input data in IN* is shifted by H_SHIFT_IN positions
- If X is set then the x-position counter XPOS is incremented by H_SHIFT_X
- If O is set then the output data in OUT is shifted out by H_SHIFT_OUT positions

If the LOOPCNT is >1 the program execution will continue at the jump target address, and the LOOPCNT is decremented. Otherwise the execution will continue with the next address

Action:

```
{
 VOP1 = VR ( DONE_VIN ) ;
 VOUT ( 0 ) = concat ( ( bits < 4 >  ) 0xf , VR ( DONE_VIN ) ( 63 , 0 ) ) ;
 if ( LOOPCNT > 1 ) {
     NIA = RELADDR ;
 }
 if ( LOOPCNT != 0 ) {
     LOOPCNT = LOOPCNT - 1 ;
 }
}
```

Affect instruction:

Register	Fields	Details
IADDR	Entire Register	
LOOPCNT	Entire Register	
VR (DONE_VIN)	Entire Register	

Affected by instruction:

Register	Fields	Details
LOOPCNT	Entire Register	Conditional
NIA	Entire Register	Conditional
VOP1	Entire Register	
VOUT (0)	Entire Register	

Encoding:

0 TYPE	12 OPCD	DONE_VIN	RELADDR	0000	0 LCNT	IXO
31 28	27 24	23 16	15 8	7 4	3	2 0

Attributes:
isp1, isp2

1.4.59 *ldvo1 DONE_VIN,RELADDR,IXO*

Description:

Close a pixel cycle and go to the next

VOUT[0] = V[DONE_VIN]

- If I is set then the input data in IN* is shifted by H_SHIFT_IN positions
- If X is set then the x-position counter XPOS is incremented by H_SHIFT_X
- If O is set then the output data in OUT is shifted out by H_SHIFT_OUT positions

If the LOOPCNT1 is >1 the program execution will continue at the jump target address, and the LOOPCNT1 is decremented. Otherwise the execution will continue with the next address

Action:

```
{
 VOP1 = VR ( DONE_VIN ) ;
 VOUT ( 0 ) = concat ( ( bits < 4 >  ) 0xf , VR ( DONE_VIN ) ( 63 , 0 ) ) ;
 if ( LOOPCNT1 > 1 ) {
     NIA = RELADDR ;
 }
 if ( LOOPCNT1 != 0 ) {
     LOOPCNT1 = LOOPCNT1 - 1 ;
 }
}
```

Affect instruction:

Register	Fields	Details
IADDR	Entire Register	
LOOPCNT1	Entire Register	
VR (DONE_VIN)	Entire Register	

Affected by instruction:

Register	Fields	Details
LOOPCNT1	Entire Register	Conditional
NIA	Entire Register	Conditional
VOP1	Entire Register	
VOUT (0)	Entire Register	

Encoding:

0 TYPE	12 OPCD	DONE_VIN	RELADDR	0000	1 LCNT	IXO
31 28	27 24	23 16	15 8	7 4	3	2 0

Attributes:
isp2

1.4.60 *loop RELADDR*

Description:

loop, check LOOPCNT > 1 and if yes then jump back to loop start. The loop counter LOOPCNT gives how often we will go through the loop, thus, LOOPCNT == 2 will jump back once to iterate twice

Action:

```
{
 if ( LOOPCNT > 1 ) {
     NIA = RELADDR ;
 }
 if ( LOOPCNT != 0 ) {
     LOOPCNT = LOOPCNT - 1 ;
 }
}
```

Affect instruction:

Register	Fields	Details
IADDR	Entire Register	
LOOPCNT	Entire Register	

Affected by instruction:

Register	Fields	Details
LOOPCNT	Entire Register	Conditional
NIA	Entire Register	Conditional

Encoding:

0 TYPE	6 OPCD	0000 / 0000	RELADDR	0000	0 LCNT	000
31 28	27 24	23 16	15 8	7 4	3	2 0

Attributes:
isp1, isp2

1.4.61 *loop1 RELADDR*

Description:

loop1, check LOOPCNT1 > 1 and if yes then jump back to loop start. The loop counter LOOPCNT1 gives how often we will go through the loop, thus, LOOPCNT1 == 2 will jump back once to iterate twice

Action:

```
{
 if ( LOOPCNT1 > 1 ) {
     NIA = RELADDR ;
 }
 if ( LOOPCNT1 != 0 ) {
     LOOPCNT1 = LOOPCNT1 - 1 ;
 }
}
```

Affect instruction:

Register	Fields	Details
IADDR	Entire Register	
LOOPCNT1	Entire Register	

Affected by instruction:

Register	Fields	Details
LOOPCNT1	Entire Register	Conditional
NIA	Entire Register	Conditional

Encoding:

0 TYPE	6 OPCD	0000 / 0000	RELADDR	0000	1 LCNT	000
31 28	27 24	23 16	15 8	7 4	3	2 0

Attributes:

isp2

1.4.62 *lsl DEST,IN1,IM4*

Description:

Bit shift left by a value. The result is masked to the lower 16 bits and no saturation is applied

R[DEST] = (R[IN1] << IM4) & 0xFFFF

Condition code flags modified: negative, zero

Action:

```
{
 salu ( SALU_OP_ASL , R ( IN1 ) , concat ( ( bits < 12 >  ) 0 , IM4 ) ) ;
 R ( DEST ) = RES ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZ

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (IN1)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
RES32	Entire Register	
SAT	Entire Register	

Encoding:

1 TYPE	15 OPCD	DEST	IN1	0	0 CLZBITS	IM4
31 28	27 24	23 16	15 8	7	6 4	3 0

Attributes:

isp1, isp2

1.4.63 *lsl DEST,IN1,IN2*

Description:

Bit shift left by a register value. The result is masked to the lower 16 bits and no saturation is applied

R[DEST] = (R[IN1] << (R[IN2] & 0xF)) & 0xFFFF

Condition code flags modified: negative, zero

Action:

```
{
 salu ( SALU_OP_ASL , R ( IN1 ) , R ( IN2 ) ) ;
 R ( DEST ) = RES ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZ

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (IN1)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
RES32	Entire Register	
SAT	Entire Register	

Encoding:

2 TYPE	15 OPCD	DEST	IN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.64 *lsl DEST,VIN1,IN2*

Description:

Bit shift left vector register values by a scalar register value and store the sum to a scalar register

VOP1 = VR[VIN1]

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

R[DEST] = sum((VOP1 << (VOP2 & 0xF)) & 0xFFFF

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_ASL , 1 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 R ( DEST ) = SSAT ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

4 TYPE	15 OPCD	DEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.65 *lsl DEST,VIN1,VIN2*

Description:

Bit shift left vector register values by vector register values and store the sum to a scalar register Do not saturate results

R[DEST] = sum((VOP1 << (VOP2 & 0xF)) & 0xFFFF

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_ASL , 1 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 R ( DEST ) = SSAT ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	
VR (VIN2)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

5 TYPE	15 OPCD	DEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.66 *lsl VDEST,VIN1,IN2*

Description:

Bit shift left vector register values by a scalar register value

VOP1 = VR[VIN1];

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

VR[VDEST] = (VOP1 << (VOP2 & 0xF)) & 0xFFFF

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_ASL , 1 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VR ( VDEST ) = VRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional

Register	Fields	Details
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

6 TYPE	15 OPCD	VDEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.67 *lsl VDEST,VIN1,VIN2*

Description:

Bit shift left vector register values by vector register values and Do not saturate results

VR[VDEST] = (VOP1 << (VOP2 & 0xF)) & 0xFFFF

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_ASL , 1 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VR ( VDEST ) = VRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	
VR (VIN2)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

7 TYPE	15 OPCD	VDEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.68 lsr DEST,IN1,IM4

Description:

Shift a register bit wise down by a value positions

R[DEST] = ((unsigned)R[IN1]) >> IM4

Condition code flags modified: negative, zero

Action:

```
{
 salu ( SALU_OP_LSR , R ( IN1 ) , concat ( ( bits < 12 >  ) 0 , IM4 ) ) ;
 R ( DEST ) = RES ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZ

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (IN1)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

1 TYPE	13 OPCD	DEST	IN1	0000	IM4
31 28	27 24	23 16	15 8	7 4	3 0

Attributes:
isp1, isp2

1.4.69 lsr DEST,IN1,IN2

Description:

Shift a register bit wise down by another register value positions

R[DEST] = ((unsigned)R[IN1]) >> (R[IN2] & 0xF)

Action:

```
{
 salu ( SALU_OP_LSR , R ( IN1 ) , R ( IN2 ) ) ;
 R ( DEST ) = RES ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZ

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (IN1)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

2 TYPE	13 OPCD	DEST	IN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.70 *lsr DEST,VIN1,IN2*

Description:

Logical shift down vector register values by a scalar register value and store the sum to a scalar register

VOP1 = VR[VIN1]

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

R[DEST] = sum((unsigned)VOP1 >> (VOP2 & 0xF)

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_LSR , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 R ( DEST ) = SRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	

Register	Fields	Details
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

4 TYPE	13 OPCD	DEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.71 *lsr DEST,VIN1,VIN2*

Description:

Logical shift down vector register values by vector register values and store the sum to a scalar register

R[DEST] = sum((unsigned)VOP1 >> (VOP2 & 0xF)

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_LSR , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 R ( DEST ) = SRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	
VR (VIN2)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

5 TYPE	13 OPCD	DEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.72 lsr VDEST,VIN1,IN2

Description:

