Instruction set reference

Table of contents

Revisions.

Rev. 1. 24.10.2023 Added reference of the XCHG instruction. Improved description of all other instructions.

Instruction table

	xF	xE	xD	xC	xB	xA	x9	x8	x7	x6	x5	x4	x3	x2	x1	x0	
Fx	NOP																Fx
Ex																	Ex
Dx																	Dx
Cx			SLEEP	BKPT	CALLI	JUMPI	ENDMSG	RET	MEM ALLOC	LOOP	JNC	JC	CALLR	JUMPR	GETPAR	SEND MSG	Cx
Bx																	Bx
Ax																	Ax
9x															FFT	FMULACC	9x
8x				AAR	IAR	LIA	SAR	LAR	POPA	PUSHA	POPD	PUSHD			LD	ST	8x
7x																	7x
6x																	6x
5x											ASRI		CSRI	CSLI	LSRI	LSLI	5x
4x		XCHG	LFR	SFR	NOT	LID	COPYSX	COPYZX	INT2FP	FP2INT	POS	RND	BSWAP	NEG	DAS	DAA	4x
3x																	3x
2x																	2x
1x					SQRT	FDIV		FMUL	FIELD GET	FIELD SET	ASR		CSR	CSL	LSR	LSL	1x
0x	FSUB	FADD	FMAX	FMIN	DIVSX	DIVZX	MULSX	MULZX	MASK COPY	XOR	OR	AND	SUBSX	SUBZX	ADDSX	ADDZX	0x
	xF	xE	xD	xC	xB	xA	x9	x8	x7	x6	x5	x4	x3	x2	x1	x0	

Note 1. All instruction OpCodes in the empty positions are available for user-defined instructions X32Carrier core.

Note 2. X32Carrier do not have FMULACC and FFT instructions and their OpCodes are available for user-defined instructions.

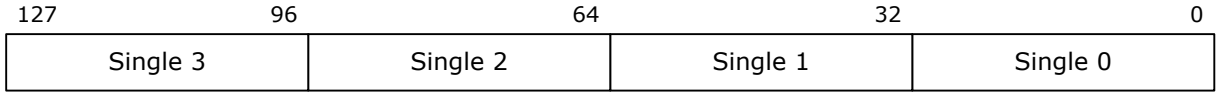
Instruction field definitions

Mnemonic	***Description***
R0,R1,R2…R31	GPR definition without complete reference of the operand size or type.
Integer style register description (idst, isrc1, isrc2)	
RB0,RB1…RB31	8-bit operand definition.
RW0,RW1…RW31	Word-wide operand definition.
RD0,RD1…RD31	Double word operand definition.
RQ0,RQ1…RQ31	64-bit operand definition.
Floating point scalar style register description (fdst, fsrc1, fsrc2)	
RFS0,RFS1…RFS31	Single precision floating point operand (32 bit).
RFD0,RFD1…RFD31	Double precision floating point operand (64 bit).
RFE0,RFE1…RFE31	Extended precision floating point operand (128 bit).
Vector floating point style register description (vdst, vsrc1, vsrc2)	
RVS0,RVS1…RVS31	Single precision 4-numbers vector.
RVD0,RVD1…RVD31	Double precision 2-numbers vector.
Address and flag registers	
AR0,AR1…AR15	Address registers.
W0,W1,W2…W7	16-bit word position definition for loading him into the GPR or ADR.
MAR0,MAR1…MAR7	Address register's pair description in the memory load/store operations.
CF,ZF,SF,OF,IF,NF,DF	Arithmetic and logic operations flags.

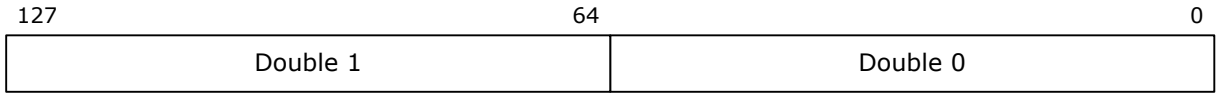
Operations on vector operands

Vector operands can be of 2 types - 4 single precision numbers packed in a 128-bit general purpose register or 2 double precision numbers.

