]>

IPUS

Reference Manual

Table of Contents

1 Core: IPUS

1.1 Registers

1.1.1 *ARGMAX*

ARGMAX	
15	0

Reset value: 0xFFFF

Description:

Core register: position of maximum value of sorting tree, applyind MASK according to CONFSORT.IGNRMASK

1.1.2 *ARGMIN*

ARGMIN	
15	0

Reset value: 0xFFFF

Description:

Core register: Position of minimum value of sorting tree, applying MASK according to CONFSORT.IGNRMASK

1.1.3 *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

1.1.4 CLIPPED

CLIPPED
15 0

Description:

Core Register (read only): Adder Tree SUM clipped according to signed or unsigned mode

1.1.5 CONFADDT

SCALE	SHR	THR	SGN	IGNRMASK	SEL
15 9	8 5	4	3	2	1 0

Description:

Core register: Configuration selecting input to Adder Tree

Fields:

Field	Range	Description
SEL	[1,0]	Input selection: • 0: W • 1: INA • 2: WW • 3: INB
IGNRMASK	[2]	Ignore Mask: • 0: apply mask • 1: ignore mask
SGN	[3]	Signed/unsigned mode selection: • 0: unsigned • 1: signed
THR	[4]	Select threshold mode: • 0: keep lower • 1: keep higher
SHR	[8,5]	Shift right bits [8:5] after scaling: • right shift result by (SHR+6) positions
SCALE	[15,9]	Scale (multiply) Adder Tree result with SCALE: • res = SUM(20:0) * CONFADDT.SCALE) >> CONFADDT.SHR • CONFADDT.SHR = 0 and CONFADDT.SCALE = 0x40 is no scaling

1.1.6 CONFALU

OPIX0AUTOINC	OPIX1AUTOINC	ACCSHR	SHR		SAT	SGN
13	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.
OPIX1AUTOINC	[12]	Perform no automatic increment of OPIXA1 when OPIX1 is accessed for reading or writing (0=autoinc 1=no autoinc)
OPIX0AUTOINC	[13]	Perform no automatic increment of OPIXA0 when OPIX0 is accessed for reading or writing (0=autoinc 1=no autoinc)

1.1.7 CONFBEST

SGN	MIN
1	0

Reset value: 0

Description:

Core register: Configuration Register for best 5 of N accelerator. The configuration should be set only at the beginning of the line before any value was added into the best 5 of N

Fields:

Field	Range	Description
MIN	[0]	Min or max is best: • 0: max • 1: min
SGN	[1]	Signed/unsigned mode selection: • 0: unsigned • 1: signed

1.1.8 CONFHIST

SHR	OFFSET
11 8	7 0

Description:

Core Register: Configuration for HBININCH function

- HBININCL = (HBININCH>>SHR) + OFFSET

Fields:

Field	Range	Description
OFFSET	[7,0]	Offset being added to bin value
SHR	[11,8]	Shift right written value by by SHR bits before applying OFFSET and selecting bin to be incremented

1.1.9 CONFSORT

SGN	IGNRMASK	SEL
3	2	1 0

Description:

Core register: Configuration selecting input to Sorting Tree

Fields:

Field	Range	Description
SEL	[1,0]	Input selection: • 0: W • 1: INA • 2: WW • 3: INB
IGNRMASK	[2]	Ignore Mask: • 0: apply mask • 1: ignore mask
SGN	[3]	Signed/unsigned mode selection: • 0: unsigned • 1: signed

1.1.10 CONFSTAT

MODE_FIELD3	MODE_FIELD2	MODE_FIELD1	MODE_FIELD0
7 6	5 4	3 2	1 0

Description:

Core Register: Configuration for STATISTIC'S engine

Fields:

Field	Range	Description
MODE_FIELD0	[1,0]	Field 0 accumulation mode model: • 0 : maximum • 1 : minimum • 2 : unsigned accumulate • 3 : signed accumulate

Field	Range	Description
MODE_FIELD1	[3,2]	Field 1 accumulation mode model: • 0 : maximum • 1 : minimum • 2 : unsigned accumulate • 3 : signed accumulate
MODE_FIELD2	[5,4]	Field 2 accumulation mode model: • 0 : maximum • 1 : minimum • 2 : unsigned accumulate • 3 : signed accumulate
MODE_FIELD3	[7,6]	Field 3 accumulation mode model: • 0 : maximum • 1 : minimum • 2 : unsigned accumulate • 3 : signed accumulate

1.1.11 CONFTHRES

CONFTHRES	
15	0

Description:

Core register: Configuration selecting threshold value for Adder Tree threshold output. The threshold will be done according to CONFADDT.THR

1.1.12 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

1.1.13 HBINA

HBINA	
7	0

Description:

Core register: Histogram bin being processed

1.1.14 HBINADD

HBINADD	
15	0

Description:

Core Register: Write: value that is added to the Histogram bin HISTA is pointing to

1.1.15 HBININCH

HBININCH	
15	0

Description:

Core Register:

Shift and offset the written value (by CONFHIST) and increment histogram bin that has the resulting value

1.1.16 HBININCL

HBININCL	
7	0

Description:

Core Register: Increment histogram bin that has the written value

1.1.17 HD_BREAKENABLE

HD_BREAKENABLE	
7	0

Reset value: 0

Description:

Host Debug Register: Enable Break Conditions.

1.1.18 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)

1.1.19 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 (read-only)	[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)

1.1.20 H_DREGA

H_DREGA
7 0

Description:

Host Register: Data register address register

1.1.21 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

1.1.22 H_HISTA

H_HISTA
7 0

Description:

Host Register: Data register address register

1.1.23 H_HISTAMAX

H_HISTAMAX
7 0

Description:

Core Register: Binb of the Histogram peak value

1.1.24 H_HISTCL

H_HISTCL
0

Description:

Host Register: Clear all Histogram bins

1.1.25 H_HISTD

H_HISTD
31 0

Description:

Host Register: Data register data register

1.1.26 H_HISTPEAK

H_HISTPEAK
31 0

Description:

Core Register: Binb of the Histogram peak value

1.1.27 H_IMEMA

H_IMEMA
5 0

Description:

Host Register: IMEM address register

1.1.28 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

1.1.29 H_INACFG

CURRCFG						NEXTCFG			
[31,27]	CURRLINE2[26]	CURRLINE1[25]	CURRLINE0[24]	[23,19]	CURRNHOOD[18]	CURRSHIFT[17,16]	[15,11]	LINE2[10]	LINE1[9]	LINE0[
31						16	15			

Reset value: 0x00010001

Description:

Host Register: Input Matrix A configuration

Fields:

Field	Range	Description
NEXTCFG	[15,0]	Setup for next line to be processed (becomes valid with next write to H_START)
SHIFT (read-only)	[1,0]	constant value of 1 pixel

Field	Range	Description
NHOOD	[2]	next line neighborhood type • 0: 2D (3x3) • 1: horizontal (9x1)
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
CURRCFG (read-only)	[31,16]	Status of current line to be processed (read only)
CURRSHIFT	[17,16]	constant value of 1 pixel
CURRNHOOD	[18]	Current line neighborhood type • 0: 2D (3x3) • 1: horizontal (9x1)
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

1.1.30 H_INALCFG

CURRCFG							NEXTCFG			
[31,27]	CURRLINE2[26]	CURRLINE1[25]	CURRLINE0[24]	[23,19]	CURRNHOOD[18]	CURRSHIFT[17,16]	[15,11]	LINE2[10]	LINE1[9]	LINE0[
31						16	15			

Reset value: 0x00010001

Description:

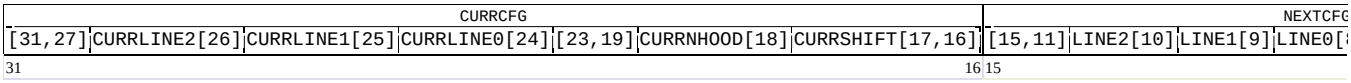
Host Register: Input Matrix Alpha configuration

Fields:

Field	Range	Description
NEXTCFG	[15,0]	Setup for next line to be processed (becomes valid with next write to H_START)
SHIFT (read-only)	[1,0]	constant value of 1 pixel
NHOOD	[2]	next line neighborhood type • 0: horizontal (9x1) • 1: 2D (3x3)

Field	Range	Description
LINE0	[8]	Indicates if line 0 is enabled for next line (for IN[0] to IN[2]) • 1: enabled
LINE1	[9]	Indicates if line 1 is enabled for next line (for IN[3] to IN[5]) • 1: enabled
LINE2	[10]	Indicates if line 2 is enabled for next line (for IN[6] to IN[8]) • 1: enabled
CURRCFG (read-only)	[31,16]	Status of current line to be processed (read only)
CURRSHIFT	[17,16]	constant value of 1 pixel
CURRNHOOD	[18]	Current line neighborhood type • 0: horizontal (9x1) • 1: 2D (3x3)
CURRLINE0	[24]	Indicates if currently line 0 is enabled • 1: enabled
CURRLINE1	[25]	Indicates if currently line 1 is enabled • 1: enabled
CURRLINE2	[26]	Indicates if currently line 2 is enabled • 1: enabled

1.1.31 H_INBCFG



Reset value: 0x00010001

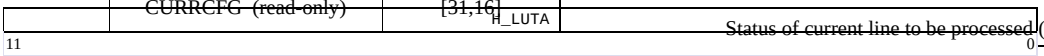
Description:

Host Register: Input Matrix B configuration

Fields:

Field	Range	Description
NEXTCFG	[15,0]	Setup for next line to be processed (becomes valid with next write to H_START)
SHIFT (read-only)	[1,0]	constant value of 1 pixel
NHOOD	[2]	next line neighborhood type • 0: 2D (3x3) • 1: horizontal (9x1)
LINE0	[8]	Indicates if line 0 is enabled for next line (for IN[0] to IN[2]) • 1: enabled
LINE1	[9]	Indicates if line 1 is enabled for next line (for IN[3] to IN[5]) • 1: enabled
LINE2	[10]	Indicates if line 2 is enabled for next line (for IN[6] to IN[8]) • 1: enabled
CURRCFG (read-only)	[31,16]	Status of current line to be processed (read only)
CURRSHIFT	[17,16]	constant value of 1 pixel
CURRNHOOD	[18]	Current line neighborhood type • 0: 2D (3x3) • 1: horizontal (9x1)
CURRLINE0	[24]	Indicates if currently line 0 is enabled • 1: enabled
CURRLINE1	[25]	Indicates if currently line 1 is enabled • 1: enabled
CURRLINE2	[26]	Indicates if currently line 2 is enabled • 1: enabled

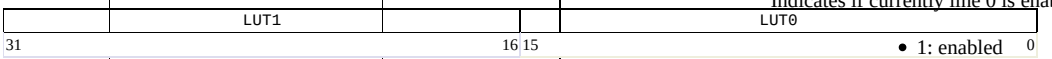
1.1.32 H_LUTA



Description:

Host Register: Look Up Table address register. Two 16 bit LUT words are in one 32 bit word of H_LUTD

1.1.33 H_LUTD



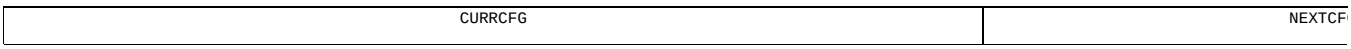
Description:

Host Register: Look Up Table data register. Any access to H_LUTD will progress H_LUTA by two LUT entries.

Fields:

Field	Range	Description
LUT0	[15,0]	First LUT entry (little endian)
LUT1	[31,16]	Second LUT entry (little endian)

1.1.34 H_OUTCFG



[31,28]	CURRLINE3[27]	CURRLINE2[26]	CURRLINE1[25]	CURRLINE0[24]	[23,18]	CURRSHIFT[17,16]	[15,12]	LINE3[11]	LINE2[10]	LINE1
31						16	15			

Reset value: 0x00010001

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 (read-only)	[1,0]	constant value of 1 pixel
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(1) • 1: enabled
LINE2	[10]	Indicates if line 2 is enabled for next line, connected to OUT(2) • 1: enabled
LINE3	[11]	Indicates if line 3 is enabled for next line, connected to OUT(3) • 1: enabled
CURRCFG (read-only)	[31,16]	Status of current line to be processed (read only)
CURRSHIFT	[17,16]	constant value of 1 pixel
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(1) • 1: enabled
CURRLINE2	[26]	Indicates if currently line 2 is enabled, connected to OUT(2) • 1: enabled
CURRLINE3	[27]	Indicates if currently line 3 is enabled, connected to OUT(3) • 1: enabled

1.1.35 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

1.1.36 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)

1.1.37 H_STATA

H_STATA
9 0

Description:

Host Register: Statistics fields address register (record x 4 + field)

1.1.38 H_STATCL

H_STATCL
0

Description:

Host Register: Clear all Statistics records

1.1.39 H_STATD

H_STATD
31 0

Description:

Host Register: Statistics register data register

1.1.40 H_STATUS

REMAINING		DEBUG	STATUS		EVENTS
			BUFFERED[9]	RUN[8]	
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

1.1.41 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
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)

1.1.42 IADDR

IADDR
7 … 0

Description:

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

1.1.43 I_ADDT

I_ADDT
20 … 0

Description:

Internal Signal: Adder Tree SUM full resolution

1.1.44 I_BUFFERED

I_BUFFERED
0

Reset value: 0

Description:

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

1.1.45 I_DEBUGMODE

I_DEBUGMODE
0

Reset value: 0

Description:

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

1.1.46 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

1.1.47 I_INACFG

CURRCFG						NEXTCFG			
[31,27]	CURRLINE2[26]	CURRLINE1[25]	CURRLINE0[24]	[23,19]	CURRNHOOD[18]	CURRSHIFT[17,16]	[15,11]	LINE2[10]	LINE1[9]	LINE0[
31 … 16							15			

Reset value: 0x00010001

Description:

Internal Register: Identical to H_INACFG (Input Matrix A 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]	constant value of 1 pixel
NHOOD	[2]	next line neighborhood type • 0: 2D (3x3) • 1: horizontal (9x1)
LINE0	[8]	Indicates if line 0 is enabled for next line (for IN[0] to IN[2]) • 1: enabled
LINE1	[9]	Indicates if line 1 is enabled for next line (for IN[3] to IN[5]) • 1: enabled
LINE2	[10]	Indicates if line 2 is enabled for next line (for IN[6] to IN[8]) • 1: enabled
CURRCFG	[31,16]	Status of current line to be processed (read only)
CURRSHIFT	[17,16]	constant value of 1 pixel
CURRNHOOD	[18]	Current line neighborhood type • 0: 2D (3x3) • 1: horizontal (9x1)
CURRLINE0	[24]	Indicates if currently line 0 is enabled (for IN[0] to IN[2]) • 1: enabled
CURRLINE1	[25]	Indicates if currently line 1 is enabled (for IN[3] to IN[5]) • 1: enabled
CURRLINE2	[26]	Indicates if currently line 2 is enabled (for IN[6] to IN[8]) • 1: enabled

1.1.48 I_INALCFG

CURRCFG							NEXTCFG			
[31,27]	CURRLINE2[26]	CURRLINE1[25]	CURRLINE0[24]	[23,19]	CURRNHOOD[18]	CURRSHIFT[17,16]	[15,11]	LINE2[10]	LINE1[9]	LINE0[
31						16	15			

Reset value: 0x00010001

Description:

Internal Register: Identical to H_INALCFG (Input Matrix A configuration)

Fields:

Field	Range	Description

Field	Range	Description
NEXTCFG	[15,0]	Setup for next line to be processed (becomes valid with next write to H_START)
SHIFT	[1,0]	constant value of 1 pixel
NHOOD	[2]	next line neighborhood type • 0: 2D (3x3) • 1: horizontal (9x1)
LINE0	[8]	Indicates if line 0 is enabled for next line • 1: enabled
LINE1	[9]	Indicates if line 1 is enabled for next line • 1: enabled
LINE2	[10]	Indicates if line 2 is enabled for next line • 1: enabled
CURRCFG	[31,16]	Status of current line to be processed (read only)
CURRSHIFT	[17,16]	constant value of 1 pixel
CURRNHOOD	[18]	Current line neighborhood type • 0: horizontal (9x1) • 1: 2D (3x3)
CURRLINE0	[24]	Indicates if currently line 0 is enabled • 1: enabled
CURRLINE1	[25]	Indicates if currently line 1 is enabled • 1: enabled
CURRLINE2	[26]	Indicates if currently line 2 is enabled • 1: enabled

1.1.49 I_INBCFG

CURRCFG							NEXTCFG			
[31,27]	CURRLINE2[26]	CURRLINE1[25]	CURRLINE0[24]	[23,19]	CURRNHOOD[18]	CURRSHIFT[17,16]	[15,11]	LINE2[10]	LINE1[9]	LINE0[
31						16	15			

Reset value: 0x00010001

Description:

Internal Register: Identical to H_INBCFG (Input Matrix A 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]	constant value of 1 pixel