Logical shift down vector register values by a scalar register value

VOP1 = VR[VIN1];

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

VR[VDEST] = (unsigned)VOP1 >> (unsigned)VOP2

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_LSR , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VR ( VDEST ) = VRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

6 TYPE	13 OPCD	VDEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.73 lsr VDEST,VIN1,VIN2

Description:

Logical shift down vector register values by vector register values and

VR[VDEST] = (unsigned)VOP1 >> (unsigned)VOP2

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_LSR , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VR ( VDEST ) = VRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	
VR (VIN2)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

7 TYPE	13 OPCD	VDEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.74 *max DEST,IM16*

Description:

Maximum of a value to a registers

R[DEST] = max (R[DEST] , IM16)

Condition code flags modified: negative, zero

Action:

```
{
 salu ( SALU_OP_MAX , R ( DEST ) , IM16 ) ;
 R ( DEST ) = RES ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZ

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (DEST)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

1 TYPE	11 OPCD	DEST	IM16
31 28	27 24	23 16	15 0

Attributes:

isp1, isp2

1.4.75 *max DEST,IN1,IN2*

Description:

Maximum of two register values

R[DEST] = max (R[IN1] , R[IN2])

Condition code flags modified: negative, zero

Action:

```
{
 salu ( SALU_OP_MAX , R ( IN1 ) , R ( IN2 ) ) ;
 R ( DEST ) = RES ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZ

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (IN1)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

2 TYPE	11 OPCD	DEST	IN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.76 *max DEST,VIN1,IN2*

Description:

Maximum of a register value and vector register values,

VOP1 = VR[VIN1]

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

R[DEST] = max (VOP1 , VOP2),

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_MAX , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 R ( DEST ) = SRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

4 TYPE	11 OPCD	DEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.77 *max DEST,VIN1,VIN2*

Description:

Maximum of two vector register value,

R[DEST] = max (VOP1 , VOP2)

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_MAX , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 R ( DEST ) = SRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	
VR (VIN2)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

5 TYPE	11 OPCD	DEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.78 *max VDEST,VIN1,IN2*

Description:

Maximum of a register value and vector register values

VOP1 = VR[VIN1];

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

VR[VDEST] = max (VOP1 , VOP2)

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_MAX , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
```

```
  VR ( VDEST ) = VRES ;
 }
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

6 TYPE	11 OPCD	VDEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.79 *max VDEST,VIN1,VIN2*

Description:

Maximum of two vector register value

VR[VDEST] = max (VOP1 , VOP2)

Vector condition code flags modified: negative, zero

Action:

```
 {
  valu ( VALU_OP_MAX , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
  VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
  VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
  VR ( VDEST ) = VRES ;
 }
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	

Register	Fields	Details
VR (VIN1)	Mask: 0x8000000000000000	
VR (VIN2)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

7 TYPE	11 OPCD	VDEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.80 *min DEST,IM16*

Description:

Minimum of a value to a registers

R[DEST] = min(R[DEST] , IM16)

Condition code flags modified: negative, zero

Action:

```
{
 salu ( SALU_OP_MIN , R ( DEST ) , IM16 ) ;
 R ( DEST ) = RES ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZ

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (DEST)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

1 TYPE	10 OPCD	DEST	IM16
31 28	27 24	23 16	15 0

Attributes:
isp1, isp2

1.4.81 *min DEST,IN1,IN2*

Description:

Minimum of two register values

R[DEST] = min (R[IN1] , R[IN2])

Condition code flags modified: negative, zero

Action:

```
{
 salu ( SALU_OP_MIN , R ( IN1 ) , R ( IN2 ) ) ;
 R ( DEST ) = RES ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZ

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (IN1)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

2 TYPE	10 OPCD	DEST	IN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.82 *min DEST,VIN1,IN2*

Description:

Minimum of a register value and vector register values,

VOP1 = VR[VIN1]

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

R[DEST] = min (VOP1 , VOP2),

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_MIN , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 R ( DEST ) = SRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

4 TYPE	10 OPCD	DEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.83 *min DEST,VIN1,VIN2*

Description:

Logical Minimum of two vector register value,

R[DEST] = min (VOP1 , VOP2)

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_MIN , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 R ( DEST ) = SRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	
VR (VIN2)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

5 TYPE	10 OPCD	DEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.84 *min VDEST,VIN1,IN2*

Description:

Logical Minimum of a register value and vector register values

VOP1 = VR[VIN1];

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

VR[VDEST] = min (VOP1 , VOP2)

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_MIN , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VR ( VDEST ) = VRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional

Register	Fields	Details
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

6 TYPE	10 OPCD	VDEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.85 *min VDEST,VIN1,VIN2*

Description:

Logical Minimum of two vector register value

VR[VDEST] = min (VOP1 , VOP2)

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_MIN , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VR ( VDEST ) = VRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	
VR (VIN2)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	

Register	Fields	Details
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

7 TYPE	10 OPCD	VDEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.86 *mov DEST,IM16*

Description:

Move constant to registers

R[DEST] = IM16

Condition code flags modified: none

Action:

```
{
 salu ( SALU_OP_MOV , IM16 , IM16 ) ;
 R ( DEST ) = RES ;
}
```

The code above uses the following routines (directly or indirectly): salu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

1 TYPE	0 OPCD	DEST	IM16
31 28	27 24	23 16	15 0

Attributes:
isp1, isp2

1.4.87 *mov DEST,IN2*

Description:

Copy register to register

R[DEST] = R[IN]

Condition code flags modified: none

Action:

```
{
 salu ( SALU_OP_MOV , R ( IN2 ) , R ( IN2 ) ) ;
 R ( DEST ) = RES ;
}
```

The code above uses the following routines (directly or indirectly): salu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

2 TYPE	0 OPCD	DEST	0000 / 0000	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.88 *mov DEST,VIN2*

Description:

Move the sum of a vector register to a scalar register

VOP = VR[VIN]

R[DEST] = SUM(VOP);

Vector condition code flags modified: none

Action:

```
{
 valu ( VALU_OP_MOV , 0 , VR ( VIN2 ) , VR ( VIN2 ) ) ;
 R ( DEST ) = SRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VR (VIN2)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

5 TYPE	0 OPCD	DEST	0000 / 0000	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.89 *mov VDEST,IN2*

Description:

Move a scalar register value to a vector register

VOP = (R[IN], R[IN], R[IN], R[IN]);

VR[VDEST] = VOP,

Vector condition code flags modified: none

Action:

```
{
 valu ( VALU_OP_MOV , 0 , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) , concat ( R ( IN2
 VR ( VDEST ) = VRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

6 TYPE	0 OPCD	VDEST	0000 / 0000	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.90 *mov VDEST,VIN2*

Description:

Copy a vector register to another vector register

VR[DEST] = VR[VOP]

Vector condition code flags modified: none

Action:

```
{
 valu ( VALU_OP_MOV , 0 , VR ( VIN2 ) , VR ( VIN2 ) ) ;
 VR ( VDEST ) = VRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VR (VIN2)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

7 TYPE	0 OPCD	VDEST	0000 / 0000	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.91 *mulh DEST,IM16*

Description:

multiply a register by a value according to CONFALU and scale to fixed point

R[DEST] = Saturate((R[DEST] * IM16) >> CONFSHR)

Condition code flags modified: none

Saturate depends on result, CONFALU.SAT and CONFALU.SGN

Action:

```
{
 salu ( SALU_OP_MULH , R ( DEST ) , IM16 ) ;
 R ( DEST ) = SAT ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu

Affect instruction:

Register	Fields	Details
CONFALU	SHR, SAT, SGN	Partial

Register	Fields	Details
R (DEST)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
RES32	Entire Register	
SAT	Entire Register	

Encoding:

1 TYPE	4 OPCD	DEST	IM16
31 28	27 24	23 16	15 0

Attributes:
isp1, isp2

1.4.92 *mulh DEST,IN1,IN2*

Description:

multiply a register by a value according to CONFALU and scale to fixed point and

R[DEST] = Saturate(((R[IN1] * R[IN2]) >> CONFSHR)

Condition code flags modified: none

Saturate depends on result, CONFALU.SAT and CONFALU.SGN

Action:

```
{
 salu ( SALU_OP_MULH , R ( IN1 ) , R ( IN2 ) ) ;
 R ( DEST ) = SAT ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu

Affect instruction:

Register	Fields	Details
CONFALU	SHR, SAT, SGN	Partial
R (IN1)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
RES32	Entire Register	
SAT	Entire Register	

Encoding:

2 TYPE	4 OPCD	DEST	IN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.93 *mulh DEST,VIN1,IN2*

Description:

Multiply a register value and vector register values, with sign mode according to CONFSGN, scale to fixpoint according to CONFSHR and store the sum

VOP1 = VR[VIN1]

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

R[DEST] = (VOP1 * VOP2) >> CONFSHR;

Action:

```
{
 valu ( VALU_OP_MULH , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 R ( DEST ) = SSAT ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, valu

Affect instruction:

Register	Fields	Details
CONFALU	SHR, SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

4 TYPE	4 OPCD	DEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.94 *mulh DEST,VIN1,VIN2*

Description:

Multiply two vector register value, with sign mode according to CONFSGN, scale to fixpoint according to CONFSHR and store the sum

R[DEST] = (VOP1 * VOP2) >> CONFSHR;

Action:

```
{
 valu ( VALU_OP_MULH , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 R ( DEST ) = SSAT ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, valu

Affect instruction:

Register	Fields	Details
CONFALU	SHR, SAT, SGN	Partial
MASKV	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	
VR (VIN2)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

5 TYPE	4 OPCD	DEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.95 *mulh VDEST,VIN1,IN2*

Description:

Multiply a register value and vector register values with sign mode according to CONFSGN, scale to fixpoint according to CONFSHR

VOP1 = VR[VIN1];

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

VR[VDEST] = (VOP1 * VOP2) >> CONFSHR;

Action:

```
{
 valu ( VALU_OP_MULH , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VR ( VDEST ) = VSAT ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, valu

Affect instruction:

Register	Fields	Details
CONFALU	SHR, SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional

Register	Fields	Details
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

6 TYPE	4 OPCD	VDEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.96 *mulh VDEST,VIN1,VIN2*

Description:

Multiply two vector register value with sign mode according to CONFSGN, and take the lower 16 bit

VR[VDEST] = (VOP1 * VOP2) >> CONFSHR;

Action:

```
{
 valu ( VALU_OP_MULH , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VR ( VDEST ) = VSAT ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, valu

Affect instruction:

Register	Fields	Details
CONFALU	SHR, SAT, SGN	Partial
MASKV	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	
VR (VIN2)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

7 TYPE	4 OPCD	VDEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.97 *mull DEST,IM16*

Description:

multiply a register by a value and take lower 16 bits

R[DEST] = (R[DEST] * IM16) & 0xFFFF

Condition code flags modified: none

Action:

```
{
 salu ( SALU_OP_MULL , R ( DEST ) , IM16 ) ;
 R ( DEST ) = RES ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (DEST)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
RES32	Entire Register	
SAT	Entire Register	

Encoding:

1 TYPE	5 OPCD	DEST	IM16
31 28	27 24	23 16	15 0

Attributes:
isp1, isp2

1.4.98 *mull DEST,IN1,IN2*

Description:

multiply two registers and take lower 16 bits

R[DEST] = (R[IN1] * R[IN2]) & 0xFFFF

Condition code flags modified: none

Action:

```
{
 salu ( SALU_OP_MULL , R ( IN1 ) , R ( IN2 ) ) ;
 R ( DEST ) = RES ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (IN1)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
RES32	Entire Register	
SAT	Entire Register	

Encoding:

2 TYPE	5 OPCD	DEST	IN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.99 *mull DEST,VIN1,IN2*

Description:

Multiply a register value and vector register values, with sign mode according to CONFSGN, scale to fixpoint according to CONFSHR and take the lower 16 bit

VOP1 = VR[VIN1]

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

R[DEST] = (VOP1 * VOP2) & 0xFFFF;

Action:

```
{
 valu ( VALU_OP_MULL , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 R ( DEST ) = SRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

4 TYPE	5 OPCD	DEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.100 *mull DEST,VIN1,VIN2*

Description:

Multiply two vector register value, with sign mode according to CONFSGN, scale to fixpoint according to CONFSHR and take the lower 16 bit

R[DEST] = (VOP1 * VOP2) & 0xFFFF;

Action:

```
{
 valu ( VALU_OP_MULL , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 R ( DEST ) = SRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	
VR (VIN2)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

5 TYPE	5 OPCD	DEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.101 *mull VDEST,VIN1,IN2*

Description:

Multiply a register value and vector register values with sign mode according to CONFSGN, and take the lower 16 bit

VOP1 = VR[VIN1];

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

VR[VDEST] = (VOP1 * VOP2) & 0xFFFF;

Action:

```
{
 valu ( VALU_OP_MULL , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VR ( VDEST ) = VRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

6 TYPE	5 OPCD	VDEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.102 *mull VDEST,VIN1,VIN2*

Description:

Multiply two vector register value with sign mode according to CONFSGN, and take the lower 16 bit

VR[VDEST] = (VOP1 * VOP2) & 0xFFFF;

Action:

```
{
 valu ( VALU_OP_MULL , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VR ( VDEST ) = VRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	
VR (VIN2)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional

Register	Fields	Details
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

7 TYPE	5 OPCD	VDEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.103 *nop*

Description:

No OPeration

Action:

```
{
 return ;
}
```

Encoding:

0 TYPE	0 OPCD	0000 / 0000	0 IM16
31 28	27 24	23 16	15 0

Attributes:
isp1, isp2

1.4.104 *or DEST,IM16*

Description:

Logic OR of a value to a registers

R[DEST] = R[DEST] | IM16

Condition code flags modified: negative, zero

Action:

```
{
 salu ( SALU_OP_OR , R ( DEST ) , IM16 ) ;
 R ( DEST ) = RES ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZ

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (DEST)	Entire Register	

Affected by instruction:

Register	Fields	Details

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

1 TYPE	8 OPCD	DEST	IM16
31 28	27 24	23 16	15 0

Attributes:
isp1, isp2

1.4.105 *or DEST,IN1,IN2*

Description:

Logic OR of two register values

R[DEST] = R[IN1] | R[IN2]

Condition code flags modified: negative, zero

Action:

```
  {
   salu ( SALU_OP_OR , R ( IN1 ) , R ( IN2 ) ) ;
   R ( DEST ) = RES ( 15 , 0 ) ;
  }
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZ

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (IN1)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

2 TYPE	8 OPCD	DEST	IN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.106 *or DEST,VIN1,IN2*

Description:

Logical OR between a register value and vector register values,

VOP1 = VR[VIN1]

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

R[DEST] = or (VOP1 | VOP2),

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_OR , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 R ( DEST ) = SRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

4 TYPE	8 OPCD	DEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.107 *or DEST,VIN1,VIN2*

Description:

Logical OR between two vector register value,

R[DEST] = or (VOP1 | VOP2)

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_OR , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 R ( DEST ) = SRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	
VR (VIN2)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

5 TYPE	8 OPCD	DEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.108 *or VDEST,VIN1,IN2*

Description:

Logical OR between a register value and vector register values

VOP1 = VR[VIN1];

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

VR[VDEST] = (VOP1 | VOP2)

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_OR , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VR ( VDEST ) = VRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	

Register	Fields	Details
R (IN2)	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

6 TYPE	8 OPCD	VDEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.109 *or VDEST,VIN1,VIN2*

Description:

Logical OR between two vector register value

VR[VDEST] = (VOP1 | VOP2)

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_OR , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VR ( VDEST ) = VRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	
VR (VIN2)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional

Register	Fields	Details
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

7 TYPE	8 OPCD	VDEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.110 *sub DEST,IM16*

Description:

Subtract a value from a registers

R[DEST] = Saturate(R[DEST] - IM16)

Condition code flags modified: carry, overflow, negative, zero

Saturate depends on result, CONFALU.SAT and CONFALU.SGN

Action:

```
{
 salu ( SALU_OP_SUB , R ( DEST ) , IM16 ) ;
 R ( DEST ) = SAT ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZCO

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
OP1	Mask: 0x10000	
OP2	Mask: 0x10000	
R (DEST)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

1 TYPE	2 OPCD	DEST	IM16
31 28	27 24	23 16	15 0

Attributes:

isp1, isp2

1.4.111 *sub DEST,IN1,IN2*

Description:

Subtract two register values

R[DEST] = Saturate(R[IN1] - R[IN2])

Condition code flags modified: carry, overflow, negative, zero

Saturate depends on result, CONFALU.SAT and CONFALU.SGN

Action:

```
{
 salu ( SALU_OP_SUB , R ( IN1 ) , R ( IN2 ) ) ;
 R ( DEST ) = SAT ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZCO

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
OP1	Mask: 0x10000	
OP2	Mask: 0x10000	
R (IN1)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

2 TYPE	2 OPCD	DEST	IN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.112 *sub DEST,VIN1,IN2*

Description:

Subtracts a register value from a vector register values and store the sum

VOP1 = VR[VIN1]

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

R[DEST] = SUM(VOP1 - VOP2);

Vector condition code flags modified: carry, overflow, negative, zero

Action:

```
{
 valu ( VALU_OP_SUB , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VFLAG_C = flag_ways ( VFLAG_C , I_VFLAG_C , MASKV ) ;
 VFLAG_O = flag_ways ( VFLAG_O , I_VFLAG_O , MASKV ) ;
 R ( DEST ) = SSAT ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_C	Entire Register	
VFLAG_N	Entire Register	
VFLAG_O	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_C	Entire Register	
VFLAG_N	Entire Register	
VFLAG_O	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

4 TYPE	2 OPCD	DEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.113 *sub DEST,VIN1,VIN2*

Description:

Subtracts two vector register value and store the sum

R[DEST] = SUM(VOP1 - VOP2);

Vector condition code flags modified: carry, overflow, negative, zero

Action:

```
{
 valu ( VALU_OP_SUB , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VFLAG_C = flag_ways ( VFLAG_C , I_VFLAG_C , MASKV ) ;
 VFLAG_O = flag_ways ( VFLAG_O , I_VFLAG_O , MASKV ) ;
 R ( DEST ) = SSAT ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial

Register	Fields	Details
MASKV	Entire Register	
VFLAG_C	Entire Register	
VFLAG_N	Entire Register	
VFLAG_O	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	
VR (VIN2)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_C	Entire Register	
VFLAG_N	Entire Register	
VFLAG_O	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

5 TYPE	2 OPCD	DEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.114 *sub VDEST,VIN1,IN2*

Description:

Subtracts a register value from a vector register value

VOP1 = VR[VIN1];

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

VR[VDEST] = (VOP1 - VOP2);