RVS0 … RVS31

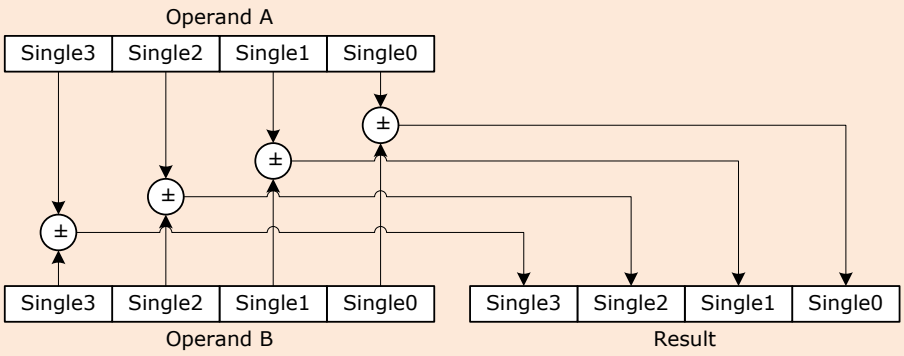
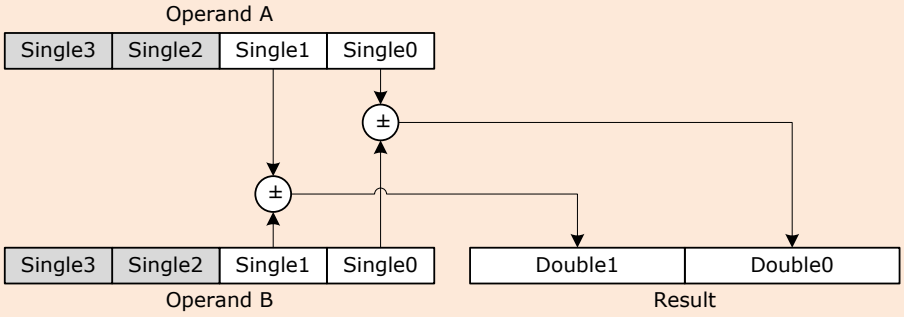
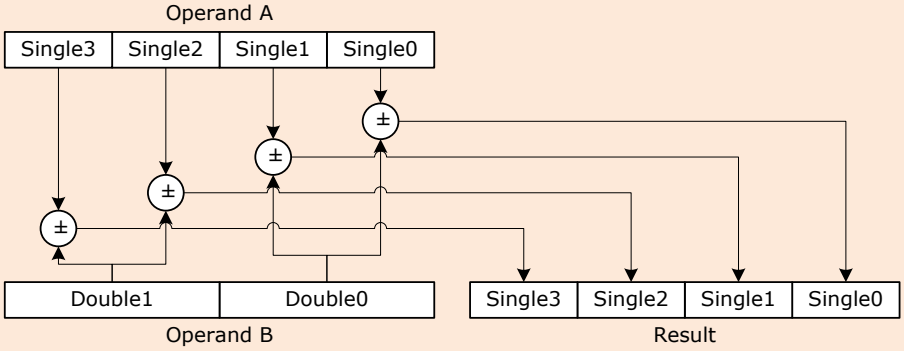
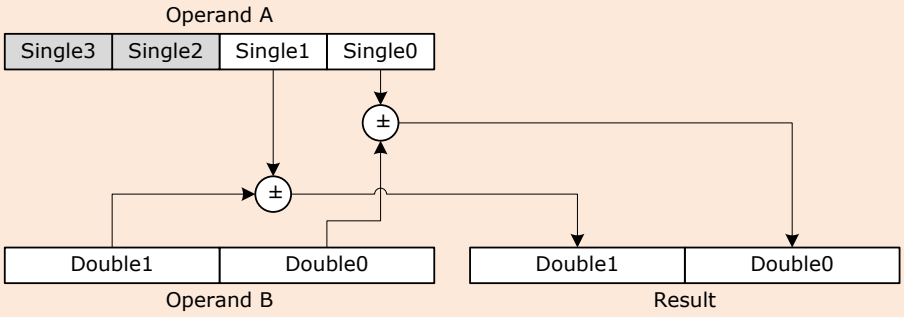