Field	Range	Description
NHOOD	[2]	next line neighborhood type • 0: 2D (3x3) • 1: horizontal (9x1)
LINE0	[8]	Indicates if line 0 is enabled for next line • 1: enabled
LINE1	[9]	Indicates if line 1 is enabled for next line • 1: enabled
LINE2	[10]	Indicates if line 2 is enabled for next line • 1: enabled
CURRCFG	[31,16]	Status of current line to be processed (read only)
CURRSHIFT	[17,16]	constant value of 1 pixel
CURRNHOOD	[18]	Current line neighborhood type • 0: horizontal (9x1) • 1: 2D (3x3)
CURRLINE0	[24]	Indicates if currently line 0 is enabled • 1: enabled
CURRLINE1	[25]	Indicates if currently line 1 is enabled • 1: enabled
CURRLINE2	[26]	Indicates if currently line 2 is enabled • 1: enabled

1.1.50 I_MFLAG_C

I_MFLAG_C	
8	0

Description:

Internal Signal: 3x3 Matrix ALU Carry Flags

1.1.51 I_MFLAG_N

I_MFLAG_N	
8	0

Description:

Internal Signal: 3x3 Matrix ALU Negative Flags

1.1.52 I_MFLAG_O

I_MFLAG_O	
8	0

Description:

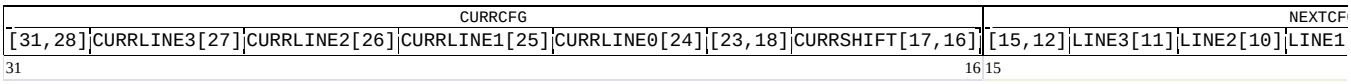
Internal Signal: 3x3 Matrix ALU Overflow Flags

1.1.53 I_MFLAG_Z

Description:

Internal Signal: 3x3 Matrix ALU Zero Flags

1.1.54 I_OUTCFG



Reset value: 0x00010001

Description:

Internal Register: Identical to H_OUTCFG (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]	constant value of 1 pixel
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(1) • 1: enabled
LINE2	[10]	Indicates if line 2 is enabled for next line, connected to OUT(2) • 1: enabled
LINE3	[11]	Indicates if line 3 is enabled for next line, connected to OUT(3) • 1: enabled
CURRCFG	[31,16]	Status of current line to be processed (read only)
CURRSHIFT	[17,16]	constant value of 1 pixel
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(1) • 1: enabled
CURRLINE2	[26]	Indicates if currently line 2 is enabled, connected to OUT(2) • 1: enabled

Field	Range	Description
CURRLINE3	[27]	Indicates if currently line 3 is enabled, connected to OUT(3) • 1: enabled

1.1.55 I_RUN

I_RUN
0

Reset value: 0

Description:

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

1.1.56 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"

1.1.57 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)

1.1.58 I_STAT_DIRTY

I_STAT_DIRTY
0

Reset value: 0

Description:

Internal Register: Statistics fields have been modified since read and need write back

1.1.59 I_STAT_NEW_LINE

I_STAT_NEW_LINE
0

Reset value: 1

Description:

Internal Signal: STAT value has not been read in that line, so do a read even if the STATA is the same

1.1.60 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)

1.1.61 LOCK

LUT	BEST5OFN	STAT	HIST	OUT	GPR12TO15		GPR8TO11		GPR4TO7		GPR0TO3	
12	11	10	9	8	7	6	5	4	3	2	1	0

Reset value: 0x1fff

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 • 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)

Field	Range	Description
OUT	[8]	Lock status for OUT0-OUT3 • 0: allow read access to Out • 1: deny
HIST	[9]	Lock status for H_HISTA and H_HISTD • 0: allow read access to Histogram host registers • 1: deny
STAT	[10]	Lock status for H_STATA and H_STATD • 0: allow read access to Statistic's host registers • 1: deny
BEST5OFN	[11]	Lock status for SOF0-4 and SOFA0-5 via the H_DREGD register • 0: allow read access • 1: deny
LUT	[12]	Lock status for H_LUTA and H_LUTD • 0: allow read access to Look Up Table host registers • 1: deny

1.1.62 LOOPCNT

LOOPCNT	
15	0

Description:

Core Register: Loop counter

1.1.63 LOOPCNT1

LOOPCNT1	
15	0

Description:

Core Register: 2nd Loop counter

1.1.64 LUT

LUT	
15	0

Description:

Core register: LUT (Look Up Table) entry being accessed at entry LUTA

- read: LUT = LookUpTable[LUTA]
- write: LookUpTable[LUTA] = LUT

1.1.65 LUTA

LUTA	
11	0

Reset value: 0

Description:

Core register: Address into Look Up Table

1.1.66 MASKOP2

MASKOP2	
15	0

Description:

Internal Signal: Bit vector operand 2, input to Mask ALU

On a write

```
func ( bits < 16 > val ) {
 MASKOP2 = val ;
}
```

1.1.67 MASKV

MASKV	
8	0

Reset value: 0x1FF

Description:

Core Register: Mask 3x3 matrix register vector (MASK(i) is equal to MASKV[i])

1.1.68 MFLAG_C

MFLAG_C	
8	0

Description:

Core Register: 3x3 Matrix ALU Carry Flags

1.1.69 MFLAG_N

MFLAG_N	
8	0

Description:

Core Register: 3x3 Matrix ALU Negative Flags

1.1.70 MFLAG_O

MFLAG_O	
8	0

Description:

Core Register: 3x3 Matrix ALU Overflow Flags

1.1.71 MFLAG_Z

MFLAG_Z

8	0

Description:

Core Register: 3x3 Matrix ALU Zero Flags

1.1.72 NHOOD

NHOOD	
8	0

Reset value: 0x1FF

Description:

Core Register: 9 bit vector to mask out Mask ALU results for special neighborhood, e.g.:

- CON_4 = 0b 010101010
- CON_8 = 0b 111111111
- CON H2 = 0b 000111000
- CON V2 = 0b 010010010

1.1.73 NIA

NIA	
7	0

Description:

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

1.1.74 ONE

ONE	
15	0

Reset value: 0xffff

Description:

Core Register: -1 constant, read only

1.1.75 OP1

OP1	
16	0

Description:

Internal Signal: ALU Operand 1

1.1.76 OP2

OP2	
16	0

Description:

Internal Signal: ALU Operand 2

1.1.77 PRNG

PRNG
15 0

Description:

Core Register: Pseudo Random Noise Value (16 LSB's of the internal PRNG)

1.1.78 RES

RES
17 0

Description:

Internal Signal: ALU Result value before saturating

1.1.79 RES32

RES32
32 0

Description:

Internal Signal: Result from multiplier

1.1.80 SAT

SAT
15 0

Description:

Internal Signal: ALU Result value after saturating

1.1.81 SCALED

SCALED
15 0

Description:

Core Register (read only): Adder Tree SUM scaled according to configured scale and scale-shift

1.1.82 SCNT

SCNT
15 0

Description:

Internal register: Count inputs to the Sort engine (best 5 of N). This counter value is stored with the value in the SOA registers

1.1.83 SIG_LINE_DONE

SIG_LINE_DONE
0

Description:

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

1.1.84 SOCLEAR

SOCLEAR
0

Description:

Core register: Initialize/Reset the Sort engine (best 5 of N).

1.1.85 SOIN

SOIN	
15	0

Description:

Core register: Sorter Input (best 5 of N) Maximum value of sorting tree

1.1.86 SORTMAX

SORTMAX	
15	0

Description:

Core register: Maximum value of sorting tree applying MASK according to CONFSORT.IGNRMASK

1.1.87 SORTMEDIAN

SORTMEDIAN	
15	0

Description:

Core register: Median value of sorting tree applying MASK according to CONFSORT.IGNRMASK

1.1.88 SORTMIN

SORTMIN	
15	0

Description:

Core register: Minimum value of adder tree applying MASK according to CONFSORT.IGNRMASK

1.1.89 SREC

SREC	
7	0

Description:

Core register: Statistics record being processed

1.1.90 SUM

SUM
15 0

Description:

Core Register (read only): Adder Tree SUM

1.1.91 THRESHOLD

THRESHOLD
15 0

Description:

Core Register (read only): Adder Tree SUM thresholded according to CONFTHRES, CONFADDT.SGN and CONFADDT.THR

1.1.92 XPOS

XPOS
15 0

Description:

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

1.1.93 YPOS

YPOS
15 0

Description:

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

1.1.94 ZERO

ZERO
15 0

Description:

Core Register: 0 constant, read only

1.2 Register Files

1.2.1 ACC

ACC
15 0

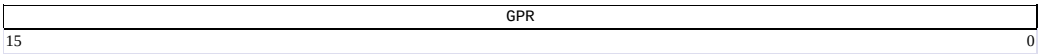
Number of registers in file: 4

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.

1.2.2 GPR

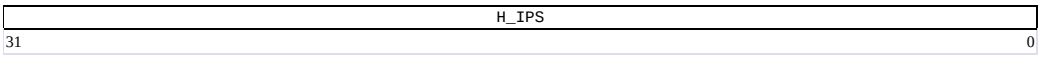


Number of registers in file: 16

Description:

Core Register: General purpose registers

1.2.3 H_IPS



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_INACFG
7	H_INBCFG
8	H_INALCFG
9	H_OUTCFG
16	H_IMEMA
17	H_IMEMD
18	H_DREGA
19	H_DREGD
20	H_HISTA
21	H_HISTD
22	H_HISTPEAK
23	H_HISTAMAX
24	H_HISTCL
25	H_STATA
26	H_STATD
27	H_STATCL
30	H_LUTA
31	H_LUTD
57	IADDR
58	CC_ZNCO
65	S_LINELEN_INA0
66	S_CHCFG_INA0
67	S_LINE_PTR_INA0
68	S_LINELEN_INA1
69	S_CHCFG_INA1
70	S_LINE_PTR_INA1
71	S_LINELEN_INA2
72	S_CHCFG_INA2
73	S_LINE_PTR_INA2
81	S_LINELEN_INB0
82	S_CHCFG_INB0

Index	Register Usage
83	S_LINE_PTR_INB0
84	S_LINELEN_INB1
85	S_CHCFG_INB1
86	S_LINE_PTR_INB1
87	S_LINELEN_INB2
88	S_CHCFG_INB2
89	S_LINE_PTR_INB2
97	S_LINELEN_INAL0
98	S_CHCFG_INAL0
99	S_LINE_PTR_INAL0
100	S_LINELEN_INAL1
101	S_CHCFG_INAL1
102	S_LINE_PTR_INAL1
103	S_LINELEN_INAL2
104	S_CHCFG_INAL2
105	S_LINE_PTR_INAL2
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
122	S_LINELEN_OUT3
123	S_CHCFG_OUT3
124	S_LINE_PTR_OUT3
129	S_CURRLINELEN_INA0
130	S_CURRCHCFG_INA0
131	S_CURRLINE_PTR_INA0
132	S_CURRLINELEN_INA1
133	S_CURRCHCFG_INA1
134	S_CURRLINE_PTR_INA1
135	S_CURRLINELEN_INA2
136	S_CURRCHCFG_INA2
137	S_CURRLINE_PTR_INA2
145	S_CURRLINELEN_INB0
146	S_CURRCHCFG_INB0
147	S_CURRLINE_PTR_INB0
148	S_CURRLINELEN_INB1
149	S_CURRCHCFG_INB1
150	S_CURRLINE_PTR_INB1
151	S_CURRLINELEN_INB2
152	S_CURRCHCFG_INB2
153	S_CURRLINE_PTR_INB2
161	S_CURRLINELEN_INAL0
162	S_CURRCHCFG_INAL0
163	S_CURRLINE_PTR_INAL0
164	S_CURRLINELEN_INAL1
165	S_CURRCHCFG_INAL1
166	S_CURRLINE_PTR_INAL1
167	S_CURRLINELEN_INAL2
168	S_CURRCHCFG_INAL2
169	S_CURRLINE_PTR_INAL2
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

Index	Register Usage
186	S_CURRLINELEN_OUT3
187	S_CURRCHCFG_OUT3
188	S_CURRLINE_PTR_OUT3

1.2.4 INA

INA	
15	0

Number of registers in file: 9

Description:

Core Register: Input A matrix register for pixels

- INA[2] INA[1] INA[0]
- INA[5] INA[4] INA[3]
- INA[8] INA[7] INA[6]
- (numbering for scan from left to right)

1.2.5 INALPHA

INALPHA	
15	0

Number of registers in file: 9

Description:

Core Register: Input A 3x3 matrix register for pixels

- INALPHA[2] INALPHA[1] INALPHA[0]
- INALPHA[5] INALPHA[4] INALPHA[3]
- INALPHA[8] INALPHA[7] INALPHA[6]
- (numbering for scan from left to right)

1.2.6 INB

INB	
15	0

Number of registers in file: 9

Description:

Core Register: Input A 3x3 matrix register for pixels

- INB[2] INB[1] INB[0]
- INB[5] INB[4] INB[3]
- INB[8] INB[7] INB[6]
- (numbering for scan from left to right)

1.2.7 I_POS_INA

I_POS_INA	
17	0

Number of registers in file: 3

Description:

Internal Register: Current pixel position in the line

1.2.8 I_POS_INAL

I_POS_INAL	
17	0

Number of registers in file: 3

Description:

Internal Register: Current pixel position in the line

1.2.9 I_POS_INB

I_POS_INB	
17	0

Number of registers in file: 3

Description:

Internal Register: Current pixel position in the line

1.2.10 I_POS_OUT

I_POS_OUT	
17	0

Number of registers in file: 4

Description:

Internal Register: Current byte position in the line

1.2.11 I_RPTCNT_INA

I_RPTCNT_INA	
1	0

Number of registers in file: 3

Reset value: 1

Description:

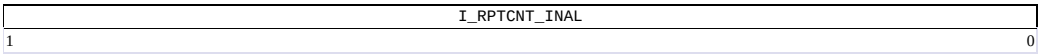
Internal Register: counting the repeat status

On a read

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

```
}
```

1.2.12 I_RPTCNT_INAL



Number of registers in file: 3

Reset value: 1

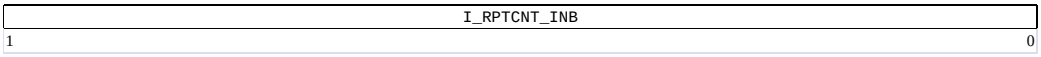
Description:

Internal Register: counting the repeat status

On a read

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

1.2.13 I_RPTCNT_INB



Number of registers in file: 3

Reset value: 1

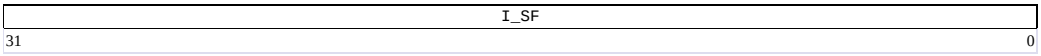
Description:

Internal Register: counting the repeat status

On a read

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

1.2.14 I_SF



Number of registers in file: 4

Description:

Internal Core Register: Statitics field: 32 bit value of concatenating SFH(i) and SFL(i)

1.2.15 I_SKIPCNT_OUT

I_SKIPCNT_OUT	
1	0

Number of registers in file: 5

Reset value: 0

Description:

Internal Register: counting the skip status

1.2.16 MASK

MASK
0

Number of registers in file: 9

Reset value: 1

Description:

Core Register: Mask 3x3 matrix register for pixels

1.2.17 MASKOP1

MASKOP1	
15	0

Number of registers in file: 9

Description:

Internal Signal: Matrix operand 1, input to Mask ALU

On a write

```
func ( unsigned index , bits < 16 > val ) {
  int i ;
 switch ( index ) {
     case 0 : for ( i = 0 ; i < 9 ; i ++ ) {
         MASKOP1 ( i ) = W ( i ) ;
     }
     break ;
     case 1 : for ( i = 0 ; i < 9 ; i ++ ) {
         MASKOP1 ( i ) = INALPHA ( i ) ;
     }
     break ;
     case 2 : for ( i = 0 ; i < 9 ; i ++ ) {
         MASKOP1 ( i ) = WW ( i ) ;
     }
     break ;
     case 5 : for ( i = 0 ; i < 9 ; i ++ ) {
         MASKOP1 ( i ) = MASK ( i ) ;
     }
     break ;
 }
}
```

1.2.18 MASKRES

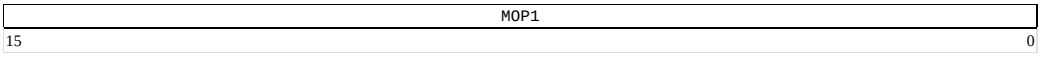
MASKRES
0

Number of registers in file: 9

Description:

Internal Signal: Matrix result of Mask ALU before saturation

1.2.19 MOP1

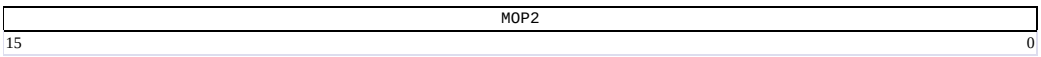


Number of registers in file: 9

Description:

Internal Signal: 3x3 Matrix Operand 1 Input to Matrix ALU

1.2.20 MOP2

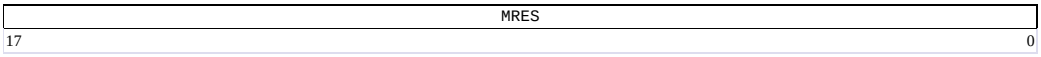


Number of registers in file: 9

Description:

Internal Signal: 3x3 Matrix Operand 2 Input to Matrix ALU

1.2.21 MRES

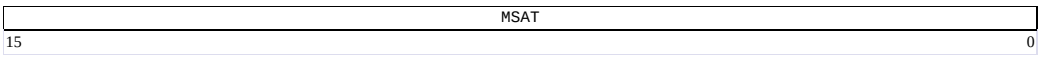


Number of registers in file: 9

Description:

Internal Signal: 3x3 Matrix result of Matrix ALU before saturation

1.2.22 MSAT

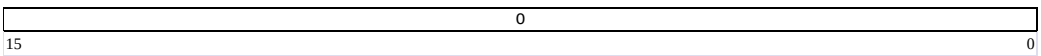


Number of registers in file: 9

Description:

Internal Signal: 3x3 Matrix result of Matrix ALU after saturation

1.2.23 O



Number of registers in file: 4

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).