Vector condition code flags modified: carry, overflow, negative, zero

Action:

```
{
 valu ( VALU_OP_SUB , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VFLAG_C = flag_ways ( VFLAG_C , I_VFLAG_C , MASKV ) ;
 VFLAG_O = flag_ways ( VFLAG_O , I_VFLAG_O , MASKV ) ;
 VR ( VDEST ) = VSAT ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial

Register	Fields	Details
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_C	Entire Register	
VFLAG_N	Entire Register	
VFLAG_O	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_C	Entire Register	
VFLAG_N	Entire Register	
VFLAG_O	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

6 TYPE	2 OPCD	VDEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.115 *sub VDEST,VIN1,VIN2*

Description:

Subtracts two vector register value

VR[VDEST] = (VOP1 - VOP2);

Vector condition code flags modified: carry, overflow, negative, zero

Action:

```
{
 valu ( VALU_OP_SUB , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VFLAG_C = flag_ways ( VFLAG_C , I_VFLAG_C , MASKV ) ;
 VFLAG_O = flag_ways ( VFLAG_O , I_VFLAG_O , MASKV ) ;
 VR ( VDEST ) = VSAT ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_C	Entire Register	
VFLAG_N	Entire Register	
VFLAG_O	Entire Register	

Register	Fields	Details
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	
VR (VIN2)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_C	Entire Register	
VFLAG_N	Entire Register	
VFLAG_O	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

7 TYPE	2 OPCD	VDEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.116 *xor DEST,IM16*

Description:

Logic OR of a value to a registers

R[DEST] = R[DEST] ^ IM16

Condition code flags modified: negative, zero

Action:

```
{
 salu ( SALU_OP_XOR , R ( DEST ) , IM16 ) ;
 R ( DEST ) = RES ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZ

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (DEST)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

1 TYPE	9 OPCD	DEST	IM16
31 28	27 24	23 16	15 0

Attributes:

isp1, isp2

1.4.117 *xor DEST,IN1,IN2*

Description:

Logic OR of two register values

R[DEST] = R[IN1] ^ R[IN2]

Condition code flags modified: negative, zero

Action:

```
{
 salu ( SALU_OP_XOR , R ( IN1 ) , R ( IN2 ) ) ;
 R ( DEST ) = RES ( 15 , 0 ) ;
}
```

The code above uses the following routines (directly or indirectly): salu, salu_cc_set_NZ

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (IN1)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
OP1	Entire Register	
OP2	Entire Register	
R (DEST)	Entire Register	
RES	Entire Register	
SAT	Entire Register	

Encoding:

2 TYPE	9 OPCD	DEST	IN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.118 *xor DEST,VIN1,IN2*

Description:

Logical XOR between a register value and vector register values,

VOP1 = VR[VIN1]

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

R[DEST] = (VOP1 ^ VOP2),

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_XOR , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
```

```
  R ( DEST ) = SRES ;
 }
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x8000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

4 TYPE	9 OPCD	DEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:
isp1, isp2

1.4.119 *xor DEST,VIN1,VIN2*

Description:

Logical XOR between two vector register value,

R[DEST] = (VOP1 ^ VOP2)

Vector condition code flags modified: negative, zero

Action:

```
 {
  valu ( VALU_OP_XOR , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
  VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
  VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
  R ( DEST ) = SRES ;
 }
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	

Register	Fields	Details
VR (VIN1)	Mask: 0x800000000000000000	
VR (VIN2)	Mask: 0x800000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
R (DEST)	Entire Register	
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

5 TYPE	9 OPCD	DEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.120 *xor VDEST,VIN1,IN2*

Description:

Logical XOR between a register value and vector register values

VOP1 = VR[VIN1];

VOP2 = (R[IN2], R[IN2], R[IN2], R[IN2]);

VR[VDEST] = (VOP1 ^ VOP2)

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_XOR , 0 , VR ( VIN1 ) , concat ( R ( IN2 ) , R ( IN2 ) , R ( IN2 ) , R ( IN2 ) ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VR ( VDEST ) = VRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
R (IN2)	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x800000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional

Register	Fields	Details
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

6 TYPE	9 OPCD	VDEST	VIN1	IN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.4.121 *xor VDEST,VIN1,VIN2*

Description:

Logical XOR between two vector register value

VR[VDEST] = (VOP1 ^ VOP2)

Vector condition code flags modified: negative, zero

Action:

```
{
 valu ( VALU_OP_XOR , 0 , VR ( VIN1 ) , VR ( VIN2 ) ) ;
 VFLAG_N = flag_ways ( VFLAG_N , I_VFLAG_N , MASKV ) ;
 VFLAG_Z = flag_ways ( VFLAG_Z , I_VFLAG_Z , MASKV ) ;
 VR ( VDEST ) = VRES ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O, flag_ways, valu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASKV	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	
VR (VIN1)	Mask: 0x80000000000000000	
VR (VIN2)	Mask: 0x80000000000000000	

Affected by instruction:

Register	Fields	Details
I_VFLAG_C	Entire Register	Conditional
I_VFLAG_L	Entire Register	Conditional
I_VFLAG_N	Entire Register	Conditional
I_VFLAG_O	Entire Register	Conditional
I_VFLAG_U	Entire Register	Conditional
I_VFLAG_Z	Entire Register	Conditional
SRES	Entire Register	
SSAT	Entire Register	
VFLAG_N	Entire Register	
VFLAG_Z	Entire Register	

Register	Fields	Details
VOP1	Entire Register	
VOP2	Entire Register	
VR (VDEST)	Entire Register	
VRES	Entire Register	
VSAT	Entire Register	

Encoding:

7 TYPE	9 OPCD	VDEST	VIN1	VIN2
31 28	27 24	23 16	15 8	7 0

Attributes:

isp1, isp2

1.5 Instructions by Attribute

1.5.1 *isp1*

- abd DEST,IM16
- abd DEST,IN1,IN2
- abd DEST,VIN1,IN2
- abd DEST,VIN1,VIN2
- abd VDEST,VIN1,IN2
- abd VDEST,VIN1,VIN2
- add DEST,IM16
- add DEST,IN1,IN2
- add DEST,VIN1,IN2
- add DEST,VIN1,VIN2
- add VDEST,VIN1,IN2
- add VDEST,VIN1,VIN2
- and DEST,IM16
- and DEST,IN1,IN2
- and DEST,VIN1,IN2
- and DEST,VIN1,VIN2
- and VDEST,VIN1,IN2
- and VDEST,VIN1,VIN2
- asl DEST,IN1,IM4
- asl DEST,IN1,IN2
- asl DEST,VIN1,IN2
- asl DEST,VIN1,VIN2
- asl VDEST,VIN1,IN2
- asl VDEST,VIN1,VIN2
- asr DEST,IN1,IM4
- asr DEST,IN1,IN2
- asr DEST,VIN1,IN2
- asr DEST,VIN1,VIN2
- asr VDEST,VIN1,IN2
- asr VDEST,VIN1,VIN2
- bal RELADDR
- bcc RELADDR
- bcs RELADDR
- beq RELADDR
- bge RELADDR
- bgt RELADDR
- bhi RELADDR
- bls RELADDR
- blt RELADDR

- bmi RELADDR
- bne RELADDR
- bnv RELADDR
- boc RELADDR
- bos RELADDR
- bpl RELADDR
- clz DEST,IN1
- clz DEST,VIN1
- clz VDEST,VIN1
- dbg
- done RELADDR,IXO
- dout DONE_IN,RELADDR,IXO
- dvot DONE_VIN,RELADDR,IXO
- halt
- ldon RELADDR,IXO
- ldot DONE_IN,RELADDR,IXO
- ldvo DONE_VIN,RELADDR,IXO
- loop RELADDR
- lsl DEST,IN1,IM4
- lsl DEST,IN1,IN2
- lsl DEST,VIN1,IN2
- lsl DEST,VIN1,VIN2
- lsl VDEST,VIN1,IN2
- lsl VDEST,VIN1,VIN2
- lsr DEST,IN1,IM4
- lsr DEST,IN1,IN2
- lsr DEST,VIN1,IN2
- lsr DEST,VIN1,VIN2
- lsr VDEST,VIN1,IN2
- lsr VDEST,VIN1,VIN2
- max DEST,IM16
- max DEST,IN1,IN2
- max DEST,VIN1,IN2
- max DEST,VIN1,VIN2
- max VDEST,VIN1,IN2
- max VDEST,VIN1,VIN2
- min DEST,IM16
- min DEST,IN1,IN2
- min DEST,VIN1,IN2
- min DEST,VIN1,VIN2
- min VDEST,VIN1,IN2
- min VDEST,VIN1,VIN2
- mov DEST,IM16
- mov DEST,IN2
- mov DEST,VIN2
- mov VDEST,IN2
- mov VDEST,VIN2
- mulh DEST,IM16
- mulh DEST,IN1,IN2
- mulh DEST,VIN1,IN2
- mulh DEST,VIN1,VIN2
- mulh VDEST,VIN1,IN2
- mulh VDEST,VIN1,VIN2
- mull DEST,IM16
- mull DEST,IN1,IN2
- mull DEST,VIN1,IN2
- mull DEST,VIN1,VIN2
- mull VDEST,VIN1,IN2
- mull VDEST,VIN1,VIN2
- nop

- or DEST,IM16
- or DEST,IN1,IN2
- or DEST,VIN1,IN2
- or DEST,VIN1,VIN2
- or VDEST,VIN1,IN2
- or VDEST,VIN1,VIN2
- sub DEST,IM16
- sub DEST,IN1,IN2
- sub DEST,VIN1,IN2
- sub DEST,VIN1,VIN2
- sub VDEST,VIN1,IN2
- sub VDEST,VIN1,VIN2
- xor DEST,IM16
- xor DEST,IN1,IN2
- xor DEST,VIN1,IN2
- xor DEST,VIN1,VIN2
- xor VDEST,VIN1,IN2
- xor VDEST,VIN1,VIN2

1.5.2 *isp2*

- abd DEST,IM16
- abd DEST,IN1,IN2
- abd DEST,VIN1,IN2
- abd DEST,VIN1,VIN2
- abd VDEST,VIN1,IN2
- abd VDEST,VIN1,VIN2
- add DEST,IM16
- add DEST,IN1,IN2
- add DEST,VIN1,IN2
- add DEST,VIN1,VIN2
- add VDEST,VIN1,IN2
- add VDEST,VIN1,VIN2
- and DEST,IM16
- and DEST,IN1,IN2
- and DEST,VIN1,IN2
- and DEST,VIN1,VIN2
- and VDEST,VIN1,IN2
- and VDEST,VIN1,VIN2
- asl DEST,IN1,IM4
- asl DEST,IN1,IN2
- asl DEST,VIN1,IN2
- asl DEST,VIN1,VIN2
- asl VDEST,VIN1,IN2
- asl VDEST,VIN1,VIN2
- asr DEST,IN1,IM4
- asr DEST,IN1,IN2
- asr DEST,VIN1,IN2
- asr DEST,VIN1,VIN2
- asr VDEST,VIN1,IN2
- asr VDEST,VIN1,VIN2
- bal RELADDR
- bcc RELADDR
- bcs RELADDR
- beq RELADDR
- bge RELADDR
- bgt RELADDR
- bhi RELADDR

- bls RELADDR
- blt RELADDR
- bmi RELADDR
- bne RELADDR
- bnv RELADDR
- boc RELADDR
- bos RELADDR
- bpl RELADDR
- clz DEST,IN1
- clz DEST,VIN1
- clz VDEST,VIN1
- dbg
- done RELADDR,IXO
- dout DONE_IN,RELADDR,IXO
- dvot DONE_VIN,RELADDR,IXO
- halt
- ldon RELADDR,IXO
- ldon1 RELADDR,IXO
- ldot DONE_IN,RELADDR,IXO
- ldot1 DONE_IN,RELADDR,IXO
- ldvo DONE_VIN,RELADDR,IXO
- ldvo1 DONE_VIN,RELADDR,IXO
- loop RELADDR
- loop1 RELADDR
- lsl DEST,IN1,IM4
- lsl DEST,IN1,IN2
- lsl DEST,VIN1,IN2
- lsl DEST,VIN1,VIN2
- lsl VDEST,VIN1,IN2
- lsl VDEST,VIN1,VIN2
- lsr DEST,IN1,IM4
- lsr DEST,IN1,IN2
- lsr DEST,VIN1,IN2
- lsr DEST,VIN1,VIN2
- lsr VDEST,VIN1,IN2
- lsr VDEST,VIN1,VIN2
- max DEST,IM16
- max DEST,IN1,IN2
- max DEST,VIN1,IN2
- max DEST,VIN1,VIN2
- max VDEST,VIN1,IN2
- max VDEST,VIN1,VIN2
- min DEST,IM16
- min DEST,IN1,IN2
- min DEST,VIN1,IN2
- min DEST,VIN1,VIN2
- min VDEST,VIN1,IN2
- min VDEST,VIN1,VIN2
- mov DEST,IM16
- mov DEST,IN2
- mov DEST,VIN2
- mov VDEST,IN2
- mov VDEST,VIN2
- mulh DEST,IM16
- mulh DEST,IN1,IN2
- mulh DEST,VIN1,IN2
- mulh DEST,VIN1,VIN2
- mulh VDEST,VIN1,IN2
- mulh VDEST,VIN1,VIN2
- mull DEST,IM16

- mull DEST,IN1,IN2
- mull DEST,VIN1,IN2
- mull DEST,VIN1,VIN2
- mull VDEST,VIN1,IN2
- mull VDEST,VIN1,VIN2
- nop
- or DEST,IM16
- or DEST,IN1,IN2
- or DEST,VIN1,IN2
- or DEST,VIN1,VIN2
- or VDEST,VIN1,IN2
- or VDEST,VIN1,VIN2
- sub DEST,IM16
- sub DEST,IN1,IN2
- sub DEST,VIN1,IN2
- sub DEST,VIN1,VIN2
- sub VDEST,VIN1,IN2
- sub VDEST,VIN1,VIN2
- xor DEST,IM16
- xor DEST,IN1,IN2
- xor DEST,VIN1,IN2
- xor DEST,VIN1,VIN2
- xor VDEST,VIN1,IN2
- xor VDEST,VIN1,VIN2

1.6 Helper Functions

Built-in helper routines are documented in the ADL user manual.

In addition, the following helper routines are defined for IPUV:

C E F S V

1.6.1 *cc_get_C*