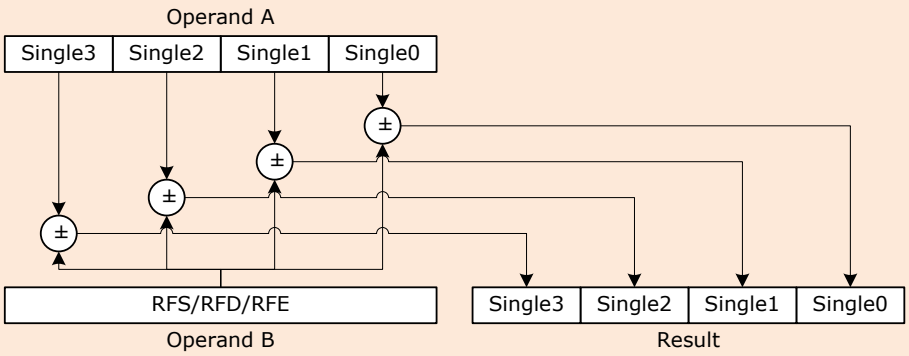
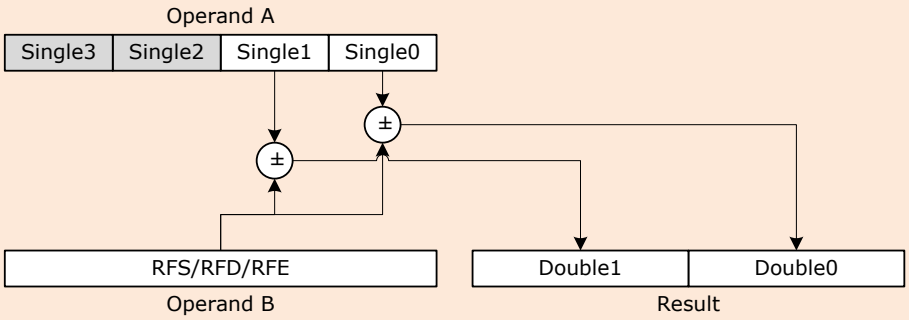
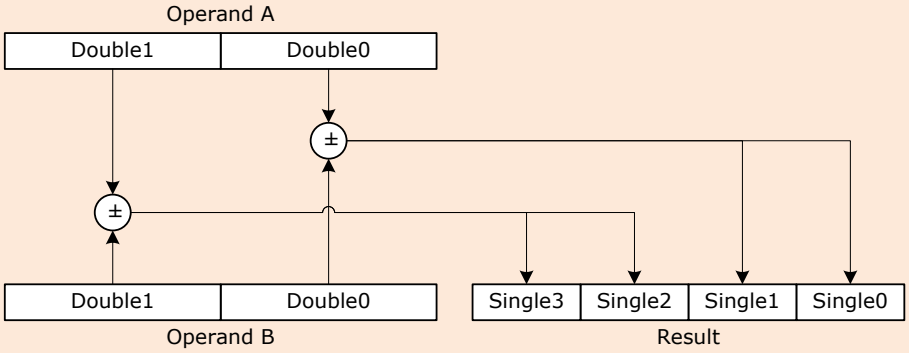
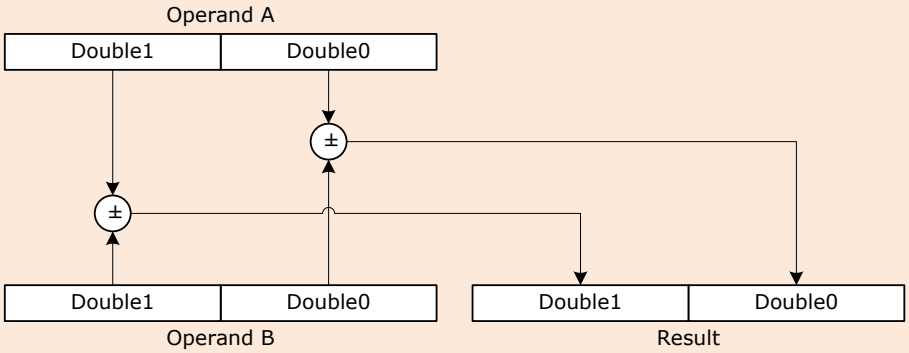
RVD0 … RVD31