1.2.24 OPIX

OPIX	
15	0

Number of registers in file: 2

Description:

Core Register: Indirect operand register.

1.2.25 OPIXA

OPIXA	
7	0

Number of registers in file: 2

Description:

Core Register: Indirect operand address register.

1.2.26 OUT

OUT	
15	0

Number of registers in file: 4

Description:

Core Register: Output pixel registers.

1.2.27 P_INA

P_INA	
15	0

Number of registers in file: 3

Description:

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

1.2.28 P_INAL

P_INAL	
15	0

Number of registers in file: 3

Description:

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

1.2.29 P_INB

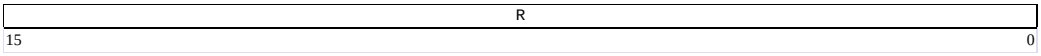
P_INB	
15	0

Number of registers in file: 3

Description:

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

1.2.30 R



Number of registers in file: 256

Description:

Core Register: All registers

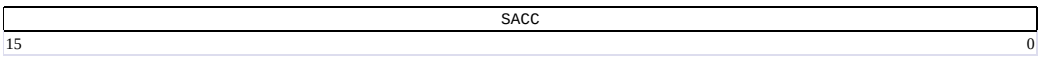
Constituent Registers

Index	Register Usage
0	LOOPCNT1
1	LOOPCNT
2	XPOS
3	YPOS
4	CONFALU
8	CONFADDT
9	CONFTHRES
10	CONFSORT
11	CONFBEST
12	CONFHIST
13	CONFSTAT
16	INA0
17	INA1
18	INA2
19	INA3
20	INA4
21	INA5
22	INA6
23	INA7
24	INA8
32	INB0
33	INB1
34	INB2
35	INB3
36	INB4
37	INB5
38	INB6
39	INB7
40	INB8
48	INALPHA0
49	INALPHA1
50	INALPHA2
51	INALPHA3
52	INALPHA4
53	INALPHA5
54	INALPHA6
55	INALPHA7
56	INALPHA8
64	W0
65	W1
66	W2
67	W3
68	W4
69	W5
70	W6

Index	Register Usage
71	W7
72	W8
80	WW0
81	WW1
82	WW2
83	WW3
84	WW4
85	WW5
86	WW6
87	WW7
88	WW8
96	MASK0
97	MASK1
98	MASK2
99	MASK3
100	MASK4
101	MASK5
102	MASK6
103	MASK7
104	MASK8
105	MASKV
106	NHOOD
108	MFLAG_O
109	MFLAG_C
110	MFLAG_N
111	MFLAG_Z
112	GPR0
113	GPR1
114	GPR2
115	GPR3
116	GPR4
117	GPR5
118	GPR6
119	GPR7
120	GPR8
121	GPR9
122	GPR10
123	GPR11
124	GPR12
125	GPR13
126	GPR14
127	GPR15
128	OUT0
129	OUT1
130	OUT2
131	OUT3
144	OPIXA0
145	OPIX0
146	OPIXA1
147	OPIX1
152	ACC0
153	ACC1
154	ACC2
155	ACC3
156	SACC0
157	SACC1
158	SACC2
159	SACC3
160	SUM
161	CLIPPED
162	THRESHOLD
163	SCALED
164	SORTMAX
165	SORTMIN

Index	Register Usage
166	SORTMEDIAN
167	ARGMIN
168	ARGMAX
208	LUTA
209	LUT
212	HBINA
213	HBINADD
214	HBININCL
215	HBININCH
216	SFL0
217	SFH0
218	SFL1
219	SFH1
220	SFL2
221	SFH2
222	SFL3
223	SFH3
224	SREC
240	SOIN
241	SOCLEAR
242	SO0
243	SO1
244	SO2
245	SO3
246	SO4
247	SOA0
248	SOA1
249	SOA2
250	SOA3
251	SOA4
252	LOCK
253	PRNG
254	ONE
255	ZERO

1.2.31 *SACC*

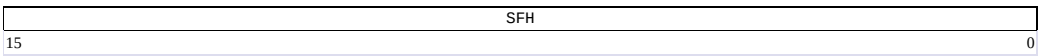


Number of registers in file: 4

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.

1.2.32 *SFH*



Number of registers in file: 4

Description:

Core Register: Statitics field: upper 16 bits (read only)

1.2.33 *SFL*

Number of registers in file: 4

Description:

Core Register: Statitics field: lower 16 bit, Writing to the the register will execute the action as specified in CONFSTAT on the high and low word of the field (sfhX,sflX with X = field number)

1.2.34 SO

SO
16 0

Number of registers in file: 5

Reset value: 0x10000

Description:

Core register: Best values accoriung to CONFBEST. These values are read only

1.2.35 SOA

SOA
15 0

Number of registers in file: 5

Reset value: 0xFFFF

Description:

Core register: Corrsponding arguments (SCNT) of the best values (SO). These values are read only

1.2.36 S_CHCFG_INA

	BUFFERTYPE	TRANSFORM	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: 3

Reset value: 0x50000

Description:

StreamIF Host Register: Configuration for input streams at INA 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

Field	Range	Description
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) • n: will present n more pixels at the beginning of the line. The (n+1)th pixel is the first valid pixel. The n first pixels are treated according to the REPLACEMODE
TRANSFORM	[27]	Apply coordiante transformation mode • 0: linear scan • 1: transformation
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

1.2.37 S_CHCFG_INAL

	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: 3

Reset value: 0x50000

Description:

StreamIF Host Register: Configuration for input streams at INAL 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

Field	Range	Description
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) • n: will present n more pixels at the beginning of the line. The (n+1)th pixel is the first valid pixel. The n 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

1.2.38 S_CHCFG_INB

	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: 3

Reset value: 0x50000

Description:

StreamIF Host Register: Configuration for input streams at INB 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

Field	Range	Description
		• 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) • n: will present n more pixels at the beginning of the line. The (n+1)th pixel is the first valid pixel. The n 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

1.2.39 S_CHCFG_OUT

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

Number of registers in file: 4

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

1.2.40 S_CURRCHCFG_INA

	BUFFERTYPE	TRANSFORM	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: 3

Description:

StreamIF Host Register: Active configuration status for input streams at INA (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 • 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) • n: will present n more pixels at the beginning of the line. The (n+1)th pixel is the first valid pixel. The n first pixels are treated according to the REPLACEMODE
TRANSFORM	[27]	Apply coordiante transformation mode • 0: linear scan • 1: transformation
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

1.2.41 S_CURRCHCFG_INAL

	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: 3

Description:

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

Fields:

Field	Range	Description
REPLAVEVALUE	[15,0]	Value to use for pixels outside the valid image area

Field	Range	Description
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) • n: will present n more pixels at the beginning of the line. The (n+1)th pixel is the first valid pixel. The n 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

1.2.42 S_CURRCHCFG_INB

	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: 3

Description:

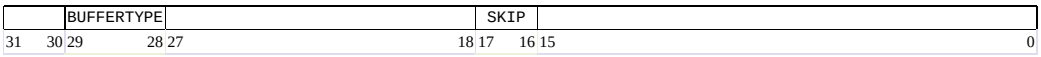
StreamIF Host Register: Active configuration status for input streams at INB (read onyl)

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

Field	Range	Description
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) • n: will present n more pixels at the beginning of the line. The (n+1)th pixel is the first valid pixel. The n 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

1.2.43 S_CURRCHCFG_OUT



Number of registers in file: 4

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

1.2.44 S_CURRLINELEN_INA

S_CURRLINELEN_INA	
15	0

Number of registers in file: 3

Reset value: 0xFFFF

Description:

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

1.2.45 S_CURRLINELEN_INAL

S_CURRLINELEN_INAL	
15	0

Number of registers in file: 3

Reset value: 0xFFFF

Description:

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

1.2.46 S_CURRLINELEN_INB

S_CURRLINELEN_INB	
15	0

Number of registers in file: 3

Reset value: 0xFFFF

Description:

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

1.2.47 S_CURRLINELEN_OUT

S_CURRLINELEN_OUT	
15	0

Number of registers in file: 4

Reset value: 0xFFFF

Description:

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

1.2.48 S_CURRLINE_PTR_INA

S_CURRLINE_PTR_INA	
31	0

Number of registers in file: 3

Description:

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

1.2.49 S_CURRLINE_PTR_INAL

S_CURRLINE_PTR_INAL	
31	0

Number of registers in file: 3

Description:

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

1.2.50 S_CURRLINE_PTR_INB

S_CURRLINE_PTR_INB	
31	0

Number of registers in file: 3

Description:

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

1.2.51 S_CURRLINE_PTR_OUT

S_CURRLINE_PTR_OUT	
31	0

Number of registers in file: 4

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 PTR_OUT(" , dec , line_no , '
 }
}
```

1.2.52 S_LINELEN_INA

S_LINELEN_INA	
15	0

Number of registers in file: 3

Reset value: 0xFFFF

Description:

StreamIF Host Register: Line length in pixels

1.2.53 S_LINELEN_INAL

S_LINELEN_INAL	
15	0

Number of registers in file: 3

Reset value: 0xFFFF

Description:

StreamIF Host Register: Line length in pixels

1.2.54 S_LINELEN_INB

S_LINELEN_INB	
15	0

Number of registers in file: 3

Reset value: 0xFFFF

Description:

StreamIF Host Register: Line length in pixels

1.2.55 S_LINELEN_OUT

S_LINELEN_OUT	
15	0

Number of registers in file: 4

Reset value: 0xFFFF

Description:

StreamIF Host Register: Line length in pixels

1.2.56 S_LINE_PTR_INA

S_LINE_PTR_INA	
31	0

Number of registers in file: 3

Description:

StreamIF Host Register: Line Start Addresses

1.2.57 S_LINE_PTR_INAL

S_LINE_PTR_INAL	
31	0

Number of registers in file: 3

Description:

StreamIF Host Register: Line Start Addresses

1.2.58 S_LINE_PTR_INB

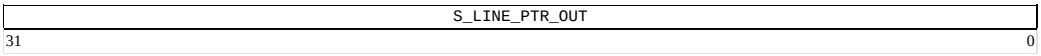
S_LINE_PTR_INB	
31	0

Number of registers in file: 3

Description:

StreamIF Host Register: Line Start Addresses

1.2.59 S_LINE_PTR_OUT

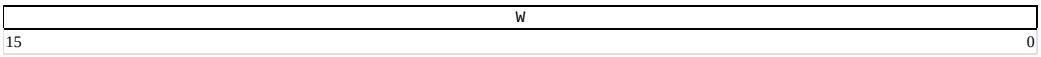


Number of registers in file: 4

Description:

StreamIF Host Register: Line Start Addresses

1.2.60 W



Number of registers in file: 9

Description:

Core Register: Working 3x3 matrix register for pixels

1.2.61 WW

Number of registers in file: 9

Description:

Core Register: Working 3x3 matrix 2 register for pixels

1.3 Instruction Fields

C D I L M O R T

1.3.1 CLZBIT

Size	1
Bits	[4,4]

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
confmxalu	7
confaddt	8
confthres	9
confsort	10
confbest	11
confhist	12
confstat	13
reserved	14
reserved	15
ina0	16
ina1	17
ina2	18
ina3	19
ina4	20
ina5	21
ina6	22
ina7	23
ina8	24
reserved	25
reserved	26
reserved	27
reserved	28
reserved	29
reserved	30
reserved	31
inb0	32
inb1	33
inb2	34
inb3	35
inb4	36
inb5	37
inb6	38
inb7	39
inb8	40

reserved	41
reserved	42
reserved	43
reserved	44
reserved	45
reserved	46
reserved	47
inalpha0	48
inalpha1	49
inalpha2	50
inalpha3	51
inalpha4	52
inalpha5	53
inalpha6	54
inalpha7	55
inalpha8	56
reserved	57
reserved	58
reserved	59
reserved	60
reserved	61
reserved	62
reserved	63
w0	64
w1	65
w2	66
w3	67
w4	68
w5	69
w6	70
w7	71
w8	72
reserved	73
reserved	74
reserved	75
reserved	76
reserved	77
reserved	78
reserved	79
ww0	80
ww1	81
ww2	82
ww3	83
ww4	84
ww5	85
ww6	86
ww7	87
ww8	88
reserved	89
reserved	90
reserved	91
reserved	92
reserved	93
reserved	94
reserved	95
m0	96
m1	97
m2	98
m3	99
m4	100
m5	101
m6	102
m7	103
m8	104
maskv	105
nhood	106
reserved	107
mflag_o	108

mflag_c	109
mflag_n	110
mflag_z	111
gpr0	112
gpr1	113
gpr2	114
gpr3	115
gpr4	116
gpr5	117
gpr6	118
gpr7	119
gpr8	120
gpr9	121
gpr10	122
gpr11	123
gpr12	124
gpr13	125
gpr14	126
gpr15	127
out0	128
out1	129
out2	130
out3	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
opixa0	144
opix0	145
opixa1	146
opix1	147
reserved	148
reserved	149
reserved	150
reserved	151
acc0	152
acc1	153
acc2	154
acc3	155
sacc0	156
sacc1	157
sacc2	158
sacc3	159
sum	160
clipped	161
threshold	162
scaled	163
max	164
sortmax	164
min	165
sortmin	165
median	166
sortmedian	166
argmin	167
argmax	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
luta	208
lut	209
reserved	210
reserved	211
hbina	212
hbinadd	213
hbinincl	214
hbininch	215
sfl0	216
sfh0	217
sfl1	218
sfh1	219
sfl2	220
sfh2	221
sfl3	222
sfh3	223
srec	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
soin	240
soclear	241

so0	242
so1	243
so2	244
so3	245
so4	246
soa0	247
soa1	248
soa2	249
soa3	250
soa4	251
lock	252
prng	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
confmxalu	7
confaddt	8
confthres	9
confsort	10
confbest	11
confhist	12
confstat	13
reserved	14
reserved	15
ina0	16
ina1	17
ina2	18
ina3	19
ina4	20
ina5	21
ina6	22
ina7	23
ina8	24
reserved	25
reserved	26
reserved	27
reserved	28
reserved	29
reserved	30
reserved	31
inb0	32
inb1	33
inb2	34
inb3	35
inb4	36
inb5	37
inb6	38
inb7	39
inb8	40
reserved	41

reserved	42
reserved	43
reserved	44
reserved	45
reserved	46
reserved	47
inalpha0	48
inalpha1	49
inalpha2	50
inalpha3	51
inalpha4	52
inalpha5	53
inalpha6	54
inalpha7	55
inalpha8	56
reserved	57
reserved	58
reserved	59
reserved	60
reserved	61
reserved	62
reserved	63
w0	64
w1	65
w2	66
w3	67
w4	68
w5	69
w6	70
w7	71
w8	72
reserved	73
reserved	74
reserved	75
reserved	76
reserved	77
reserved	78
reserved	79
ww0	80
ww1	81
ww2	82
ww3	83
ww4	84
ww5	85
ww6	86
ww7	87
ww8	88
reserved	89
reserved	90
reserved	91
reserved	92
reserved	93
reserved	94
reserved	95
m0	96
m1	97
m2	98
m3	99
m4	100
m5	101
m6	102
m7	103
m8	104
maskv	105
nhood	106
reserved	107
mflag_o	108
mflag_c	109

mflag_n	110
mflag_z	111
gpr0	112
gpr1	113
gpr2	114
gpr3	115
gpr4	116
gpr5	117
gpr6	118
gpr7	119
gpr8	120
gpr9	121
gpr10	122
gpr11	123
gpr12	124
gpr13	125
gpr14	126
gpr15	127
out0	128
out1	129
out2	130
out3	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
opixa0	144
opix0	145
opixa1	146
opix1	147
reserved	148
reserved	149
reserved	150
reserved	151
acc0	152
acc1	153
acc2	154
acc3	155
sacc0	156
sacc1	157
sacc2	158
sacc3	159
sum	160
clipped	161
threshold	162
scaled	163
max	164
sortmax	164
min	165
sortmin	165
median	166
sortmedian	166
argmin	167
argmax	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
luta	208
lut	209
reserved	210
reserved	211
hbina	212
hbinadd	213
hbinincl	214
hbininch	215
sfl0	216
sfh0	217
sfl1	218
sfh1	219
sfl2	220
sfh2	221
sfl3	222
sfh3	223
srec	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
soin	240
soclear	241
so0	242

so1	243
so2	244
so3	245
so4	246
soa0	247
soa1	248
soa2	249
soa3	250
soa4	251
lock	252
prng	253
one	254
zero	255

1.3.5 *IM16*

Size	16
Bits	[15,0]

Description:

Operator 2 immediate 16 bits

1.3.6 *IM4*

Size	4
Bits	[3,0]

Description:

Operator 2 immediate 4 bits

1.3.7 *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
confmxalu	7
confaddt	8
confthres	9
confsort	10
confbest	11
confhist	12
confstat	13
reserved	14
reserved	15
ina0	16
ina1	17
ina2	18
ina3	19
ina4	20

ina5	21
ina6	22
ina7	23
ina8	24
reserved	25
reserved	26
reserved	27
reserved	28
reserved	29
reserved	30
reserved	31
inb0	32
inb1	33
inb2	34
inb3	35
inb4	36
inb5	37
inb6	38
inb7	39
inb8	40
reserved	41
reserved	42
reserved	43
reserved	44
reserved	45
reserved	46
reserved	47
inalpha0	48
inalpha1	49
inalpha2	50
inalpha3	51
inalpha4	52
inalpha5	53
inalpha6	54
inalpha7	55
inalpha8	56
reserved	57
reserved	58
reserved	59
reserved	60
reserved	61
reserved	62
reserved	63
w0	64
w1	65
w2	66
w3	67
w4	68
w5	69
w6	70
w7	71
w8	72
reserved	73
reserved	74
reserved	75
reserved	76
reserved	77
reserved	78
reserved	79
ww0	80
ww1	81
ww2	82
ww3	83
ww4	84
ww5	85
ww6	86
ww7	87
ww8	88

reserved	89
reserved	90
reserved	91
reserved	92
reserved	93
reserved	94
reserved	95
m0	96
m1	97
m2	98
m3	99
m4	100
m5	101
m6	102
m7	103
m8	104
maskv	105
nhood	106
reserved	107
mflag_o	108
mflag_c	109
mflag_n	110
mflag_z	111
gpr0	112
gpr1	113
gpr2	114
gpr3	115
gpr4	116
gpr5	117
gpr6	118
gpr7	119
gpr8	120
gpr9	121
gpr10	122
gpr11	123
gpr12	124
gpr13	125
gpr14	126
gpr15	127
out0	128
out1	129
out2	130
out3	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
opixa0	144
opix0	145
opixa1	146
opix1	147
reserved	148
reserved	149
reserved	150
reserved	151
acc0	152
acc1	153
acc2	154
acc3	155
sacc0	156

sacc1	157
sacc2	158
sacc3	159
sum	160
clipped	161
threshold	162
scaled	163
max	164
sortmax	164
min	165
sortmin	165
median	166
sortmedian	166
argmin	167
argmax	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
luta	208
lut	209
reserved	210
reserved	211
hbina	212
hbinadd	213
hbinincl	214
hbininch	215
sfl0	216
sfh0	217
sfl1	218
sfh1	219
sfl2	220
sfh2	221

sfl3	222
sfh3	223
srec	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
soin	240
soclear	241
so0	242
so1	243
so2	244
so3	245
so4	246
soa0	247
soa1	248
soa2	249
soa3	250
soa4	251
lock	252
prng	253
one	254
zero	255

1.3.8 *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
confmxalu	7
confaddt	8
confthres	9
confsort	10
confbest	11
confhist	12
confstat	13
reserved	14
reserved	15
ina0	16
ina1	17
ina2	18
ina3	19
ina4	20
ina5	21

ina6	22
ina7	23
ina8	24
reserved	25
reserved	26
reserved	27
reserved	28
reserved	29
reserved	30
reserved	31
inb0	32
inb1	33
inb2	34
inb3	35
inb4	36
inb5	37
inb6	38
inb7	39
inb8	40
reserved	41
reserved	42
reserved	43
reserved	44
reserved	45
reserved	46
reserved	47
inalpha0	48
inalpha1	49
inalpha2	50
inalpha3	51
inalpha4	52
inalpha5	53
inalpha6	54
inalpha7	55
inalpha8	56
reserved	57
reserved	58
reserved	59
reserved	60
reserved	61
reserved	62
reserved	63
w0	64
w1	65
w2	66
w3	67
w4	68
w5	69
w6	70
w7	71
w8	72
reserved	73
reserved	74
reserved	75
reserved	76
reserved	77
reserved	78
reserved	79
ww0	80
ww1	81
ww2	82
ww3	83
ww4	84
ww5	85
ww6	86
ww7	87
ww8	88
reserved	89

reserved	90
reserved	91
reserved	92
reserved	93
reserved	94
reserved	95
m0	96
m1	97
m2	98
m3	99
m4	100
m5	101
m6	102
m7	103
m8	104
maskv	105
nhood	106
reserved	107
mflag_o	108
mflag_c	109
mflag_n	110
mflag_z	111
gpr0	112
gpr1	113
gpr2	114
gpr3	115
gpr4	116
gpr5	117
gpr6	118
gpr7	119
gpr8	120
gpr9	121
gpr10	122
gpr11	123
gpr12	124
gpr13	125
gpr14	126
gpr15	127
out0	128
out1	129
out2	130
out3	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
opixa0	144
opix0	145
opixa1	146
opix1	147
reserved	148
reserved	149
reserved	150
reserved	151
acc0	152
acc1	153
acc2	154
acc3	155
sacc0	156
sacc1	157

sacc2	158
sacc3	159
sum	160
clipped	161
threshold	162
scaled	163
max	164
sortmax	164
min	165
sortmin	165
median	166
sortmedian	166
argmin	167
argmax	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
luta	208
lut	209
reserved	210
reserved	211
hbina	212
hbinadd	213
hbinincl	214
hbininch	215
sfl0	216
sfh0	217
sfl1	218
sfh1	219
sfl2	220
sfh2	221
sfl3	222

sfh3	223
srec	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
soin	240
soclear	241
so0	242
so1	243
so2	244
so3	245
so4	246
soa0	247
soa1	248
soa2	249
soa3	250
soa4	251
lock	252
prng	253
one	254
zero	255

1.3.9 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

1.3.10 LCNT

Size	1
Bits	[3,3]

Description:

Loop counter selection

Enumeration	Value
loopcnt1	0
loopcnt	1

1.3.11 MASKIN1

Size	2
Bits	[17,16]

Description:

Matrix Operator 1 register

- 0: W
- 1: INALPHA
- 2: WW
- 3: Reserved

Enumeration	Value
w	0
inalpha	1
ww	2
reserved	3

1.3.12 MIN1

Size	3
Bits	[18,16]

Description:

Matrix Operator 1 register

Enumeration	Value
w	0
ina	1
ww	2
inb	3
mzero	4

1.3.13 MIN2

Size	2
Bits	[1,0]

Description:

Matrix Operator 2 register

Enumeration	Value
w	0
ina	1
ww	2
inb	3

1.3.14 OPCD

Size	4
Bits	[27,24]

Description:

Extended opcode

1.3.15 *RELADDR*

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

Description:

Relative branch target address

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

1.3.16 *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
MASKI	11
MASK	12
M=MxI	13
M=MxS	14
M=MxM	15

1.4 Instructions

A B C D H L M N O P S T 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

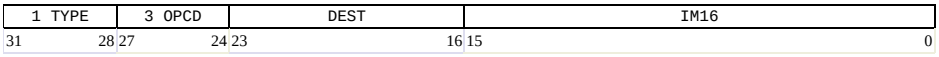
Affect instruction:

Register	Fields	Details
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:



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

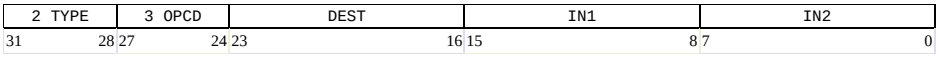
Affect instruction:

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:



Attributes:
isp1, isp2

1.4.3 absdiff w,MIN1,IM16

Description:

Absolute difference between a matrix register and a constant value and store the result in the working matrix register (W)

Matrix condition code flags modified: negative, zero

W[i] = absdiff(MIN1[i] - IM16)

Action:

```
{
  int i ;
 malu ( MALU_OP_ABSDIFF , MIN1 , 0 , 5 , IM16 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MRES ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_N, cc_get_Z, flag_ways, malu

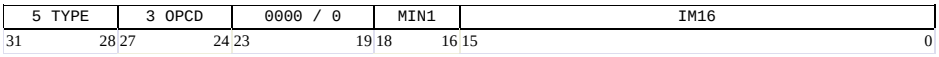
Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_N	Entire Register	
MFLAG_Z	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.4 *absdiff w,MIN1,IN2*

Description:

Absolute difference between a matrix register and a register value and store the result in the working matrix register (W)

W[i] = absdiff(MIN1[i] - R[IN2])

Matrix condition code flags modified: negative, zero

Action:

```
{
  int i ;
 malu ( MALU_OP_ABSDIFF , MIN1 , 0 , 5 , R ( IN2 ) ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MRES ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_N, cc_get_Z, flag_ways, malu

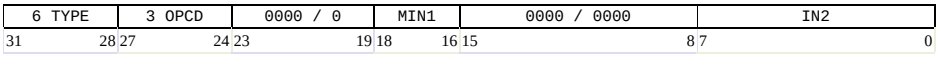
Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_N	Entire Register	
MFLAG_Z	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.5 *absdiff w,MIN1,MIN2*

Description:

Absolute difference between two matrix registers and store the result in the working matrix register (W)

W[i] = absdiff(MIN1[i] - MIN2[i])

Matrix condition code flags modified: negative, zero

Action:

```
{
  int i ;
 malu ( MALU_OP_ABSDIFF , MIN1 , 0 , MIN2 , 0 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
```

```
        if ( MASK ( i ) == 1 ) W ( i ) = MRES ( i ) ;
    }
    MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
    MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_N, cc_get_Z, flag_ways, malu

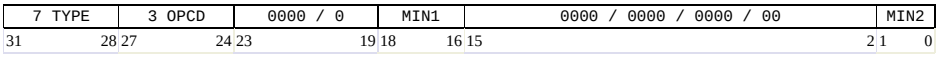
Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_N	Entire Register	
MFLAG_Z	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.6 *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:

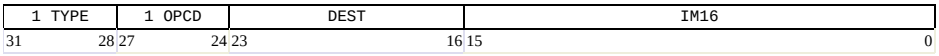
Register	Fields	Details
CONFALU	SAT, SGN	Partial

Register	Fields	Details
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:



Attributes:

isp1, isp2

1.4.7 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

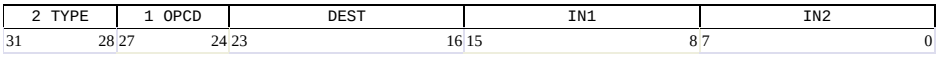
Affect instruction:

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:



Attributes:

isp1, isp2

1.4.8 *add w,MIN1,IM16*

Description:

Add a constant value to a matrix register and store the result in the working matrix register (W)

W[i] = Saturate(MIN1[i] + IM16)

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

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

Action:

```
{
  int i ;
 malu ( MALU_OP_ADD , MIN1 , 0 , 5 , IM16 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MSAT ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
 MFLAG_O = flag_ways ( MFLAG_O , I_MFLAG_O , MASKV ) ;
 MFLAG_C = flag_ways ( MFLAG_C , I_MFLAG_C , MASKV ) ;
}
```

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

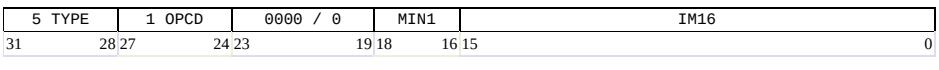
Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.9 *add w,MIN1,IN2*

Description:

Add a register value to a matrix register and store the result in the working matrix register (W)

W[i] = Saturate(MIN1[i] + R[IN2])

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

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

Action:

```
{
  int i ;
 malu ( MALU_OP_ADD , MIN1 , 0 , 5 , R ( IN2 ) ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MSAT ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
 MFLAG_O = flag_ways ( MFLAG_O , I_MFLAG_O , MASKV ) ;
 MFLAG_C = flag_ways ( MFLAG_C , I_MFLAG_C , MASKV ) ;
}
```

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

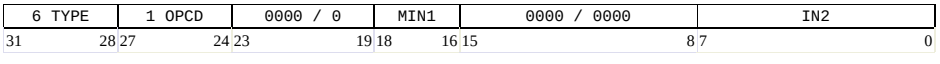
Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.10 *add w,MIN1,MIN2*

Description:

Add two matrix registers to and store the result in the working matrix (W)

W[i] = Saturate(MIN1[i] + MIN2[i])

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

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

Action:

```
{
  int i ;
 malu ( MALU_OP_ADD , MIN1 , 0 , MIN2 , 0 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MSAT ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
 MFLAG_O = flag_ways ( MFLAG_O , I_MFLAG_O , MASKV ) ;
 MFLAG_C = flag_ways ( MFLAG_C , I_MFLAG_C , MASKV ) ;
}
```

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

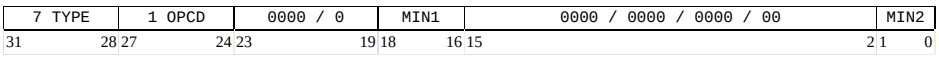
Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.11 *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

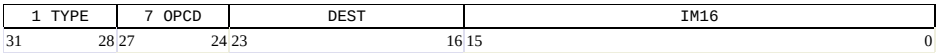
Affect instruction:

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:



Attributes:
isp1, isp2

1.4.12 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:

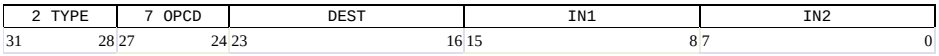
Register	Fields	Details
CONFALU	SAT, SGN	Partial
R (IN1)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

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

Register	Fields	Details
RES	Entire Register	
SAT	Entire Register	

Encoding:



Attributes:

isp1, isp2

1.4.13 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

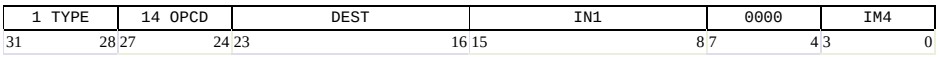
Affect instruction:

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:



Attributes:

isp1, isp2

1.4.14 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

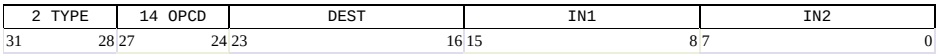
Affect instruction:

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:



Attributes:

isp1, isp2

1.4.15 *asl w,MIN1,IM4*

Description:

Shift left of a matrix registers by a constant value and store the result in the working matrix (W). Depending on CONFALU.SGN and CONFALU.SAT the result is saturated

W[i] = Saturate(MIN1[i] << IM4)

Matrix condition code flags modified: negative, zero

Action:

```
{
  int i ;
 malu ( MALU_OP_ASL , MIN1 , 0 , 5 , IM4 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MSAT ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_N, cc_get_Z, flag_ways, malu

Affect instruction:

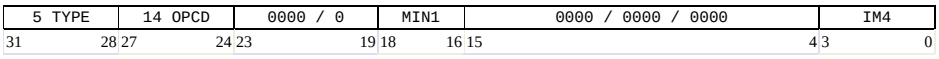
Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	

Register	Fields	Details
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_N	Entire Register	
MFLAG_Z	Entire Register	
MRES (0)	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.16 *asl w,MIN1,IN2*

Description:

Shift left of a matrix registers by a register value and store the result in the working matrix (W). Depending on CONFALU.SGN and CONFALU.SAT the result is saturated

W[i] = Saturate(MIN1[i] << (R[IN2] % 0xF))

Matrix condition code flags modified: negative, zero

Action:

```
{
  int i ;
 malu ( MALU_OP_ASL , MIN1 , 0 , 5 , R ( IN2 ) ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MSAT ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_N, cc_get_Z, flag_ways, malu

Affect instruction:

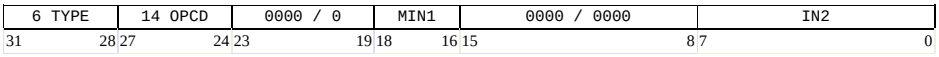
Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_N	Entire Register	
MFLAG_Z	Entire Register	
MRES (0)	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.17 *asl w,MIN1,MIN2*

Description:

Shift left of a matrix registers by a matrix register and store the result in the working matrix (W). Depending on CONFALU.SGN and CONFALU.SAT the result is saturated

W[i] = Saturate(MIN1[i] << (MIN2[i]) & 0xF)

Matrix condition code flags modified: negative, zero

Action:

```
{
  int i ;
 malu ( MALU_OP_ASL , MIN1 , 0 , MIN2 , 0 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MSAT ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_N, cc_get_Z, flag_ways, malu

Affect instruction:

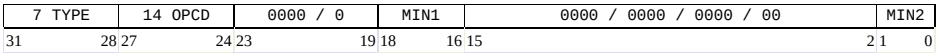
Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_N	Entire Register	
MFLAG_Z	Entire Register	

Register	Fields	Details
MRES (0)	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.18 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

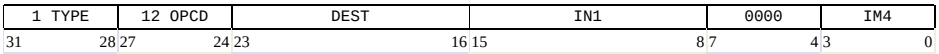
Affect instruction:

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:



Attributes:

isp1, isp2

1.4.19 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

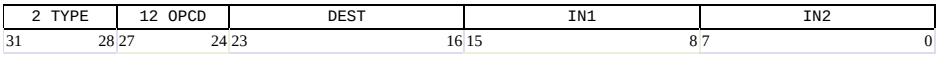
Affect instruction:

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:



Attributes:

isp1, isp2

1.4.20 asr w,MIN1,IM4

Description:

Arithmetic shift right of a matrix registers by a constant value and store the result in the working matrix (W)

W[i] = ((signed)MIN1[i]) >> IM4

Matrix condition code flags modified: negative, zero

Action:

```
{
  int i ;
 malu ( MALU_OP_ASR , MIN1 , 0 , 5 , IM4 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MRES ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_N, cc_get_Z, flag_ways, malu

Affect instruction:

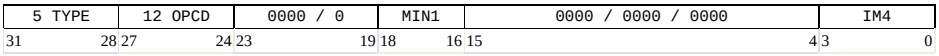
Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	

Affected by instruction:

Register	Fields	Details

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_N	Entire Register	
MFLAG_Z	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.21 *asr w,MIN1,IN2*

Description:

Arithmetic shift right of a matrix registers by a register value and store the result in the working matrix (W)

W[i] = ((signed)MIN1[i]) >> (R[IN2] % 0xF)

Matrix condition code flags modified: negative, zero

Action:

```
{
  int i ;
 malu ( MALU_OP_ASR , MIN1 , 0 , 5 , R ( IN2 ) ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MRES ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_N, cc_get_Z, flag_ways, malu

Affect instruction:

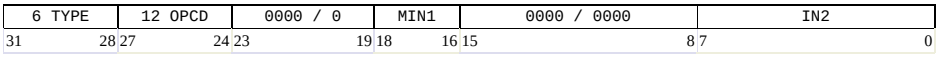
Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_N	Entire Register	
MFLAG_Z	Entire Register	

Register	Fields	Details
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.22 asr w,MIN1,MIN2

Description:

Arithmetic shift right of a matrix registers by a matrix register and store the result in the working matrix (W)

W[i] = ((signed)MIN1[i]) >> (MIN2[i] & 0xF)

Matrix condition code flags modified: negative, zero

Action:

```
{
  int i ;
 malu ( MALU_OP_ASR , MIN1 , 0 , MIN2 , 0 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MRES ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_N, cc_get_Z, flag_ways, malu

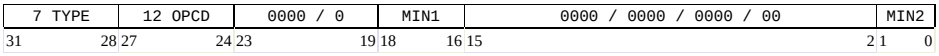
Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_N	Entire Register	
MFLAG_Z	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.23 bal RELADDR

Description:

Unconditional branch instruction.