```
int cc_get_C ( valu_op_t alu_op , short res , short op1 , short op2 ) {
  }
```

1.6.2 *cc_get_O*

```
int cc_get_O ( valu_op_t alu_op , short res , short op1 , short op2 ) {
  }
```

1.6.3 *End_Of_Line*

```
void End_Of_Line ( void  ) {
   I_RUN = 0 ;
   I_EVENTS = 1 ;
   SIG_LINE_DONE = 1 ;
   if ( I_BUFFERED == 0 ) {
       halt (  ) ;
   } else {
       StartLine (  ) ;
   }
  }
```

The code above uses the following routines (directly or indirectly): StartLine, StreamDMAStartLine

1.6.4 *flag_ways*

```
bits < 4 > flag_ways ( bits < 4 > flag_registers , bits < 4 > new_flags , bits < 4 > mask ) {
   return ( flag_registers & ~ mask ) | ( new_flags & mask ) ;
  }
```

1.6.5 *salu*

```
void salu ( salu_op_t alu_op , sbits < 16 > op1 , sbits < 16 > op2 ) {
    int32_t sop1 , sop2 , sres ;
   if ( CONFALU . SGN == 1 ) {
       sop1 = op1 . int32 (  ) ;
       sop2 = op2 . int32 (  ) ;
   } else {
       sop1 = op1 . uint32 (  ) ;
       sop2 = op2 . uint32 (  ) ;
   }
   OP1 = op1 ;
   OP2 = op2 ;
   switch ( alu_op ) {
       case SALU_OP_MOV : RES = sop2 ;
       break ;
       case SALU_OP_ADD : RES = sop1 + sop2 ;
       salu_cc_set_NZCO ( alu_op , RES , OP1 , OP2 ) ;
       break ;
       case SALU_OP_SUB : RES = sop1 - sop2 ;
       salu_cc_set_NZCO ( alu_op , RES , OP1 , OP2 ) ;
       break ;
       case SALU_OP_ABSDIFF : RES = ( sop1 > sop2 ) ? ( sop1 - sop2 ) : ( sop2 - sop1 ) ;
       salu_cc_set_NZ ( RES ) ;
       break ;
       case SALU_OP_MULH : {
            int sign = 0 ;
            int shr ;
           shr = CONFALU . SHR . uint32 (  ) ;
           sres = sop1 * sop2 ;
           if ( CONFALU . SGN == 1 ) {
               RES32 = sres >> shr ;
           } else {
               RES32 = ( ( unsigned int  ) sres ) >> shr ;
           }
           if ( ( sop1 != 0 ) && ( sop2 != 0 ) ) {
               if ( sop1 < 0 ) sign ++ ;
               if ( sop2 < 0 ) sign ++ ;
           }
           if ( sign == 1 ) {
               RES = concat ( ( bits < 1 >  ) 1 , ( RES32 ( 30 , 16 ) . uint32 (  ) == 0x7fff ) ? ( bit
           } else {
               if ( CONFALU . SGN == 1 ) {
                   RES = concat ( ( bits < 1 >  ) 0 , ( RES32 ( 30 , 16 ) . uint32 (  ) == 0 ) ? ( bits
               } else {
                   RES = concat ( ( bits < 1 >  ) 0 , ( RES32 ( 31 , 16 ) . uint32 (  ) == 0 ) ? ( bits
               }
           }
       }
       break ;
       case SALU_OP_MULL : RES32 = sop1 * sop2 ;
       RES = RES32 ( 15 , 0 ) ;
       break ;
       case SALU_OP_AND : RES = sop1 & sop2 ;
       salu_cc_set_NZ ( RES ) ;
       break ;
       case SALU_OP_OR : RES = sop1 | sop2 ;
       salu_cc_set_NZ ( RES ) ;
       break ;
       case SALU_OP_XOR : RES = sop1 ^ sop2 ;
       salu_cc_set_NZ ( RES ) ;
       break ;
       case SALU_OP_MIN : RES = ( sop1 < sop2 ) ? sop1 : sop2 ;
       salu_cc_set_NZ ( RES ) ;
       break ;
       case SALU_OP_MAX : RES = ( sop1 > sop2 ) ? sop1 : sop2 ;
       salu_cc_set_NZ ( RES ) ;
       break ;
       case SALU_OP_ASR : RES = op1 . int32 (  ) >> ( sop2 & 0xf ) ;
       salu_cc_set_NZ ( RES ) ;
       break ;
```