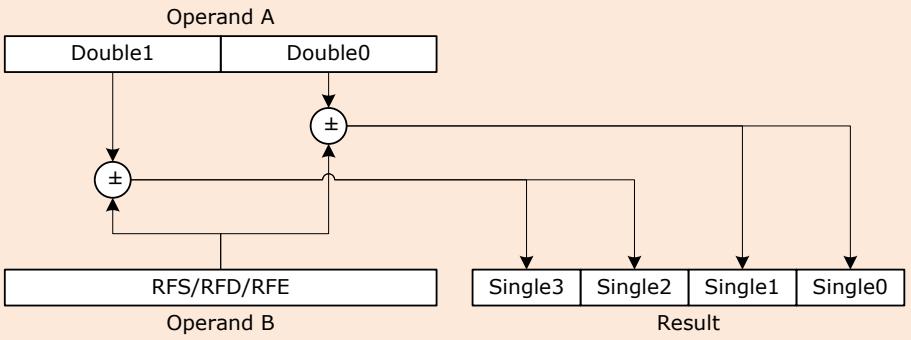
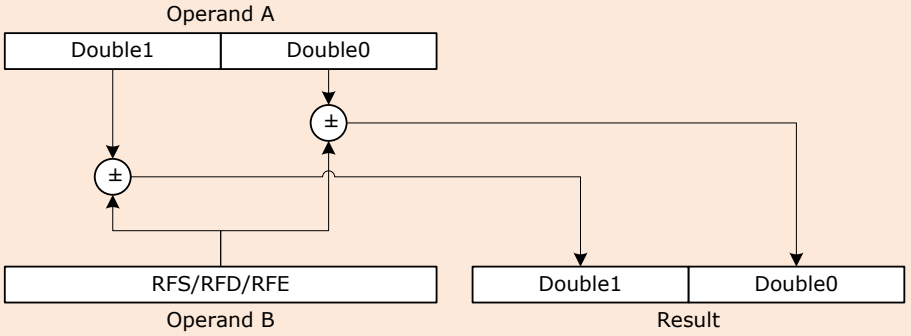


The FADD, FSUB, FMUL, and FDIV instructions can process 4-single or 2-double vector operands that can be stored in 128-bit general purpose registers. Operations are possible with vectors of the same format, with a vector and a scalar, as well as with vectors of different formats. The table below describes the logic of the processor with vector operands depending on the types of sources and the type of result destination.

Op. A	Op. B	Desired format of the result	Description of the result of the operation

Op. A	Op. B	Desired format of the result	Description of the result of the operation
RVS	RVS	RVS	
RVS	RVS	RVD	
RVS	RVS	RFS RFD RFE	Same as scalar operation between two single precision operands
RVS	RVD	RVS	
RVS	RVD	RVD	
RVS	RVD	RFS RFD RFE	Same as scalar operation between single and double precision operands

Op. A	Op. B	Desired format of the result	Description of the result of the operation
RVS	RFS RFD RFE	RVS	
RVS	RFS RFD RFE	RVD	
RVS	RFS RFD RFE	RFS RFD RFE	Same as scalar operation between single and single/double/extended precision operands
RVD	RVD	RVS	
RVD	RVD	RVD	
RVD	RVD	RFS RFD RFE	Same as scalar operation between two double precision operands

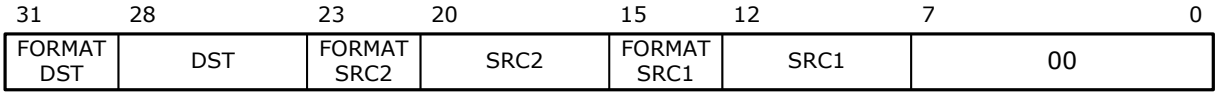
Op. A	Op. B	Desired format of the result	Description of the result of the operation
RVD	RFS RFD RFE	RVS	
RVD	RFS RFD RFE	RVD	
RVD	RFS RFD RFE	RFS RFD RFE	Same as scalar operation between double and single/double/extended precision operands

ADDZX – Addition zero-extended

Mnemonic:

ADDZX idst,isrc1:isrc2

Format:



Description.

Integer addition with operands extension by zeroed bits. Source operands can have a different size. Both source operands expand up to 64 bit by means of bits with zero state in left. Result's width is pointed by the Format DST field.

Altered flags in AFR[dst]:

CF[15:0], ZF, SF, OF

Example:

ADDZX Rw5,Rw6:Rb23

Exceptions:

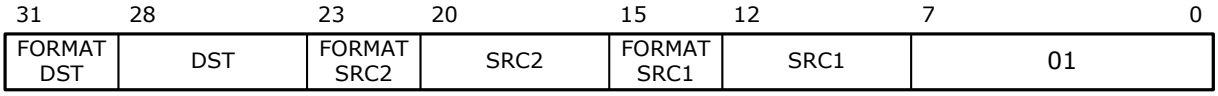
None.