Action:

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

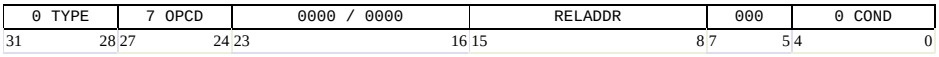
Affect instruction:

Register	Fields	Details
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	

Encoding:



Attributes:

isp1, isp2

1.4.24 bcc RELADDR

Description:

Branch if carry flag is cleared.

Action:

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

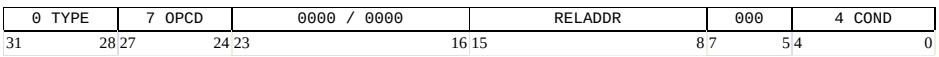
Affect instruction:

Register	Fields	Details
CC_ZNCO	C	Partial
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.25 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

1.4.26 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

1.4.27 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 ==
}
```

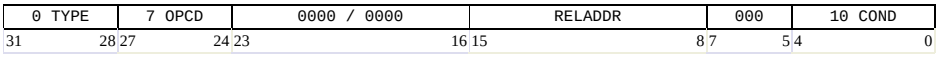
Affect instruction:

Register	Fields	Details
CC_ZNCO	N, O	Partial
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.28 *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 ==
}
```

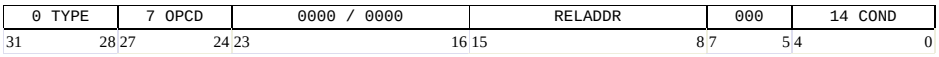
Affect instruction:

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

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.29 *bhi RELADDR*

Description:

Branch if higher (unsigned).

Action:

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

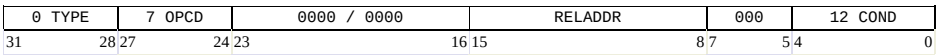
Affect instruction:

Register	Fields	Details
CC_ZNCO	C, Z	Partial
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.30 *bls RELADDR*

Description:

Branch if lower or same (unsigned).

Action:

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

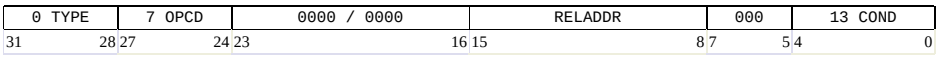
Affect instruction:

Register	Fields	Details
CC_ZNCO	C, Z	Partial
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.31 *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 ;
}
```

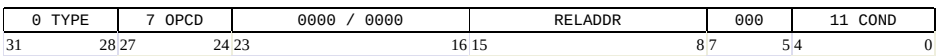
Affect instruction:

Register	Fields	Details
CC_ZNCO	N, O	Partial
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.32 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

1.4.33 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

1.4.34 bnv RELADDR

Description:

Branch never instruction. This is equivalent to a NOP.

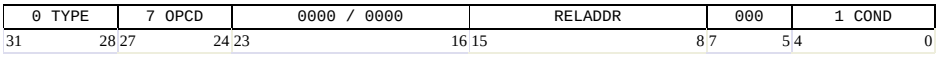
Action:

```
{
}
```

Affect instruction:

Register	Fields	Details
IADDR	Entire Register	

Encoding:



Attributes:
isp1, isp2

1.4.35 boc RELADDR

Description:

Branch if overflow flag is cleared.

Action:

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

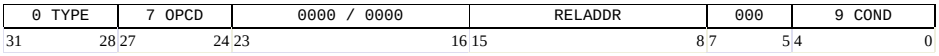
Affect instruction:

Register	Fields	Details
CC_ZNCO	O	Partial
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	Conditional

Encoding:



Attributes:
isp1, isp2

1.4.36 bos RELADDR

Description:

Branch if overflow flag is set.

Action:

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

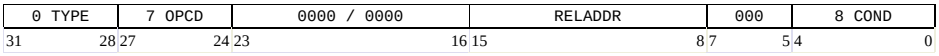
Affect instruction:

Register	Fields	Details
CC_ZNCO	O	Partial
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.37 bpl RELADDR

Description:

Branch if negative flag is cleared.

Action:

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

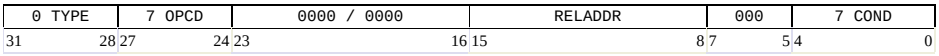
Affect instruction:

Register	Fields	Details
CC_ZNCO	N	Partial
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.38 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

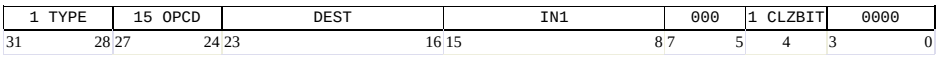
Affect instruction:

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:



Attributes:

isp1, isp2

1.4.39 *clz w,MIN1*

Description:

Shift left of a matrix registers by a constant value and store the result in the working matrix (W). Depending on CONFALU.SGN and CONFALU.SAT the result is saturated

W[i] = clz MIN1[i])

Matrix condition code flags modified: none

Action:

```
{
  int i ;
 malu ( MALU_OP_CLZ , MIN1 , 0 , 5 , 0 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MRES ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_N, cc_get_Z, flag_ways, malu

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_N	Entire Register	
MFLAG_Z	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:

5 TYPE	15 OPCD	0000 / 0	MIN1	0000 / 0000 / 000	1 CLZBIT	0000
31 28	27 24	23 19	18 16	15 5	4	3 0

Attributes:

isp1, isp2

1.4.40 *dbg*

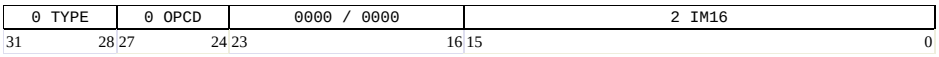
Description:

dbg: enter debug mode

Action:

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

Encoding:



Attributes:
isp1, isp2

1.4.41 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 ;
}
```

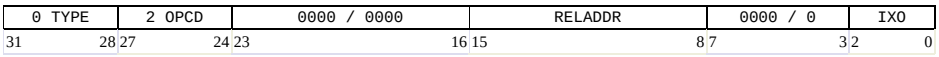
Affect instruction:

Register	Fields	Details
IADDR	Entire Register	

Affected by instruction:

Register	Fields	Details
NIA	Entire Register	

Encoding:



Attributes:
isp1, isp2

1.4.42 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 ;
}
```

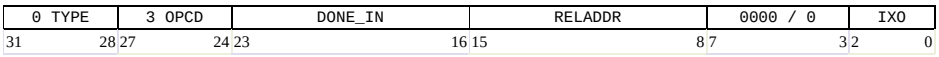
Affect instruction:

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:



Attributes:

isp1, isp2

1.4.43 *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:

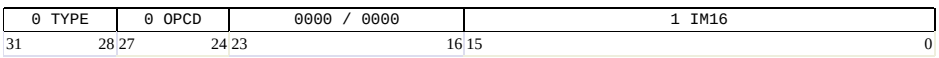
Register	Fields	Details
H_POS	Entire Register	Conditional
H_XCFG	Entire Register	Conditional
I_BUFFERED	Entire Register	
I_INACFG	NEXTCFG	Partial, Conditional
I_INALCFG	NEXTCFG	Partial, Conditional
I_INBCFG	NEXTCFG	Partial, Conditional
I_OUTCFG	NEXTCFG	Partial, Conditional
I_RUN	Entire Register	Conditional
I_START	NEXT	Partial, Conditional
S_CHCFG_INA (0, 2)	Entire Register	Conditional
S_CHCFG_INAL (0, 2)	Entire Register	Conditional
S_CHCFG_INB (0, 2)	Entire Register	Conditional
S_CHCFG_OUT (0, 3)	Entire Register	Conditional
S_LINELEN_INA (0, 2)	Entire Register	Conditional
S_LINELEN_INAL (0, 2)	Entire Register	Conditional
S_LINELEN_INB (0, 2)	Entire Register	Conditional
S_LINELEN_OUT (0, 3)	Entire Register	Conditional
S_LINE_PTR_INA (0, 2)	Entire Register	Conditional
S_LINE_PTR_INAL (0, 2)	Entire Register	Conditional
S_LINE_PTR_INB (0, 2)	Entire Register	Conditional
S_LINE_PTR_OUT (0, 3)	Entire Register	Conditional

Affected by instruction:

Register	Fields	Details
H_CURRXCFG	Entire Register	Conditional
I_BUFFERED	Entire Register	Conditional
I_EVENTS	Entire Register	

Register	Fields	Details
I_INACFG	CURRCFG	Partial, Conditional
I_INALCFG	CURRCFG	Partial, Conditional
I_INBCFG	CURRCFG	Partial, Conditional
I_OUTCFG	CURRCFG	Partial, Conditional
I_POS_INA (0, 2)	Entire Register	Conditional
I_POS_INAL (0, 2)	Entire Register	Conditional
I_POS_INB (0, 2)	Entire Register	Conditional
I_POS_OUT (0, 3)	Entire Register	Conditional
I_RPTCNT_INA (0, 2)	Entire Register	Conditional
I_RPTCNT_INAL (0, 2)	Entire Register	Conditional
I_RPTCNT_INB (0, 2)	Entire Register	Conditional
I_RUN	Entire Register	
I_SKIPCNT_OUT (0, 3)	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_INA (0, 2)	Entire Register	Conditional
S_CURRCHCFG_INAL (0, 2)	Entire Register	Conditional
S_CURRCHCFG_INB (0, 2)	Entire Register	Conditional
S_CURRCHCFG_OUT (0, 3)	Entire Register	Conditional
S_CURRLINELEN_INA (0, 2)	Entire Register	Conditional
S_CURRLINELEN_INAL (0, 2)	Entire Register	Conditional
S_CURRLINELEN_INB (0, 2)	Entire Register	Conditional
S_CURRLINELEN_OUT (0, 3)	Entire Register	Conditional
S_CURRLINE_PTR_INA (0, 2)	Entire Register	Conditional
S_CURRLINE_PTR_INAL (0, 2)	Entire Register	Conditional
S_CURRLINE_PTR_INB (0, 2)	Entire Register	Conditional
S_CURRLINE_PTR_OUT (0, 3)	Entire Register	Conditional
XPOS	Entire Register	Conditional
YPOS	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.44 Idon 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 ;
 }
}
```

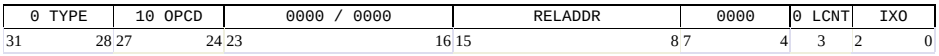
Affect instruction:

Register	Fields	Details
IADDR	Entire Register	
LOOPCNT	Entire Register	

Affected by instruction:

Register	Fields	Details
LOOPCNT	Entire Register	Conditional
NIA	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.45 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 ;
 }
}
```

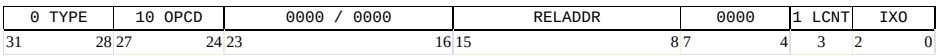
Affect instruction:

Register	Fields	Details
IADDR	Entire Register	
LOOPCNT1	Entire Register	

Affected by instruction:

Register	Fields	Details
LOOPCNT1	Entire Register	Conditional
NIA	Entire Register	Conditional

Encoding:



Attributes:

isp2

1.4.46 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
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

1.4.47 Idot1 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

1.4.48 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

1.4.49 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 ;
 }
}
```

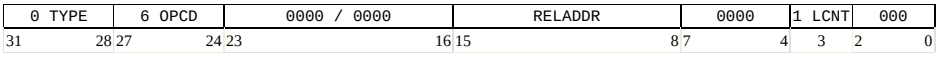
Affect instruction:

Register	Fields	Details
IADDR	Entire Register	
LOOPCNT1	Entire Register	

Affected by instruction:

Register	Fields	Details
LOOPCNT1	Entire Register	Conditional
NIA	Entire Register	Conditional

Encoding:



Attributes:

isp2

1.4.50 *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

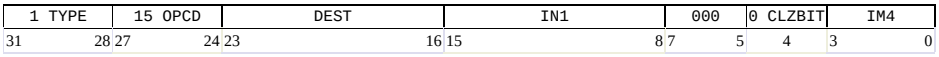
Affect instruction:

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:



Attributes:

isp1, isp2

1.4.51 *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

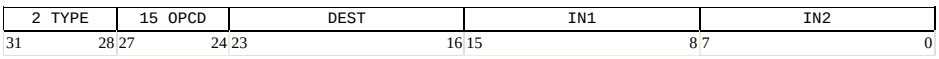
Affect instruction:

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:



Attributes:
isp1, isp2

1.4.52 *lsl w,MIN1,IM4*

Description:

Bit shift left of a matrix registers by a constant value and store the result in the working matrix (W).

W[i] = (MIN1[i]) << IM4) & 0xFFFF

Matrix condition code flags modified: negative, zero

Action:

```
{
  int i ;
 malu ( MALU_OP_ASL , MIN1 , 0 , 5 , IM4 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MRES ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_N, cc_get_Z, flag_ways, malu

Affect instruction:

Register	Fields	Details

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_N	Entire Register	
MFLAG_Z	Entire Register	
MRES (0)	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:

5 TYPE	15 OPCD	0000 / 0	MIN1	0000 / 0000 / 000	0 CLZBIT	IM4
31 28	27 24	23 19	18 16	15 5	4	3 0

Attributes:

isp1, isp2

1.4.53 *lsl w,MIN1,IN2*

Description:

Bit shift left of a matrix registers by a register value and store the result in the working matrix (W). Do not do any saturation

W[i] = (MIN1[i]) << (R[IN2] & 0xF)) & 0xFFFF

Matrix condition code flags modified: negative, zero

Action:

```
{
  int i ;
 malu ( MALU_OP_ASL , MIN1 , 0 , 5 , R ( IN2 ) ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MRES ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_N, cc_get_Z, flag_ways, malu

Affect instruction:

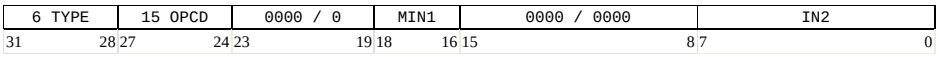
Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	

Register	Fields	Details
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_N	Entire Register	
MFLAG_Z	Entire Register	
MRES (0)	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.54 *lsl w,MIN1,MIN2*

Description:

Bit shift left of a matrix registers by a matrix register and store the result in the working matrix (W). Do not do any saturation

W[i] = (MIN1[i]) << (MIN2[i] & 0xF)) & 0xFFFF

Matrix condition code flags modified: negative, zero

Action:

```
{
  int i ;
 malu ( MALU_OP_ASL , MIN1 , 0 , MIN2 , 0 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MRES ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_N, cc_get_Z, flag_ways, malu

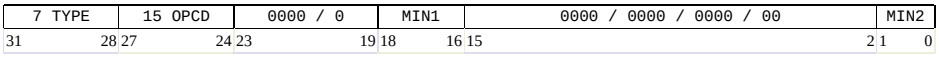
Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_N	Entire Register	
MFLAG_Z	Entire Register	
MRES (0)	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.55 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

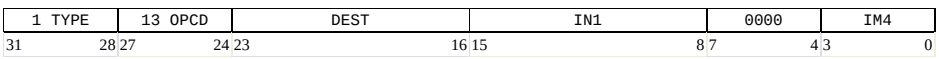
Affect instruction:

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:



Attributes:

isp1, isp2

1.4.56 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

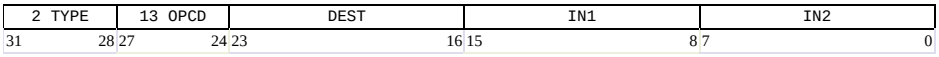
Affect instruction:

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:



Attributes:
isp1, isp2

1.4.57 lsr w,MIN1,IM4

Description:

Logic shift right of a matrix registers by a constant value and store the result in the working matrix (W)

W[i] = ((unsigned)MIN1[i]) >> IM4

Matrix condition code flags modified: negative, zero

Action:

```
{
  int i ;
 malu ( MALU_OP_LSR , MIN1 , 0 , 5 , IM4 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MRES ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_N, cc_get_Z, flag_ways, malu

Affect instruction:

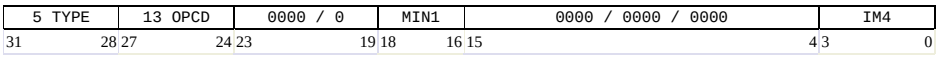
Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	

Register	Fields	Details
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_N	Entire Register	
MFLAG_Z	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.58 lsr w,MIN1,IN2

Description:

Logic shift right of a matrix registers by a register value and store the result in the working matrix (W)

W[i] = ((unsigned)MIN1[i]) >> (R[IN2] & 0xF)

Matrix condition code flags modified: negative, zero

Action:

```
{
  int i ;
 malu ( MALU_OP_LSR , MIN1 , 0 , 5 , R ( IN2 ) ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MRES ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_N, cc_get_Z, flag_ways, malu

Affect instruction:

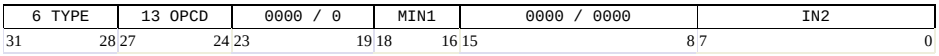
Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_N	Entire Register	
MFLAG_Z	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.59 *lsr w,MIN1,MIN2*

Description:

Logic shift right of a matrix registers by a matrix register and store the result in the working matrix (W)

W[i] = ((unsigned)MIN1[i]) >> (MIN2[i] & 0xF)

Matrix condition code flags modified: negative, zero

Action:

```
{
  int i ;
 malu ( MALU_OP_LSR , MIN1 , 0 , MIN2 , 0 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MRES ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_N, cc_get_Z, flag_ways, malu

Affect instruction:

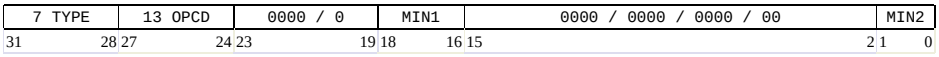
Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_N	Entire Register	
MFLAG_Z	Entire Register	
MRES (0, 8)	Entire Register	

Register	Fields	Details
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.60 *mand IM16*

Description:

Select MASK if Matrix value is less than constant

MASK[i] = (MASKIN[i] == IM16)? NHOOD[i] : 0;

Action:

```
{
  int i ;
 maskalu ( MASKALU_OP_AND , 1 , IM16 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     MASK ( i ) = MASKRES ( i ) ;
 }
}
```

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

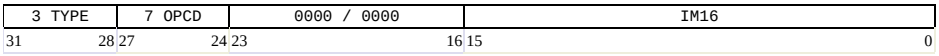
Affect instruction:

Register	Fields	Details
MASKOP1 (0, 8)	Entire Register	
MASKOP2	Entire Register	Partial

Affected by instruction:

Register	Fields	Details
MASK (0, 8)	Entire Register	
MASKOP1 (1)	Entire Register	
MASKOP2	Entire Register	
MASKRES (0, 8)	Entire Register	

Encoding:



Attributes:

isp1, isp2

1.4.61 *mand IN2*

Description:

Select MASK if Matrix value is less than register value

MASK[i] = (MASKIN[i] == IM16)? NHOOD[i] : 0;

Action:

```
{
  int i ;
 maskalu ( MASKALU_OP_AND , 1 , R ( IN2 ) ) ;
```

```
 for ( i = 0 ; i < 9 ; i ++ ) {
     MASK ( i ) = MASKRES ( i ) ;
 }
}
```

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

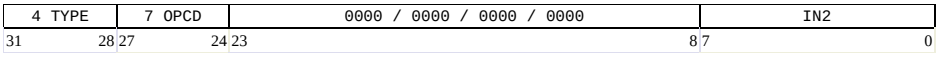
Affect instruction:

Register	Fields	Details
MASKOP1 (0, 8)	Entire Register	
MASKOP2	Entire Register	Partial
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
MASK (0, 8)	Entire Register	
MASKOP1 (1)	Entire Register	
MASKOP2	Entire Register	
MASKRES (0, 8)	Entire Register	

Encoding:



Attributes:

isp1, isp2

1.4.62 *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

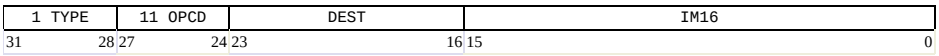
Affect instruction:

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:



Attributes:

isp1, isp2

1.4.63 *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

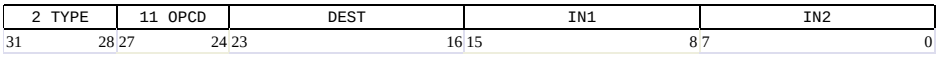
Affect instruction:

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:



Attributes:
isp1, isp2

1.4.64 *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

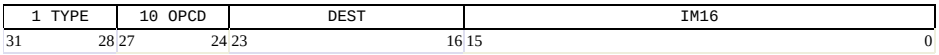
Affect instruction:

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:



Attributes:
isp1, isp2

1.4.65 *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

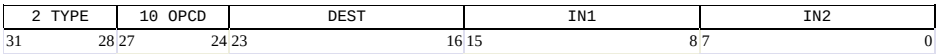
Affect instruction:

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:



Attributes:
isp1, isp2

1.4.66 *minv*

Description:

Invert MASK bits and mask out using NHOOD

MASK[i] = (!MASK[i]) && NHOOD[i]

Action:

```
{
  int i ;
 maskalu ( MASKALU_OP_INV , 1 , 0 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     MASK ( i ) = MASKRES ( i ) ;
 }
}
```

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

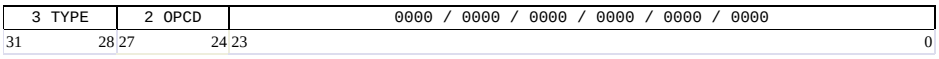
Affect instruction:

Register	Fields	Details
MASKOP1 (0, 8)	Entire Register	
NHOOD	Entire Register	Partial

Affected by instruction:

Register	Fields	Details
MASK (0, 8)	Entire Register	
MASKOP1 (1)	Entire Register	
MASKOP2	Entire Register	
MASKRES (0, 8)	Entire Register	

Encoding:



Attributes:

isp1, isp2

1.4.67 mith MASKIN1,IM16

Description:

Select MASK if Matrix value is less or equal than constant

MASK[i] = (MASKIN[i] <= IM16)? NHOOD[i] : 0;

Action:

```
{
  int i ;
 maskalu ( MASKALU_OP_ITH , MASKIN1 , IM16 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     MASK ( i ) = MASKRES ( i ) ;
 }
}
```

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

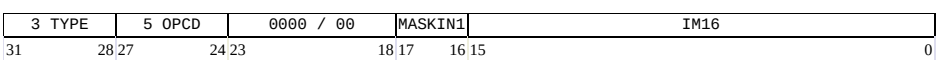
Affect instruction:

Register	Fields	Details
MASKOP1 (0, 8)	Entire Register	
MASKOP2	Entire Register	
NHOOD	Entire Register	Partial

Affected by instruction:

Register	Fields	Details
MASK (0, 8)	Entire Register	
MASKOP1 (MASKIN1)	Entire Register	
MASKOP2	Entire Register	
MASKRES (0, 8)	Entire Register	

Encoding:



Attributes:

isp1, isp2

1.4.68 *mith MASKIN1,IN2*

Description:

Select MASK if Matrix value is less or equal than register value

MASK[i] = (MASKIN[i] == IN2)? NHOOD[i] : 0;

Action:

```
{
  int i ;
 maskalu ( MASKALU_OP_ITH , MASKIN1 , R ( IN2 ) ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     MASK ( i ) = MASKRES ( i ) ;
 }
}
```

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

Affect instruction:

Register	Fields	Details
MASKOP1 (0, 8)	Entire Register	
MASKOP2	Entire Register	
NHOOD	Entire Register	Partial
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
MASK (0, 8)	Entire Register	
MASKOP1 (MASKIN1)	Entire Register	
MASKOP2	Entire Register	
MASKRES (0, 8)	Entire Register	

Encoding:

4 TYPE	5 OPCD	0000 / 00	MASKIN1	0000 / 0000	IN2
31 28	27 24	23 18	17 16	15 8	7 0

Attributes:

isp1, isp2

1.4.69 *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

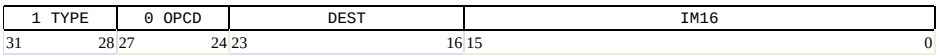
Affect instruction:

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:



Attributes:

isp1, isp2

1.4.70 mov DEST,IN2

Description:

Copy register to register

R[DEST] = R[IN2]

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

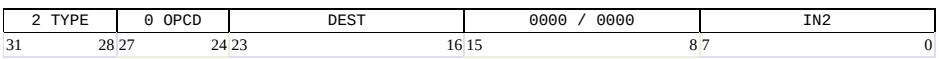
Affect instruction:

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:



Attributes:

isp1, isp2

1.4.71 mov w,IM16

Description:

Copy a constant value to all register of the working register matrix (W)

W[i] = IM16

Matrix condition code flags modified: none

Action:

```
{
  int i ;
 malu ( MALU_OP_MOV , 5 , IM16 , 5 , IM16 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MRES ( i ) ;
 }
}
```

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

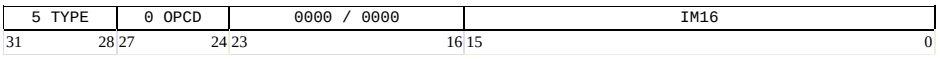
Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MFLAG_C	Entire Register	
MFLAG_O	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	
MRES (0, 8)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.72 mov w,IN2

Description:

Copy a register value to all register of the working register matrix (W)

W[i] = R[IN2]

Matrix condition code flags modified: none

Action:

```
{
  int i ;
 malu ( MALU_OP_MOV , 5 , IN2 , 5 , R ( IN2 ) ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MRES ( i ) ;
 }
}
```

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

Affect instruction:

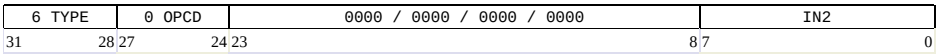
Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	

Register	Fields	Details
MFLAG_C	Entire Register	
MFLAG_O	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	
MRES (0, 8)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.73 *mov w,MIN2*

Description:

Copy a matrix register to the woring register matrix (W)

W[i] = MIN2[i]

Matrix condition code flags modified: none

Action:

```
{
  int i ;
 malu ( MALU_OP_MOV , MIN2 , 0 , MIN2 , 0 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MRES ( i ) ;
 }
}
```

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

Affect instruction:

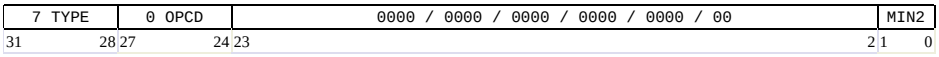
Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MFLAG_C	Entire Register	
MFLAG_O	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	
MRES (0, 8)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	

Register	Fields	Details
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.74 *mset IM16*

Description:

use bit vector from a constant and copy it to the MASK bits

MASK[i] = IM9[i]

Action:

```
{
  int i ;
 maskalu ( MASKALU_OP_SET , 1 , IM16 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     MASK ( i ) = MASKRES ( i ) ;
 }
}
```

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

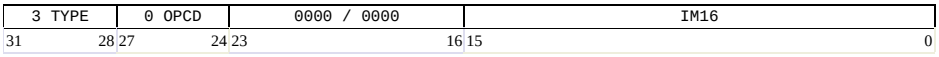
Affect instruction:

Register	Fields	Details
MASKRES (0, 8)	Entire Register	

Affected by instruction:

Register	Fields	Details
MASK (0, 8)	Entire Register	
MASKOP1 (1)	Entire Register	
MASKOP2	Entire Register	

Encoding:



Attributes:

isp1, isp2

1.4.75 *mset IN2*

Description:

use bit vector from a register and copy it to the MASK bits

MASK[i] = R[IN2][i]

Action:

```
{
  int i ;
 maskalu ( MASKALU_OP_SET , 1 , R ( IN2 ) ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     MASK ( i ) = MASKRES ( i ) ;
 }
```

```
}
```

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

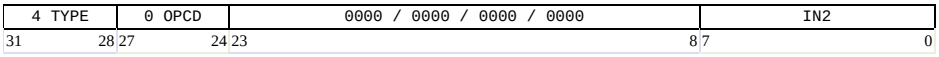
Affect instruction:

Register	Fields	Details
MASKRES (0, 8)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
MASK (0, 8)	Entire Register	
MASKOP1 (1)	Entire Register	
MASKOP2	Entire Register	

Encoding:



Attributes:

isp1, isp2

1.4.76 mset MASKIN1,IM16

Description:

Select MASK if Matrix value is equal to a constant

MASK[i] = (MASKIN[i] == IM16)? NHOOD[i] : 0;

Action:

```
{
  int i ;
 maskalu ( MASKALU_OP_SEL , MASKIN1 , IM16 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     MASK ( i ) = MASKRES ( i ) ;
 }
}
```

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

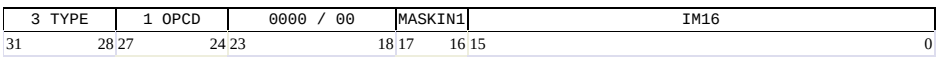
Affect instruction:

Register	Fields	Details
MASKOP1 (0, 8)	Entire Register	
MASKOP2	Entire Register	
NHOOD	Entire Register	Partial

Affected by instruction:

Register	Fields	Details
MASK (0, 8)	Entire Register	
MASKOP1 (MASKIN1)	Entire Register	
MASKOP2	Entire Register	
MASKRES (0, 8)	Entire Register	

Encoding:



Attributes:

isp1, isp2

1.4.77 mset MASKIN1,IN2

Description:

Select MASK if Matrix value is equal to a register value

MASK[i] = (MASKIN[i] == IN2)? NHOOD[i] : 0;

Action:

```
{
  int i ;
 maskalu ( MASKALU_OP_SEL , MASKIN1 , R ( IN2 ) ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     MASK ( i ) = MASKRES ( i ) ;
 }
}
```

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

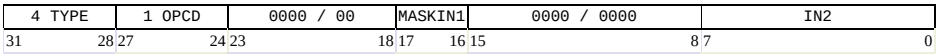
Affect instruction:

Register	Fields	Details
MASKOP1 (0, 8)	Entire Register	
MASKOP2	Entire Register	
NHOOD	Entire Register	Partial
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
MASK (0, 8)	Entire Register	
MASKOP1 (MASKIN1)	Entire Register	
MASKOP2	Entire Register	
MASKRES (0, 8)	Entire Register	

Encoding:



Attributes:
isp1, isp2

1.4.78 mthr MASKIN1,IM16

Description:

Select MASK if Matrix value is bigger or equal than constant

MASK[i] = (MASKIN[i] >= IM16)? NHOOD[i] : 0;

Action:

```
{
  int i ;
 maskalu ( MASKALU_OP_THR , MASKIN1 , IM16 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     MASK ( i ) = MASKRES ( i ) ;
 }
}
```

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

Affect instruction:

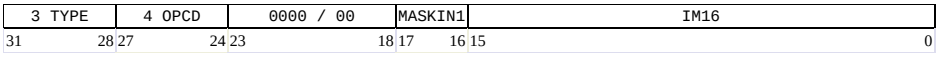
Register	Fields	Details
MASKOP1 (0, 8)	Entire Register	
MASKOP2	Entire Register	
NHOOD	Entire Register	Partial

Affected by instruction:

Register	Fields	Details
MASK (0, 8)	Entire Register	

Register	Fields	Details
MASKOP1 (MASKIN1)	Entire Register	
MASKOP2	Entire Register	
MASKRES (0, 8)	Entire Register	

Encoding:



Attributes:

isp1, isp2

1.4.79 *mthr MASKIN1,IN2*

Description:

Select MASK if Matrix value is bigger or equal than register value

MASK[i] = (MASKIN[i] >= IN2)? NHOOD[i] : 0;

Action:

```
{
  int i ;
 maskalu ( MASKALU_OP_THR , MASKIN1 , R ( IN2 ) ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     MASK ( i ) = MASKRES ( i ) ;
 }
}
```

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

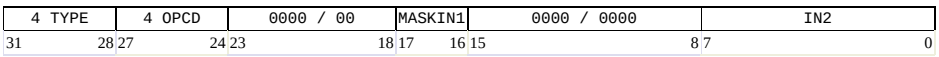
Affect instruction:

Register	Fields	Details
MASKOP1 (0, 8)	Entire Register	
MASKOP2	Entire Register	
NHOOD	Entire Register	Partial
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
MASK (0, 8)	Entire Register	
MASKOP1 (MASKIN1)	Entire Register	
MASKOP2	Entire Register	
MASKRES (0, 8)	Entire Register	

Encoding:



Attributes:

isp1, isp2

1.4.80 *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

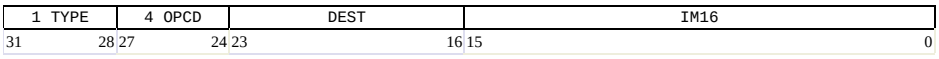
Affect instruction:

Register	Fields	Details
CONFALU	SHR, SAT, SGN	Partial
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:



Attributes:

isp1, isp2

1.4.81 *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

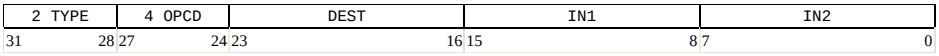
Affect instruction:

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:



Attributes:

isp1, isp2

1.4.82 *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

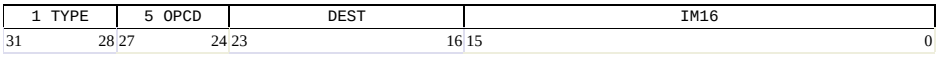
Affect instruction:

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:



Attributes:

isp1, isp2

1.4.83 *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

1.4.84 *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

1.4.85 *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

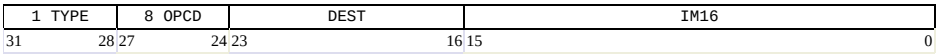
Affect instruction:

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

Affected by instruction:

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

Encoding:



Attributes:
isp1, isp2

1.4.86 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

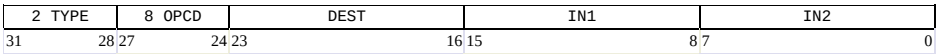
Affect instruction:

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:



Attributes:
isp1, isp2

1.4.87 param DEST,IN2

Description:

Virtual instruction to be transformed during IMEM upload to "mov R[DEST],IM16" with IM16 = PARAM[IN] from parameter memory in host

R[DEST] = PARAM[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:

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

Attributes:
isp1, isp2

1.4.88 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

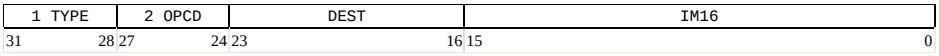
Affect instruction:

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:



Attributes:

isp1, isp2

1.4.89 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

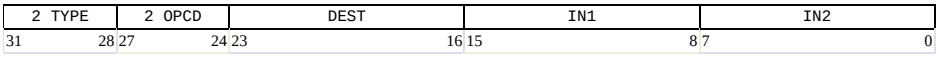
Affect instruction:

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:



Attributes:

isp1, isp2

1.4.90 sub w,MIN1,IM16

Description:

Subtract a constant value from a matrix register and store the result in the working matrix register (W)

W[i] = Saturate(MIN1[i] - IM16)

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

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

Action:

```
{
```

```
 int i ;
malu ( MALU_OP_SUB , MIN1 , 0 , 5 , IM16 ) ;
for ( i = 0 ; i < 9 ; i ++ ) {
    if ( MASK ( i ) == 1 ) W ( i ) = MSAT ( i ) ;
}
MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
MFLAG_O = flag_ways ( MFLAG_O , I_MFLAG_O , MASKV ) ;
MFLAG_C = flag_ways ( MFLAG_C , I_MFLAG_C , MASKV ) ;
}
```

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

Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:

5 TYPE	2 OPCD	0000 / 0	MIN1	IM16
31 28	27 24	23 19	18 16	15 0

Attributes:

isp1, isp2

1.4.91 *sub w,MIN1,IN2*

Description:

Subtract a register value from a matrix register and store the result in the working matrix register (W)

W[i] = Saturate(MIN1[i] - R[IN2])

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

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

Action:

```
{
  int i ;
 malu ( MALU_OP_SUB , MIN1 , 0 , 5 , R ( IN2 ) ) ;
```

```
  for ( i = 0 ; i < 9 ; i ++ ) {
      if ( MASK ( i ) == 1 ) W ( i ) = MSAT ( i ) ;
  }
  MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
  MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
  MFLAG_O = flag_ways ( MFLAG_O , I_MFLAG_O , MASKV ) ;
  MFLAG_C = flag_ways ( MFLAG_C , I_MFLAG_C , MASKV ) ;
 }
```

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

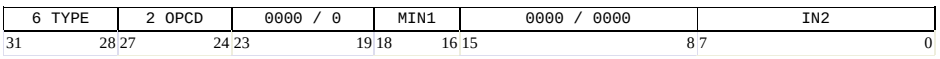
Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.92 sub w,MIN1,MIN2

Description:

Subtract two matrix registers and store the result in the working matrix (W)

W[i] = Saturate(MIN1[i] - MIN2[i])

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

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

Action:

```
{
  int i ;
 malu ( MALU_OP_SUB , MIN1 , 0 , MIN2 , 0 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
```

```
        if ( MASK ( i ) == 1 ) W ( i ) = MSAT ( i ) ;
    }
    MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
    MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
    MFLAG_O = flag_ways ( MFLAG_O , I_MFLAG_O , MASKV ) ;
    MFLAG_C = flag_ways ( MFLAG_C , I_MFLAG_C , MASKV ) ;
  }
```

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

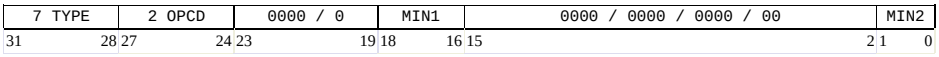
Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.93 swp

Description:

Swap W and WW

tmp[i] = WW[i]

WW[i] = W[i]

W[i] = tmp[i]

Matrix condition code flags modified: negative, zero

Action:

```
{
  int i ;
 malu ( MALU_OP_SWP , 2 , 0 , 2 , 0 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
```

```
    WW ( i ) = W ( i ) ;
 }
 for ( i = 0 ; i < 9 ; i ++ ) {
    W ( i ) = MRES ( i ) ;
 }
}
```

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

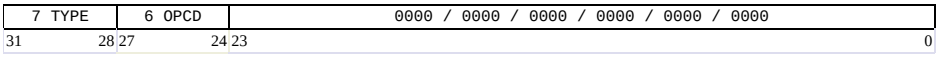
Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MFLAG_C	Entire Register	
MFLAG_O	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	
MRES (0, 8)	Entire Register	
W (0, 8)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0, 8)	Entire Register	
WW (0, 8)	Entire Register	

Encoding:



Attributes:

isp1, isp2

1.4.94 *thf w,MIN1,IM16*

Description:

Threshold Filter: threshold a matrix register with a constant value and keep the remaining values in the working matrix register (W)

W[i] = (MIN1[i] < IM16)? 0: MIN1[i]

Matrix condition code flags modified: negative, zero

Action:

```
{
  int i ;
 malu ( MALU_OP_THF , MIN1 , 0 , 5 , IM16 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
    if ( MASK ( i ) == 1 ) W ( i ) = MRES ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_N, cc_get_Z, flag_ways, malu

Affect instruction:

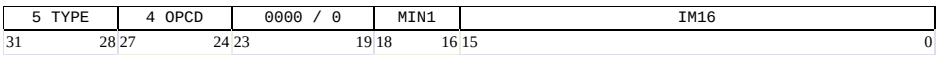
Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	

Register	Fields	Details
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_N	Entire Register	
MFLAG_Z	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.95 *thf w,MIN1,IN2*

Description:

Threshold Filter: threshold a matrix register with a register value and keep the remaining values in the working matrix register (W)

W[i] = (MIN1[i] < R[IN2])? 0: MIN1[i]

Matrix condition code flags modified: negative, zero

Action:

```
{
  int i ;
 malu ( MALU_OP_THF , MIN1 , 0 , 5 , R ( IN2 ) ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MRES ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_N, cc_get_Z, flag_ways, malu

Affect instruction:

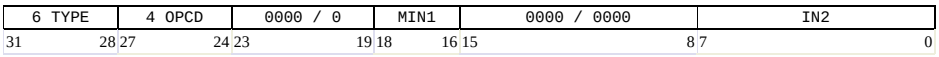
Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	

Register	Fields	Details
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_N	Entire Register	
MFLAG_Z	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.96 *thf w,MIN1,MIN2*

Description:

Threshold Filter: threshold a matrix register with a matrix register and keep the remaining values in the working matrix register (W)

W[i] = (MIN1[i] < MIN2[i])? 0: MIN1[i]

Matrix condition code flags modified: negative, zero

Action:

```
{
  int i ;
 malu ( MALU_OP_THF , MIN1 , 0 , MIN2 , 0 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MRES ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_N, cc_get_Z, flag_ways, malu

Affect instruction:

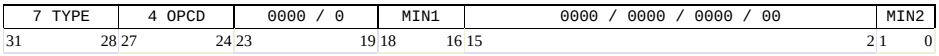
Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	

Affected by instruction:

Register	Fields	Details

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_N	Entire Register	
MFLAG_Z	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.97 *thm w,MIN1,IM16*

Description:

Threshold to Max: threshold a matrix register with a constant value and store MAX for the remaining values in the working matrix register (W)

W[i] = (MIN1[i] < IM16)? 0: 0xFFFF

Matrix condition code flags modified: negative, zero

Action:

```
{
  int i ;
 malu ( MALU_OP_THM , MIN1 , 0 , 5 , IM16 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MRES ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_N, cc_get_Z, flag_ways, malu

Affect instruction:

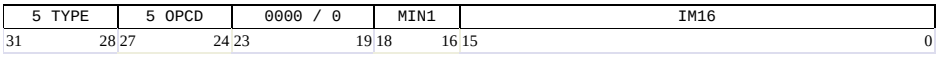
Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_N	Entire Register	
MFLAG_Z	Entire Register	

Register	Fields	Details
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.98 *thm w,MIN1,IN2*

Description:

Threshold to Max: threshold a matrix register with a register value and store MAX for the remaining values in the working matrix register (W)

W[i] = (MIN1[i] < R[IN2])? 0: 0xFFFF

Matrix condition code flags modified: negative, zero

Action:

```
{
  int i ;
 malu ( MALU_OP_THM , MIN1 , 0 , 5 , R ( IN2 ) ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MRES ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_N, cc_get_Z, flag_ways, malu

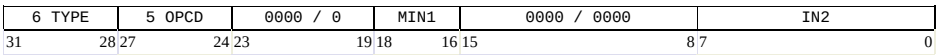
Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	
R (IN2)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_N	Entire Register	
MFLAG_Z	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.99 *thm w,MIN1,MIN2*

Description:

Threshold to Max: threshold a matrix register with a matrix register and store MAX for the remaining values in the working matrix register (W)

W[i] = (MIN1[i] < MIN2[i])? 0: 0xFFFF

Matrix condition code flags modified: negative, zero

Action:

```
{
  int i ;
 malu ( MALU_OP_THM , MIN1 , 0 , MIN2 , 0 ) ;
 for ( i = 0 ; i < 9 ; i ++ ) {
     if ( MASK ( i ) == 1 ) W ( i ) = MRES ( i ) ;
 }
 MFLAG_N = flag_ways ( MFLAG_N , I_MFLAG_N , MASKV ) ;
 MFLAG_Z = flag_ways ( MFLAG_Z , I_MFLAG_Z , MASKV ) ;
}
```

The code above uses the following routines (directly or indirectly): cc_get_N, cc_get_Z, flag_ways, malu

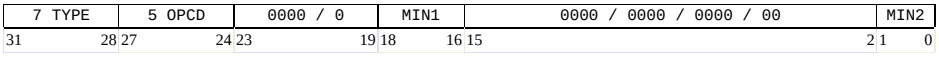
Affect instruction:

Register	Fields	Details
CONFALU	SAT, SGN	Partial
MASK (0, 8)	Entire Register	
MASKV	Entire Register	
MFLAG_C	Entire Register	
MFLAG_N	Entire Register	
MFLAG_O	Entire Register	
MFLAG_Z	Entire Register	
MOP1 (0, 8)	Entire Register	
MOP2 (0, 8)	Entire Register	
MRES (0)	Entire Register	

Affected by instruction:

Register	Fields	Details
I_MFLAG_C	Entire Register	
I_MFLAG_N	Entire Register	
I_MFLAG_O	Entire Register	
I_MFLAG_Z	Entire Register	
MFLAG_N	Entire Register	
MFLAG_Z	Entire Register	
MRES (0, 8)	Entire Register	
MSAT (0)	Entire Register	
MSAT (0, 8)	Entire Register	
W (0)	Entire Register	Conditional
W (0, 8)	Entire Register	Conditional

Encoding:



Attributes:

isp1, isp2

1.4.100 *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

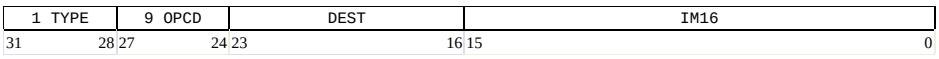
Affect instruction:

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:



Attributes:
isp1, isp2

1.4.101 *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

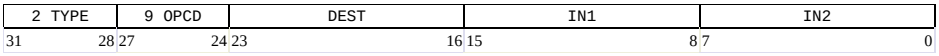
Affect instruction:

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:



Attributes:
isp1, isp2

1.5 Instructions by Attribute

1.5.1 *isp1*

- abd DEST,IM16
- abd DEST,IN1,IN2
- absdiff w,MIN1,IM16
- absdiff w,MIN1,IN2
- absdiff w,MIN1,MIN2
- add DEST,IM16
- add DEST,IN1,IN2
- add w,MIN1,IM16
- add w,MIN1,IN2
- add w,MIN1,MIN2
- and DEST,IM16
- and DEST,IN1,IN2
- asl DEST,IN1,IM4
- asl DEST,IN1,IN2
- asl w,MIN1,IM4
- asl w,MIN1,IN2
- asl w,MIN1,MIN2
- asr DEST,IN1,IM4
- asr DEST,IN1,IN2
- asr w,MIN1,IM4
- asr w,MIN1,IN2
- asr w,MIN1,MIN2
- 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 w,MIN1
- dbg
- done RELADDR,IXO
- dout DONE_IN,RELADDR,IXO
- halt
- ldon RELADDR,IXO
- ldot DONE_IN,RELADDR,IXO
- loop RELADDR
- lsl DEST,IN1,IM4

- lsl DEST,IN1,IN2
- lsl w,MIN1,IM4
- lsl w,MIN1,IN2
- lsl w,MIN1,MIN2
- lsr DEST,IN1,IM4
- lsr DEST,IN1,IN2
- lsr w,MIN1,IM4
- lsr w,MIN1,IN2
- lsr w,MIN1,MIN2
- mand IM16
- mand IN2
- max DEST,IM16
- max DEST,IN1,IN2
- min DEST,IM16
- min DEST,IN1,IN2
- minv
- mith MASKIN1,IM16
- mith MASKIN1,IN2
- mov DEST,IM16
- mov DEST,IN2
- mov w,IM16
- mov w,IN2
- mov w,MIN2
- mset IM16
- mset IN2
- mset MASKIN1,IM16
- mset MASKIN1,IN2
- mthr MASKIN1,IM16
- mthr MASKIN1,IN2
- mulh DEST,IM16
- mulh DEST,IN1,IN2
- mull DEST,IM16
- mull DEST,IN1,IN2
- nop
- or DEST,IM16
- or DEST,IN1,IN2
- param DEST,IN2
- sub DEST,IM16
- sub DEST,IN1,IN2
- sub w,MIN1,IM16
- sub w,MIN1,IN2
- sub w,MIN1,MIN2
- swp
- thf w,MIN1,IM16
- thf w,MIN1,IN2
- thf w,MIN1,MIN2
- thm w,MIN1,IM16
- thm w,MIN1,IN2
- thm w,MIN1,MIN2
- xor DEST,IM16
- xor DEST,IN1,IN2

1.5.2 *isp2*

- abd DEST,IM16
- abd DEST,IN1,IN2
- absdiff w,MIN1,IM16
- absdiff w,MIN1,IN2

- absdiff w,MIN1,MIN2
- add DEST,IM16
- add DEST,IN1,IN2
- add w,MIN1,IM16
- add w,MIN1,IN2
- add w,MIN1,MIN2
- and DEST,IM16
- and DEST,IN1,IN2
- asl DEST,IN1,IM4
- asl DEST,IN1,IN2
- asl w,MIN1,IM4
- asl w,MIN1,IN2
- asl w,MIN1,MIN2
- asr DEST,IN1,IM4
- asr DEST,IN1,IN2
- asr w,MIN1,IM4
- asr w,MIN1,IN2
- asr w,MIN1,MIN2
- 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 w,MIN1
- dbg
- done RELADDR,IXO
- dout DONE_IN,RELADDR,IXO
- halt
- ldon RELADDR,IXO
- ldon1 RELADDR,IXO
- ldot DONE_IN,RELADDR,IXO
- ldot1 DONE_IN,RELADDR,IXO
- loop RELADDR
- loop1 RELADDR
- lsl DEST,IN1,IM4
- lsl DEST,IN1,IN2
- lsl w,MIN1,IM4
- lsl w,MIN1,IN2
- lsl w,MIN1,MIN2
- lsr DEST,IN1,IM4
- lsr DEST,IN1,IN2
- lsr w,MIN1,IM4
- lsr w,MIN1,IN2
- lsr w,MIN1,MIN2
- mand IM16
- mand IN2
- max DEST,IM16
- max DEST,IN1,IN2
- min DEST,IM16

- min DEST,IN1,IN2
- minv
- mith MASKIN1,IM16
- mith MASKIN1,IN2
- mov DEST,IM16
- mov DEST,IN2
- mov w,IM16
- mov w,IN2
- mov w,MIN2
- mset IM16
- mset IN2
- mset MASKIN1,IM16
- mset MASKIN1,IN2
- mthr MASKIN1,IM16
- mthr MASKIN1,IN2
- mulh DEST,IM16
- mulh DEST,IN1,IN2
- mull DEST,IM16
- mull DEST,IN1,IN2
- nop
- or DEST,IM16
- or DEST,IN1,IN2
- param DEST,IN2
- sub DEST,IM16
- sub DEST,IN1,IN2
- sub w,MIN1,IM16
- sub w,MIN1,IN2
- sub w,MIN1,MIN2
- swp
- thf w,MIN1,IM16
- thf w,MIN1,IN2
- thf w,MIN1,MIN2
- thm w,MIN1,IM16
- thm w,MIN1,IN2
- thm w,MIN1,MIN2
- xor DEST,IM16
- xor DEST,IN1,IN2

1.6 Helper Functions

Built-in helper routines are documented in the ADL user manual.