```
        case SALU_OP_LSR : RES = ( sop1 & 0xffff ) >> ( sop2 & 0xf ) ;
        salu_cc_set_NZ ( RES ) ;
        break ;
        case SALU_OP_ASL : RES32 = sop1 << ( sop2 & 0xf ) ;
        if ( sop1 < 0 ) {
            RES = concat ( ( bits < 1 >  ) 1 , ( RES32 ( 30 , 16 ) . uint32 (  ) == 0x7fff ) ? ( bits <
        } else {
            RES = concat ( ( bits < 1 >  ) 0 , ( RES32 ( 30 , 16 ) . uint32 (  ) == 0 ) ? ( bits < 1 >
        }
        salu_cc_set_NZ ( RES ) ;
        break ;
        case SALU_OP_CLZ :  int m = 0 ;
        while ( m < 16 && ( ( sop2 & 0x8000 ) == 0 ) ) {
            ++ m ;
            sop2 <<= 1 ;
        }
        if ( m == 16 ) m |= 0xfff0 ;
        RES = m ;
        salu_cc_set_NZ ( RES ) ;
        break ;
    }
    if ( CONFALU . SAT == 1 ) {
        if ( CONFALU . SGN == 1 ) {
            if ( RES . int32 (  ) < ( - 0x8000 ) ) {
                SAT = ( - 0x8000 ) ;
            } else if ( RES . int32 (  ) > ( 0x7FFF ) ) {
                SAT = ( 0x7FFF ) ;
            } else {
                SAT = RES ( 15 , 0 ) ;
            }
        } else {
            if ( RES . int32 (  ) < ( 0x0000 ) ) {
                SAT = ( 0x0000 ) ;
            } else if ( RES . uint32 (  ) > ( 0xFFFF ) ) {
                SAT = ( 0xFFFF ) ;
            } else {
                SAT = RES ( 15 , 0 ) ;
            }
        }
    } else {
        SAT = RES ( 15 , 0 ) ;
    }
  }
```

The code above uses the following routines (directly or indirectly): salu_cc_set_NZ, salu_cc_set_NZCO

1.6.6 *salu_cc_set_NZ*

```
void salu_cc_set_NZ ( sbits < 16 > res ) {
  }
```

1.6.7 *salu_cc_set_NZCO*

```
void salu_cc_set_NZCO ( salu_op_t alu_op , sbits < 16 > res , sbits < 16 > op1 , sbits < 16 > op2 ) {
  }
```

1.6.8 *StartLine*

```
void StartLine (  ) {
   if ( I_BUFFERED == 0 ) return ;
   if ( I_RUN == 1 ) return ;
   I_RUN = 1 ;
   I_BUFFERED = 0 ;
   {
        int start = I_START . NEXT . uint32 (  ) ;
       NIA = start ;
       I_START . CURR = start ;
   }
   H_CURRXCFG = H_XCFG ;
   I_INCFG . CURRCFG = I_INCFG . NEXTCFG ;
   I_OUTCFG . CURRCFG = I_OUTCFG . NEXTCFG ;
   XPOS = H_POS ( 31 , 16 ) ;
   YPOS = H_POS ( 15 , 0 ) ;
   StreamDMAStartLine (  ) ;
   LOCK = - 1 ;
   MASKV = - 1 ;
  }
```

The code above uses the following routines (directly or indirectly): StreamDMAStartLine

1.6.9 *StreamDMAStartLine*

```
void StreamDMAStartLine (  ) {
    int i ;
   for ( i = 0 ; i < 5 ; i ++ ) {
       S_CURRCHCFG_IN ( i ) = S_CHCFG_IN ( i ) ;
   }
   for ( i = 0 ; i < 5 ; i ++ ) {
       S_CURRLINE_PTR_IN ( i ) = S_LINE_PTR_IN ( i ) ;
   }
   for ( i = 0 ; i < 5 ; i ++ ) {
       S_CURRLINELEN_IN ( i ) = S_LINELEN_IN ( i ) ;
   }
   for ( i = 0 ; i < 5 ; i ++ ) {
       I_RPTCNT_IN ( i ) = S_CHCFG_IN ( i ) ( 19 , 18 ) ;
   }
   for ( i = 0 ; i < 5 ; i ++ ) {
       I_POS_IN ( i ) = 0 - S_CHCFG_IN ( i ) ( 26 , 24 ) ;
   }
   for ( i = 0 ; i < 3 ; i ++ ) {
       S_CURRCHCFG_OUT ( i ) = S_CHCFG_OUT ( i ) ;
   }
   for ( i = 0 ; i < 3 ; i ++ ) {
       S_CURRLINE_PTR_OUT ( i ) = S_LINE_PTR_OUT ( i ) ;
   }
   for ( i = 0 ; i < 3 ; i ++ ) {
       S_CURRLINELEN_OUT ( i ) = S_LINELEN_OUT ( i ) ;
   }
   for ( i = 0 ; i < 3 ; i ++ ) {
       I_POS_OUT ( i ) = 0 ;
   }
   for ( i = 0 ; i < 3 ; i ++ ) {
       I_SKIPCNT_OUT ( i ) = 0 ;
   }
  }
```

1.6.10 *valu*