ADDSX – Addition, sign-extended

Mnemonic:

ADDSX idst,isrc1:isrc2

Format:



Description.

Integer addition with extension of the source operands by their sign bits. Source operands can have different bit depths.

Altered flags in AFR[dst]:

CF[15:0], ZF, SF, OF

Example:

ADDSX Rq24,Rb10:Rd20

Exceptions:

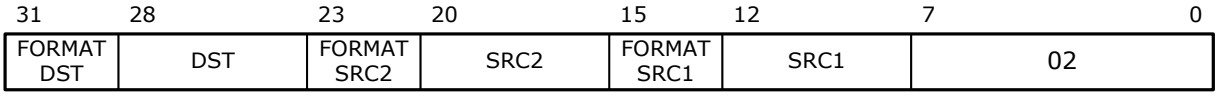
None.

SUBZX – Subtraction, zero-extended operands

Mnemonic:

SUBZX idst,isrc1:isrc2

Format:



Description.

Integer subtraction. Both source operands expand up to 64 bit by means of bits with zero state in left. Dst=src1-src2

Altered flags in AFR[dst]:

CF[15:0], ZF, SF, OF

Example:

SUBZX Rb6,Rb10:Rb11

Exceptions:

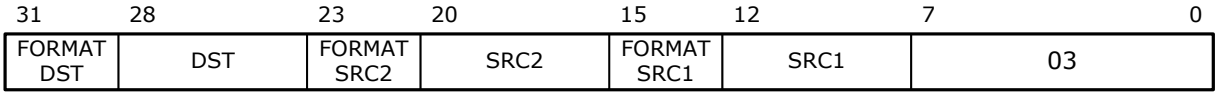
None.

SUBSX – Subtraction, sign-extended operands

Mnemonic:

SUBSX idst,isrc1:isrc2

Format:



Description.

Integer subtraction with extension of the source operands by their sign bits. Dst=src1-src2

Altered flags in AFR[dst]:

CF[15:0], ZF, SF, OF

Example:

SUBSX Rb6,Rb21:Rb11

Exceptions:

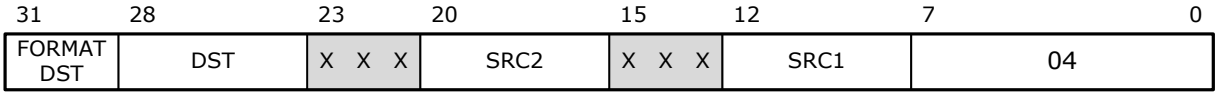
None.

AND – logical “and”

Mnemonic:

AND idst,isrc1:isrc2

Format:



Description.

Logical «AND». Formats of the source operands don’t have a matter because instruction always uses a 64-bit operands. “Format DST” field describes which is a part of the result should be written into destination register.

Altered flags in AFR[dst]:

CF[15:0], ZF, SF, OF

Example:

AND Rd7,Rd11:Rd25

Exceptions:

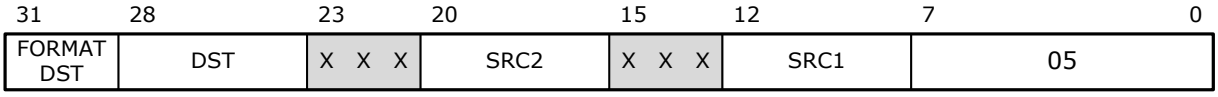
None.

OR – logical “or”

Mnemonic:

OR idst,isrc1:isrc2

Format:



Description.

Logical «OR».

Altered flags in AFR[dst]:

CF[15:0], ZF, SF, OF

Example:

OR Rw17,Rw21:Rw5

Exceptions:

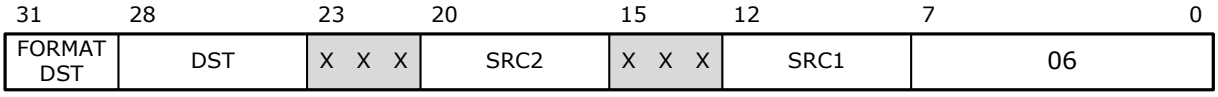
None.