In addition, the following helper routines are defined for IPUS:

C E F M S

1.6.1 *cc_get_C*

```
int cc_get_C ( malu_op_t alu_op , sbits < 16 > res , sbits < 16 > op1 , sbits < 16 > op2 ) {
  }
```

1.6.2 *cc_get_N*

```
int cc_get_N ( bits < 16 > res ) {
  }
```

1.6.3 cc_get_O

```
int cc_get_O ( malu_op_t alu_op , sbits < 16 > res , sbits < 16 > op1 , sbits < 16 > op2 ) {
  }
```

1.6.4 cc_get_Z

```
int cc_get_Z ( sbits < 16 > res ) {
  }
```

1.6.5 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

1.6.6 flag_ways

```
bits < 9 > flag_ways ( bits < 9 > flag_registers , bits < 9 > new_flags , bits < 9 > mask ) {
   return ( flag_registers & ~ mask ) | ( new_flags & mask ) ;
  }
```

1.6.7 malu

```
void malu ( malu_op_t malu_op , bits < 3 > idx1 , sbits < 16 > op1 , bits < 3 > idx2 , sbits < 16 > op2
    int32_t sop1 [ 9 ] , sop2 [ 9 ] ;
    int i ;
    bits < 9 > z , n , c , o ;
   if ( CONFALU . SGN == 1 ) {
       for ( i = 0 ; i < 9 ; i ++ ) {
           sop1 [ i ] = MOP1 ( i ) . int32 (  ) ;
           sop2 [ i ] = MOP2 ( i ) . int32 (  ) ;
       }
   } else {
       for ( i = 0 ; i < 9 ; i ++ ) {
           sop1 [ i ] = MOP1 ( i ) . uint32 (  ) ;
           sop2 [ i ] = MOP2 ( i ) . uint32 (  ) ;
       }
   }
   c = MFLAG_C ;
   o = MFLAG_O ;
   switch ( malu_op ) {
       case MALU_OP_ADD : for ( i = 0 ; i < 9 ; i ++ ) {
           MRES ( i ) = sop1 [ i ] + sop2 [ i ] ;
           z ( i , i ) = cc_get_Z ( MRES ( i ) ) ;
           n ( i , i ) = cc_get_N ( MRES ( i ) ) ;
           c ( i , i ) = cc_get_C ( malu_op , MRES ( i ) , MOP1 ( i ) , MOP2 ( i ) ) ;
           o ( i , i ) = cc_get_O ( malu_op , MRES ( i ) , MOP1 ( i ) , MOP2 ( i ) ) ;
       }
       break ;
       case MALU_OP_SUB : for ( i = 0 ; i < 9 ; i ++ ) {
           MRES ( i ) = sop1 [ i ] - sop2 [ i ] ;
           z ( i , i ) = cc_get_Z ( MRES ( i ) ) ;
           n ( i , i ) = cc_get_N ( MRES ( i ) ) ;
           c ( i , i ) = cc_get_C ( malu_op , MRES ( i ) , MOP1 ( i ) , MOP2 ( i ) ) ;
           o ( i , i ) = cc_get_O ( malu_op , MRES ( i ) , MOP1 ( i ) , MOP2 ( i ) ) ;
       }
       break ;
       case MALU_OP_ABSDIFF : for ( i = 0 ; i < 9 ; i ++ ) {
```

```
    MRES ( i ) = ( sop1 [ i ] > sop2 [ i ] ) ? ( sop1 [ i ] - sop2 [ i ] ) : ( sop2 [ i ] - sop1
    z ( i , i ) = cc_get_Z ( MRES ( i ) ) ;
    n ( i , i ) = cc_get_N ( MRES ( i ) ) ;
}
break ;
case MALU_OP_THF : for ( i = 0 ; i < 9 ; i ++ ) {
    MRES ( i ) = ( sop1 [ i ] < sop2 [ i ] ) ? 0 : MOP1 ( i ) ;
    z ( i , i ) = cc_get_Z ( MRES ( i ) ) ;
    n ( i , i ) = cc_get_N ( MRES ( i ) ) ;
}
break ;
case MALU_OP_THM : for ( i = 0 ; i < 9 ; i ++ ) {
    MRES ( i ) = ( sop1 [ i ] < sop2 [ i ] ) ? 0 : 0xFFFF ;
    z ( i , i ) = cc_get_Z ( MRES ( i ) ) ;
    n ( i , i ) = cc_get_N ( MRES ( i ) ) ;
}
break ;
case MALU_OP_AND : for ( i = 0 ; i < 9 ; i ++ ) {
    MRES ( i ) = sop1 [ i ] & sop2 [ i ] ;
    z ( i , i ) = cc_get_Z ( MRES ( i ) ) ;
    n ( i , i ) = cc_get_N ( MRES ( i ) ) ;
}
break ;
case MALU_OP_OR : for ( i = 0 ; i < 9 ; i ++ ) {
    MRES ( i ) = sop1 [ i ] | sop2 [ i ] ;
    z ( i , i ) = cc_get_Z ( MRES ( i ) ) ;
    n ( i , i ) = cc_get_N ( MRES ( i ) ) ;
}
break ;
case MALU_OP_XOR : for ( i = 0 ; i < 9 ; i ++ ) {
    MRES ( i ) = sop1 [ i ] | sop2 [ i ] ;
    z ( i , i ) = cc_get_Z ( MRES ( i ) ) ;
    n ( i , i ) = cc_get_N ( MRES ( i ) ) ;
}
break ;
case MALU_OP_MIN : for ( i = 0 ; i < 9 ; i ++ ) {
    MRES ( i ) = ( sop1 [ i ] < sop2 [ i ] ) ? sop1 [ i ] : sop2 [ i ] ;
    z ( i , i ) = cc_get_Z ( MRES ( i ) ) ;
    n ( i , i ) = cc_get_N ( MRES ( i ) ) ;
}
break ;
case MALU_OP_MAX : for ( i = 0 ; i < 9 ; i ++ ) {
    MRES ( i ) = ( sop1 [ i ] > sop2 [ i ] ) ? sop1 [ i ] : sop2 [ i ] ;
    z ( i , i ) = cc_get_Z ( MRES ( i ) ) ;
    n ( i , i ) = cc_get_N ( MRES ( i ) ) ;
}
break ;
case MALU_OP_ASR : for ( i = 0 ; i < 9 ; i ++ ) {
    MRES ( i ) = MOP1 ( i ) . int32 (  ) >> ( sop2 [ i ] & 0xf ) ;
    z ( i , i ) = cc_get_Z ( MRES ( i ) ) ;
    n ( i , i ) = cc_get_N ( MRES ( i ) ) ;
}
break ;
case MALU_OP_LSR : for ( i = 0 ; i < 9 ; i ++ ) {
    MRES ( i ) = ( sop1 [ i ] & 0xffff ) >> ( sop2 [ i ] & 0xf ) ;
    z ( i , i ) = cc_get_Z ( MRES ( i ) ) ;
    n ( i , i ) = cc_get_N ( MRES ( i ) ) ;
}
break ;
case MALU_OP_ASL : for ( i = 0 ; i < 9 ; i ++ ) {
     int32_t sres ;
    sres = sop1 [ i ] << ( sop2 [ i ] & 0xf ) ;
    if ( sop1 [ i ] < 0 ) {
        MRES ( i ) = 0x20000 | ( ( ( sres & 0x7fff0000 ) == 0x7fff0000 ) ? 0x10000 : 0 ) | ( sre
    } else {
        MRES ( i ) = ( ( ( sres & 0x7fff0000 ) == 0 ) ? 0 : 0x10000 ) | ( sres & 0xffff ) ;
    }
    z ( i , i ) = cc_get_Z ( MRES ( i ) ) ;
    n ( i , i ) = cc_get_N ( MRES ( i ) ) ;
}
break ;
case MALU_OP_CLZ : for ( i = 0 ; i < 9 ; i ++ ) {
     int m ;
    m = 0 ;
    while ( m < 16 && ( ( sop1 [ i ] & 0x8000 ) == 0 ) ) {
        ++ m ;
        sop1 [ i ] <<= 1 ;
    }
    if ( m == 16 ) m |= 0xfff0 ;
    MRES ( i ) = m ;
    z ( i , i ) = cc_get_Z ( MRES ( i ) ) ;
    n ( i , i ) = cc_get_N ( MRES ( i ) ) ;
}
```

```
        break ;
        default : for ( i = 0 ; i < 9 ; i ++ ) {
            MRES ( i ) = MOP2 ( i ) ;
        }
        break ;
    }
    I_MFLAG_Z = z ;
    I_MFLAG_N = n ;
    I_MFLAG_C = c ;
    I_MFLAG_O = o ;
    for ( i = 0 ; i < 9 ; i ++ ) {
        if ( CONFALU . SAT == 1 ) {
            if ( CONFALU . SGN == 1 ) {
                if ( MRES ( i ) . int32 (  ) < ( - 0x8000 ) ) {
                    MSAT ( i ) = ( - 0x8000 ) ;
                } else if ( MRES ( i ) . int32 (  ) > ( 0x7FFF ) ) {
                    MSAT ( i ) = ( 0x7FFF ) ;
                } else {
                    MSAT ( i ) = ( MRES ( i ) ) ( 15 , 0 ) ;
                }
            } else {
                if ( MRES ( i ) . int32 (  ) < ( 0x0000 ) ) {
                    MSAT ( i ) = ( 0x0000 ) ;
                } else if ( MRES ( i ) . uint32 (  ) > ( 0xFFFF ) ) {
                    MSAT ( i ) = ( 0xFFFF ) ;
                } else {
                    MSAT ( i ) = ( MRES ( i ) ) ( 15 , 0 ) ;
                }
            }
        } else {
            MSAT ( i ) = ( MRES ( i ) ) ( 15 , 0 ) ;
        }
    }
}
```

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

1.6.8 *maskalu*

```
void maskalu ( maskalu_op_t maskalu_op , bits < 2 > idx1 , bits < 16 > op2 ) {
    int i ;
    MASKOP1 ( idx1 ) = 0 ;
    MASKOP2 = op2 ;
    switch ( maskalu_op ) {
        case MASKALU_OP_SEL : for ( i = 0 ; i < 9 ; i ++ ) {
            MASKRES ( i ) = ( MASKOP1 ( i ) == MASKOP2 ) ? NHOOD ( i ) : 0 ;
        }
        break ;
        case MASKALU_OP_INV : for ( i = 0 ; i < 9 ; i ++ ) {
            MASKRES ( i ) = ( MASKOP1 ( i ) == 0 ) ? NHOOD ( i ) : 0 ;
        }
        break ;
        case MASKALU_OP_THR : for ( i = 0 ; i < 9 ; i ++ ) {
            MASKRES ( i ) = ( MASKOP1 ( i ) . uint32 (  ) >= MASKOP2 . uint32 (  ) ) ? NHOOD ( i ) : 0 ;
        }
        break ;
        case MASKALU_OP_ITH : for ( i = 0 ; i < 9 ; i ++ ) {
            MASKRES ( i ) = ( MASKOP1 ( i ) . uint32 (  ) <= MASKOP2 . uint32 (  ) ) ? NHOOD ( i ) : 0 ;
        }
        break ;
        case MASKALU_OP_AND : for ( i = 0 ; i < 9 ; i ++ ) {
            MASKRES ( i ) = ( MASKOP1 ( i ) != 0 ) ? MASKOP2 ( i ) : 0 ;
        }
        break ;
        default : for ( i = 0 ; i < 9 ; i ++ ) {
            MASKRES ( i ) = MASKOP2 ( i ) ;
        }
        break ;
    }
}
```

1.6.9 *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

1.6.10 *salu_cc_set_NZ*

```
void salu_cc_set_NZ ( sbits < 16 > res ) {
  }
```

1.6.11 *salu_cc_set_NZCO*

```
void salu_cc_set_NZCO ( salu_op_t alu_op , sbits < 16 > res , sbits < 16 > op1 , sbits < 16 > op2 ) {
  }
```

1.6.12 *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_INACFG . CURRCFG = I_INACFG . NEXTCFG ;
   I_INBCFG . CURRCFG = I_INBCFG . NEXTCFG ;
   I_INALCFG . CURRCFG = I_INALCFG . 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

1.6.13 *StreamDMAStartLine*

```
void StreamDMAStartLine (  ) {
    int i ;
```

```
 for ( i = 0 ; i < 3 ; i ++ ) {
     S_CURRCHCFG_INA ( i ) = S_CHCFG_INA ( i ) ;
 }
 for ( i = 0 ; i < 3 ; i ++ ) {
     S_CURRCHCFG_INB ( i ) = S_CHCFG_INB ( i ) ;
 }
 for ( i = 0 ; i < 3 ; i ++ ) {
     S_CURRCHCFG_INAL ( i ) = S_CHCFG_INAL ( i ) ;
 }
 for ( i = 0 ; i < 3 ; i ++ ) {
     S_CURRLINE_PTR_INA ( i ) = S_LINE_PTR_INA ( i ) ;
 }
 for ( i = 0 ; i < 3 ; i ++ ) {
     S_CURRLINE_PTR_INB ( i ) = S_LINE_PTR_INB ( i ) ;
 }
 for ( i = 0 ; i < 3 ; i ++ ) {
     S_CURRLINE_PTR_INAL ( i ) = S_LINE_PTR_INAL ( i ) ;
 }
 for ( i = 0 ; i < 3 ; i ++ ) {
     S_CURRLINELEN_INA ( i ) = S_LINELEN_INA ( i ) ;
 }
 for ( i = 0 ; i < 3 ; i ++ ) {
     S_CURRLINELEN_INB ( i ) = S_LINELEN_INB ( i ) ;
 }
 for ( i = 0 ; i < 3 ; i ++ ) {
     S_CURRLINELEN_INAL ( i ) = S_LINELEN_INAL ( i ) ;
 }
 for ( i = 0 ; i < 3 ; i ++ ) {
     I_RPTCNT_INA ( i ) = S_CURRCHCFG_INA ( i ) ( 19 , 18 ) ;
 }
 for ( i = 0 ; i < 3 ; i ++ ) {
     I_RPTCNT_INB ( i ) = S_CURRCHCFG_INB ( i ) ( 19 , 18 ) ;
 }
 for ( i = 0 ; i < 3 ; i ++ ) {
     I_RPTCNT_INAL ( i ) = S_CURRCHCFG_INAL ( i ) ( 19 , 18 ) ;
 }
 for ( i = 0 ; i < 3 ; i ++ ) {
     I_POS_INA ( i ) = 0 - S_CURRCHCFG_INA ( i ) ( 26 , 24 ) ;
 }
 for ( i = 0 ; i < 3 ; i ++ ) {
     I_POS_INB ( i ) = 0 - S_CURRCHCFG_INB ( i ) ( 26 , 24 ) ;
 }
 for ( i = 0 ; i < 3 ; i ++ ) {
     I_POS_INAL ( i ) = 0 - S_CURRCHCFG_INAL ( i ) ( 26 , 24 ) ;
 }
 for ( i = 0 ; i < 4 ; i ++ ) {
     S_CURRCHCFG_OUT ( i ) = S_CHCFG_OUT ( i ) ;
 }
 for ( i = 0 ; i < 4 ; i ++ ) {
     S_CURRLINE_PTR_OUT ( i ) = S_LINE_PTR_OUT ( i ) ;
 }
 for ( i = 0 ; i < 4 ; i ++ ) {
     S_CURRLINELEN_OUT ( i ) = S_LINELEN_OUT ( i ) ;
 }
 for ( i = 0 ; i < 4 ; i ++ ) {
     I_POS_OUT ( i ) = 0 ;
 }
 for ( i = 0 ; i < 4 ; i ++ ) {
     I_SKIPCNT_OUT ( i ) = 0 ;
 }
}
```

View document source. Generated on: 2018/02/23 20:29:41 CET. Generated by prest release 0.3.40 from reStructuredText source.