```
void valu ( valu_op_t valu_op , bits < 1 > force_unsigned , bits < 64 > vop1 , bits < 64 > vop2 ) {
    int sop10 , sop11 , sop12 , sop13 ;
    int sop20 , sop21 , sop22 , sop23 ;
    int res0 , res1 , res2 , res3 ;
    int res320 , res321 , res322 , res323 ;
    int minint , maxint ;
    int check_nz ;
    int check_co ;
    int check_clip ;
    int saturate ;
    bits < 16 > op10 , op11 , op12 , op13 ;
    bits < 16 > op20 , op21 , op22 , op23 ;
    bits < 16 > bres0 , bres1 , bres2 , bres3 ;
   op10 = vop1 ( 63 , 48 ) ;
   op11 = vop1 ( 47 , 32 ) ;
   op12 = vop1 ( 31 , 16 ) ;
   op13 = vop1 ( 15 , 0 ) ;
   op20 = vop2 ( 63 , 48 ) ;
   op21 = vop2 ( 47 , 32 ) ;
   op22 = vop2 ( 31 , 16 ) ;
   op23 = vop2 ( 15 , 0 ) ;
   if ( CONFALU . SGN == 1 ) {
       sop10 = op10 . int32 (  ) ;
       sop11 = op11 . int32 (  ) ;
       sop12 = op12 . int32 (  ) ;
       sop13 = op13 . int32 (  ) ;
       sop20 = op20 . int32 (  ) ;
       sop21 = op21 . int32 (  ) ;
       sop22 = op22 . int32 (  ) ;
       sop23 = op23 . int32 (  ) ;
   } else {
       sop10 = op10 . uint32 (  ) ;
       sop11 = op11 . uint32 (  ) ;
       sop12 = op12 . uint32 (  ) ;
       sop13 = op13 . uint32 (  ) ;
       sop20 = op20 . uint32 (  ) ;
       sop21 = op21 . uint32 (  ) ;
       sop22 = op22 . uint32 (  ) ;
       sop23 = op23 . uint32 (  ) ;
   }
   if ( ( CONFALU . SGN == 1 ) && ( force_unsigned == 0 ) ) {
       minint = ( - 0x8000 ) ;
       maxint = ( 0x7FFF ) ;
   } else {
       minint = ( 0x0000 ) ;
       maxint = ( 0xFFFF ) ;
   }
   check_nz = 0 ;
   check_co = 0 ;
   check_clip = 0 ;
   saturate = CONFALU ( 1 ) . uint32 (  ) ;
   VOP1 = vop1 ;
   VOP2 = vop2 ;
   switch ( valu_op ) {
       case VALU_OP_MOV : res0 = sop10 ;
       res1 = sop11 ;
       res2 = sop12 ;
       res3 = sop13 ;
       SRES = ( ( MASKV ( 0 ) == 1 ) ? res0 : 0 ) + ( ( MASKV ( 1 ) == 1 ) ? res1 : 0 ) + ( ( MASKV ( 2
       break ;
       case VALU_OP_ADD : check_nz = 1 ;
       check_co = 1 ;
       check_clip = 1 ;
       res0 = sop10 + sop20 ;
       res1 = sop11 + sop21 ;
       res2 = sop12 + sop22 ;
       res3 = sop13 + sop23 ;
       SRES = ( ( MASKV ( 0 ) == 1 ) ? res0 : 0 ) + ( ( MASKV ( 1 ) == 1 ) ? res1 : 0 ) + ( ( MASKV ( 2
       break ;
       case VALU_OP_SUB : check_nz = 1 ;
       check_co = 1 ;
       check_clip = 1 ;
       res0 = sop10 - sop20 ;
       res1 = sop11 - sop21 ;
       res2 = sop12 - sop22 ;
       res3 = sop13 - sop23 ;
       SRES = ( ( MASKV ( 0 ) == 1 ) ? res0 : 0 ) + ( ( MASKV ( 1 ) == 1 ) ? res1 : 0 ) + ( ( MASKV ( 2
       break ;
       case VALU_OP_ABSDIFF : check_nz = 1 ;
       res0 = ( sop10 > sop20 ) ? ( sop10 - sop20 ) : ( sop20 - sop10 ) ;
       res1 = ( sop11 > sop21 ) ? ( sop11 - sop21 ) : ( sop21 - sop11 ) ;
       res2 = ( sop12 > sop22 ) ? ( sop12 - sop22 ) : ( sop22 - sop12 ) ;
       res3 = ( sop13 > sop23 ) ? ( sop13 - sop23 ) : ( sop23 - sop13 ) ;
       SRES = ( ( MASKV ( 0 ) == 1 ) ? res0 : 0 ) + ( ( MASKV ( 1 ) == 1 ) ? res1 : 0 ) + ( ( MASKV ( 2
       break ;
       case VALU_OP_MULH :  int sign0 , sign1 , sign2 , sign3 ;
        int shr ;
       shr = CONFALU . SHR . uint32 (  ) ;
       check_clip = 1 ;
       if ( CONFALU . SGN == 1 ) {
           res320 = ( sop10 * sop20 ) >> shr ;
           res321 = ( sop11 * sop21 ) >> shr ;
           res322 = ( sop12 * sop22 ) >> shr ;
           res323 = ( sop13 * sop23 ) >> shr ;
       } else {
           res320 = ( ( unsigned int  ) ( sop10 * sop20 ) ) >> shr ;
           res321 = ( ( unsigned int  ) ( sop11 * sop21 ) ) >> shr ;
           res322 = ( ( unsigned int  ) ( sop12 * sop22 ) ) >> shr ;
           res323 = ( ( unsigned int  ) ( sop13 * sop23 ) ) >> shr ;
       }
       sign0 = sign1 = sign2 = sign3 = 0 ;
       res0 = res320 & 0xffff ;
       if ( ( sop10 != 0 ) && ( sop20 != 0 ) ) {
           if ( sop10 < 0 ) sign0 ++ ;
           if ( sop20 < 0 ) sign0 ++ ;
       }
       if ( sign0 == 1 ) {
           res0 |= 0xfffe0000 ;
           if ( ( res320 & 0x7fff8000 ) == 0x7fff8000 ) res0 |= 0x10000 ;
       } else {
           if ( CONFALU ( 0 ) == 1 ) {
               if ( ( res320 & 0x7fff8000 ) != 0 ) res0 |= 0x10000 ;
           } else {
               if ( ( res320 & 0xffff0000 ) != 0 ) res0 |= 0x10000 ;
           }
       }
       res1 = res321 & 0xffff ;
       if ( ( sop11 != 0 ) && ( sop21 != 0 ) ) {
           if ( sop11 < 0 ) sign1 ++ ;
           if ( sop21 < 0 ) sign1 ++ ;
       }
       if ( sign1 == 1 ) {
           res1 |= 0xfffe0000 ;
           if ( ( res321 & 0x7fff8000 ) == 0x7fff8000 ) res1 |= 0x10000 ;
       } else {
           if ( CONFALU ( 0 ) == 1 ) {
               if ( ( res321 & 0x7fff8000 ) != 0 ) res1 |= 0x10000 ;
           } else {
               if ( ( res321 & 0xffff0000 ) != 0 ) res1 |= 0x10000 ;
           }
       }
       res2 = res322 & 0xffff ;
       if ( ( sop12 != 0 ) && ( sop22 != 0 ) ) {
           if ( sop12 < 0 ) sign2 ++ ;
           if ( sop22 < 0 ) sign2 ++ ;
       }
       if ( sign2 == 1 ) {
           res2 |= 0xfffe0000 ;
           if ( ( res322 & 0x7fff8000 ) == 0x7fff8000 ) res2 |= 0x10000 ;
       } else {
           if ( CONFALU . SGN == 1 ) {
               if ( ( res322 & 0x7fff8000 ) != 0 ) res2 |= 0x10000 ;
           } else {
               if ( ( res322 & 0xffff0000 ) != 0 ) res2 |= 0x10000 ;
           }
       }
       res3 = res323 & 0xffff ;
       if ( ( sop13 != 0 ) && ( sop23 != 0 ) ) {
           if ( sop13 < 0 ) sign3 ++ ;
           if ( sop23 < 0 ) sign3 ++ ;
       }
       if ( sign3 == 1 ) {
           res3 |= 0xfffe0000 ;
           if ( ( res323 & 0x7fff8000 ) == 0x7fff8000 ) res3 |= 0x10000 ;
       } else {
           if ( CONFALU . SGN == 1 ) {
               if ( ( res323 & 0x7fff8000 ) != 0 ) res3 |= 0x10000 ;
           } else {
               if ( ( res323 & 0xffff0000 ) != 0 ) res3 |= 0x10000 ;
           }
       }
       SRES = ( ( MASKV ( 0 ) == 1 ) ? res0 : 0 ) + ( ( MASKV ( 1 ) == 1 ) ? res1 : 0 ) + ( ( MASKV ( 2
       break ;
       case VALU_OP_MULL : res320 = sop10 * sop20 ;
       res321 = sop11 * sop21 ;
       res322 = sop12 * sop22 ;
       res323 = sop13 * sop23 ;
       res0 = ( res320 & 0xffff ) ;
       res1 = ( res321 & 0xffff ) ;
       res2 = ( res322 & 0xffff ) ;
       res3 = ( res323 & 0xffff ) ;
       SRES = ( ( MASKV ( 0 ) == 1 ) ? res0 : 0 ) + ( ( MASKV ( 1 ) == 1 ) ? res1 : 0 ) + ( ( MASKV ( 2
       break ;
       case VALU_OP_AND : check_nz = 1 ;
       res0 = sop10 & sop20 ;
       res1 = sop11 & sop21 ;
       res2 = sop12 & sop22 ;
       res3 = sop13 & sop23 ;
       SRES = ( ( MASKV ( 0 ) == 1 ) ? res0 : 0xffff ) & ( ( MASKV ( 1 ) == 1 ) ? res1 : 0xffff ) & ( (
       break ;
       case VALU_OP_OR : check_nz = 1 ;
       res0 = sop10 | sop20 ;
       res1 = sop11 | sop21 ;
       res2 = sop12 | sop22 ;
       res3 = sop13 | sop23 ;
       SRES = ( ( MASKV ( 0 ) == 1 ) ? res0 : 0 ) | ( ( MASKV ( 1 ) == 1 ) ? res1 : 0 ) | ( ( MASKV ( 2
       break ;
       case VALU_OP_XOR : check_nz = 1 ;
       res0 = sop10 ^ sop20 ;
       res1 = sop11 ^ sop21 ;
       res2 = sop12 ^ sop22 ;
       res3 = sop13 ^ sop23 ;
       SRES = ( ( MASKV ( 0 ) == 1 ) ? res0 : 0 ) ^ ( ( MASKV ( 1 ) == 1 ) ? res1 : 0 ) ^ ( ( MASKV ( 2
       break ;
       case VALU_OP_MIN : check_nz = 1 ;
       res0 = ( sop10 < sop20 ) ? sop10 : sop20 ;
       res1 = ( sop11 < sop21 ) ? sop11 : sop21 ;
```

```
res2 = ( sop12 < sop22 ) ? sop12 : sop22 ;
res3 = ( sop13 < sop23 ) ? sop13 : sop23 ;
{
     int mres0 , mres1 , mres2 , mres3 , smin1 , smin2 ;
    mres0 = ( MASKV ( 0 ) == 1 ) ? res0 : maxint ;
    mres1 = ( MASKV ( 1 ) == 1 ) ? res1 : maxint ;
    mres2 = ( MASKV ( 2 ) == 1 ) ? res2 : maxint ;
    mres3 = ( MASKV ( 3 ) == 1 ) ? res3 : maxint ;
    smin1 = ( mres0 < mres1 ) ? mres0 : mres1 ;
    smin2 = ( mres2 < mres3 ) ? mres2 : mres3 ;
    SRES = ( smin1 < smin2 ) ? smin1 : smin2 ;
}
break ;
case VALU_OP_MAX : check_nz = 1 ;
res0 = ( sop10 > sop20 ) ? sop10 : sop20 ;
res1 = ( sop11 > sop21 ) ? sop11 : sop21 ;
res2 = ( sop12 > sop22 ) ? sop12 : sop22 ;
res3 = ( sop13 > sop23 ) ? sop13 : sop23 ;
{
     int mres0 , mres1 , mres2 , mres3 , smax1 , smax2 ;
    mres0 = ( MASKV ( 0 ) == 1 ) ? res0 : minint ;
    mres1 = ( MASKV ( 1 ) == 1 ) ? res1 : minint ;
    mres2 = ( MASKV ( 2 ) == 1 ) ? res2 : minint ;
    mres3 = ( MASKV ( 3 ) == 1 ) ? res3 : minint ;
    smax1 = ( mres0 > mres1 ) ? mres0 : mres1 ;
    smax2 = ( mres2 > mres3 ) ? mres2 : mres3 ;
    SRES = ( smax1 > smax2 ) ? smax1 : smax2 ;
}
break ;
case VALU_OP_ASR : check_nz = 1 ;
res0 = op10 . int32 (  ) >> ( sop20 & 0xf ) ;
res1 = op11 . int32 (  ) >> ( sop21 & 0xf ) ;
res2 = op12 . int32 (  ) >> ( sop22 & 0xf ) ;
res3 = op13 . int32 (  ) >> ( sop23 & 0xf ) ;
SRES = ( ( MASKV ( 0 ) == 1 ) ? res0 : 0 ) + ( ( MASKV ( 1 ) == 1 ) ? res1 : 0 ) + ( ( MASKV ( 2
break ;
case VALU_OP_LSR : check_nz = 1 ;
res0 = ( sop10 & 0xffff ) >> ( sop20 & 0xf ) ;
res1 = ( sop11 & 0xffff ) >> ( sop21 & 0xf ) ;
res2 = ( sop12 & 0xffff ) >> ( sop22 & 0xf ) ;
res3 = ( sop13 & 0xffff ) >> ( sop23 & 0xf ) ;
SRES = ( ( MASKV ( 0 ) == 1 ) ? res0 : 0 ) + ( ( MASKV ( 1 ) == 1 ) ? res1 : 0 ) + ( ( MASKV ( 2
break ;
case VALU_OP_ASL : check_nz = 1 ;
check_clip = 1 ;
res320 = sop10 << ( sop20 & 0xf ) ;
res321 = sop11 << ( sop21 & 0xf ) ;
res322 = sop12 << ( sop22 & 0xf ) ;
res323 = sop13 << ( sop23 & 0xf ) ;
if ( force_unsigned == 1 ) {
    res320 &= 0xffff ;
    res321 &= 0xffff ;
    res322 &= 0xffff ;
    res323 &= 0xffff ;
}
res0 = res320 & 0xffff ;
if ( ( sop10 < 0 ) && ( force_unsigned == 0 ) ) {
    res0 |= 0xfffe0000 ;
    if ( ( res320 & 0x7fff8000 ) == 0x7fff8000 ) res0 |= 0x10000 ;
} else {
    if ( ( CONFALU ( 0 ) == 1 ) & ( force_unsigned == 0 ) ) {
        if ( ( res320 & 0x7fff8000 ) != 0 ) res0 |= 0x10000 ;
    } else {
        if ( ( res320 & 0xffff0000 ) != 0 ) res0 |= 0x10000 ;
    }
}
res1 = res321 & 0xffff ;
if ( ( sop11 < 0 ) && ( force_unsigned == 0 ) ) {
    res1 |= 0xfffe0000 ;
    if ( ( res321 & 0x7fff8000 ) == 0x7fff8000 ) res1 |= 0x10000 ;
} else {
    if ( ( CONFALU ( 0 ) == 1 ) & ( force_unsigned == 0 ) ) {
        if ( ( res321 & 0x7fff8000 ) != 0 ) res1 |= 0x10000 ;
    } else {
        if ( ( res321 & 0xffff0000 ) != 0 ) res1 |= 0x10000 ;
    }
}
res2 = res322 & 0xffff ;
if ( ( sop12 < 0 ) && ( force_unsigned == 0 ) ) {
    res2 |= 0xfffe0000 ;
    if ( ( res322 & 0x7fff8000 ) == 0x7fff8000 ) res2 |= 0x10000 ;
} else {
    if ( ( CONFALU ( 0 ) == 1 ) & ( force_unsigned == 0 ) ) {
            if ( ( res322 & 0x7fff8000 ) != 0 ) res2 |= 0x10000 ;
        } else {
            if ( ( res322 & 0xffff0000 ) != 0 ) res2 |= 0x10000 ;
        }
    }
    res3 = res323 & 0xffff ;
    if ( ( sop13 < 0 ) && ( force_unsigned == 0 ) ) {
        res3 |= 0xfffe0000 ;
        if ( ( res323 & 0x7fff8000 ) == 0x7fff8000 ) res3 |= 0x10000 ;
    } else {
        if ( ( CONFALU ( 0 ) == 1 ) & ( force_unsigned == 0 ) ) {
            if ( ( res323 & 0x7fff8000 ) != 0 ) res3 |= 0x10000 ;
        } else {
            if ( ( res323 & 0xffff0000 ) != 0 ) res3 |= 0x10000 ;
        }
    }
    SRES = ( ( MASKV ( 0 ) == 1 ) ? res0 : 0 ) + ( ( MASKV ( 1 ) == 1 ) ? res1 : 0 ) + ( ( MASKV ( 2
    break ;
    case VALU_OP_CLZ :  int m ;
    check_nz = 1 ;
    m = 0 ;
    while ( m < 16 && ( ( sop20 & 0x8000 ) == 0 ) ) {
        ++ m ;
        sop20 <<= 1 ;
    }
    if ( m == 16 ) m |= 0xfff0 ;
    res0 = m ;
    m = 0 ;
    while ( m < 16 && ( ( sop21 & 0x8000 ) == 0 ) ) {
        ++ m ;
        sop21 <<= 1 ;
    }
    if ( m == 16 ) m |= 0xfff0 ;
    res1 = m ;
    m = 0 ;
    while ( m < 16 && ( ( sop22 & 0x8000 ) == 0 ) ) {
        ++ m ;
        sop22 <<= 1 ;
    }
    if ( m == 16 ) m |= 0xfff0 ;
    res2 = m ;
    m = 0 ;
    while ( m < 16 && ( ( sop23 & 0x8000 ) == 0 ) ) {
        ++ m ;
        sop23 <<= 1 ;
    }
    if ( m == 16 ) m |= 0xfff0 ;
    res3 = m ;
    {
         int mres0 , mres1 , mres2 , mres3 , smin1 , smin2 ;
        mres0 = ( MASKV ( 0 ) == 1 ) ? res0 : 0xfff0 ;
        mres1 = ( MASKV ( 1 ) == 1 ) ? res1 : 0xfff0 ;
        mres2 = ( MASKV ( 2 ) == 1 ) ? res2 : 0xfff0 ;
        mres3 = ( MASKV ( 3 ) == 1 ) ? res3 : 0xfff0 ;
        smin1 = ( mres0 < mres1 ) ? mres0 : mres1 ;
        smin2 = ( mres2 < mres3 ) ? mres2 : mres3 ;
        SRES = ( smin1 < smin2 ) ? smin1 : smin2 ;
    }
    break ;
}
bres0 = res0 ;
bres1 = res1 ;
bres2 = res2 ;
bres3 = res3 ;
VRES = concat ( MASKV , bres0 , bres1 , bres2 , bres3 ) ;
if ( check_nz == 1 ) {
     int z = 0 , n = 0 ;
    if ( ( res0 & 0xffff ) == 0 ) z = 1 ;
    if ( ( res1 & 0xffff ) == 0 ) z |= 2 ;
    if ( ( res2 & 0xffff ) == 0 ) z |= 4 ;
    if ( ( res3 & 0xffff ) == 0 ) z |= 8 ;
    I_VFLAG_Z = z ;
    if ( res0 & 0x8000 ) n = 1 ;
    if ( res1 & 0x8000 ) n |= 2 ;
    if ( res2 & 0x8000 ) n |= 4 ;
    if ( res3 & 0x8000 ) n |= 8 ;
    I_VFLAG_N = n ;
}
if ( check_co == 1 ) {
     int c , o ;
    c = 0 ;
    c = cc_get_C ( valu_op , ( short  ) res0 , ( short  ) sop10 , ( short  ) sop20 ) ;
    c |= cc_get_C ( valu_op , ( short  ) res1 , ( short  ) sop11 , ( short  ) sop21 ) << 1 ;
```

```
        c |= cc_get_C ( valu_op , ( short  ) res2 , ( short  ) sop12 , ( short  ) sop22 ) << 2 ;
        c |= cc_get_C ( valu_op , ( short  ) res3 , ( short  ) sop13 , ( short  ) sop23 ) << 3 ;
        I_VFLAG_C = c ;
        o = cc_get_O ( valu_op , ( short  ) res0 , ( short  ) sop10 , ( short  ) sop20 ) ;
        o |= cc_get_O ( valu_op , ( short  ) res1 , ( short  ) sop11 , ( short  ) sop21 ) << 1 ;
        o |= cc_get_O ( valu_op , ( short  ) res2 , ( short  ) sop12 , ( short  ) sop22 ) << 2 ;
        o |= cc_get_O ( valu_op , ( short  ) res3 , ( short  ) sop13 , ( short  ) sop23 ) << 3 ;
        I_VFLAG_O = o ;
    }
    if ( check_clip == 1 ) {
         int u = 0 , l = 0 ;
        if ( res0 < minint ) {
            l = 1 ;
            if ( saturate ) res0 = minint ;
        } else if ( res0 > maxint ) {
            u = 1 ;
            if ( saturate ) res0 = maxint ;
        }
        if ( res1 < minint ) {
            l |= 2 ;
            if ( saturate ) res1 = minint ;
        } else if ( res1 > maxint ) {
            u |= 2 ;
            if ( saturate ) res1 = maxint ;
        }
        if ( res2 < minint ) {
            l |= 4 ;
            if ( saturate ) res2 = minint ;
        } else if ( res2 > maxint ) {
            u |= 1 ;
            if ( saturate ) res2 = maxint ;
        }
        if ( res3 < minint ) {
            l |= 8 ;
            if ( saturate ) res3 = minint ;
        } else if ( res3 > maxint ) {
            u |= 8 ;
            if ( saturate ) res3 = maxint ;
        }
        I_VFLAG_L = l ;
        I_VFLAG_U = u ;
    }
    bres0 = res0 ;
    bres1 = res1 ;
    bres2 = res2 ;
    bres3 = res3 ;
    VSAT = concat ( MASKV , bres0 , bres1 , bres2 , bres3 ) ;
    if ( CONFALU . SAT == 1 ) {
        if ( SRES . int32 (  ) < minint ) {
            SSAT = minint ;
        } else if ( SRES . int32 (  ) > maxint ) {
            SSAT = maxint ;
        } else {
            SSAT = SRES ( 15 , 0 ) ;
        }
    } else {
        SSAT = SRES ( 15 , 0 ) ;
    }
  }
```

The code above uses the following routines (directly or indirectly): cc_get_C, cc_get_O

View document source. Generated on: 2018/02/23 20:33:41 CET. Generated by prest release 0.3.40 from reStructuredText source.