XOR – exclusive “or”

Mnemonic:

XOR idst,isrc1:isrc2

Format:



Description.

Exclusive «OR» operation.

Altered flags in AFR[dst]:

CF[15:0], ZF, SF, OF

Example:

XOR Rw4,Rw1:Rw9

Exceptions:

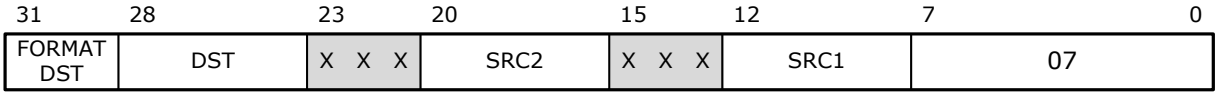
None.

MASKCOPY – masked bit field copy

Mnemonic:

MASKCOPY idst,isrc1:isrc2

Format:



Description.

The bits from the register R[src1] are copied to the corresponding bits of the register R[dst] if the corresponding bits of the register R[src2] are set to 1.

Altered flags in AFR[dst]:

CF[15:0], ZF, SF, OF

Example:

MASKCOPY Rd30,Rw1:R20

Exceptions:

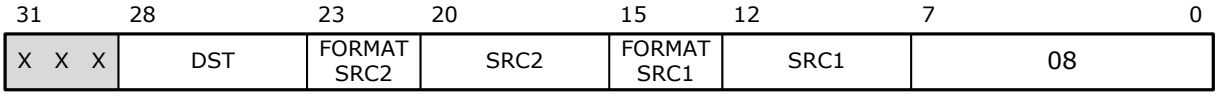
None.

MULZX – integer multiplication, unsigned

Mnemonic:

MULZX idst,isrc1:isrc2

Format:



Description.

Unsigned integer multiplication. Depth of the result depends of depths of the source operands.

Operand 1	***Operand 2***	***Result***
Byte	Byte	Word
Byte	Word	Dword
Byte	Dword	Qword
Byte	Qword	Qword
Word	Word	Dword
Word	Dword	Qword
Word	Qword	Qword
Dword	Dword	Qword
Dword	Qword	Qword
Qword	Qword	Qword

Altered flags in AFR[dst]:

ZF, SF.

Example:

MULZX R0,Rb1:Rd2

Exceptions:

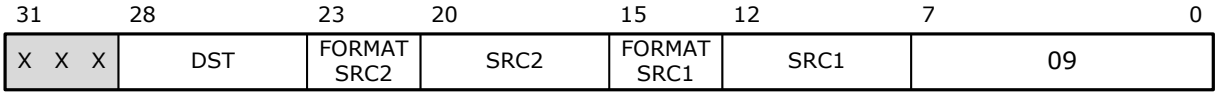
None.

MULSX – integer multiplication, signed

Mnemonic:

MULSX idst,isrc1:isrc2

Format:



Description.

Signed integer multiplication. Depth of the result depends of depths of the source operands.

Operand 1	***Operand 2***	***Result***
Byte	Byte	Word
Byte	Word	Dword
Byte	Dword	Qword
Byte	Qword	Qword
Word	Word	Dword
Word	Dword	Qword
Word	Qword	Qword
Dword	Dword	Qword
Dword	Qword	Qword
Qword	Qword	Qword

Altered flags in AFR[dst]:

ZF, SF.

Example:

MULSX R0,Rb1:Rd2

Exceptions:

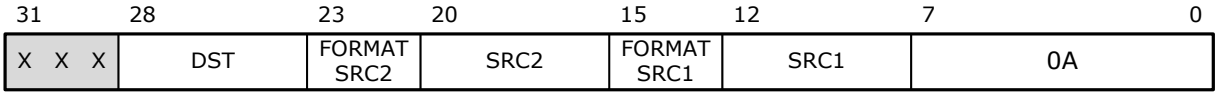
None.

DIVZX – integer division, unsigned

Mnemonic:

DIVZX idst,isrc1:isrc2

Format:



Description.

Unsigned integer division. SRC1 contains a dividend and SRC2 has a divisor. Result's format depends of depth of the SRC1 operand.

Altered flags in AFR[dst]:

ZF, SF.

Example:

```
DIVZX   R0,Rd1:Rb2      ; division R0=R1/R2
```

Exceptions:

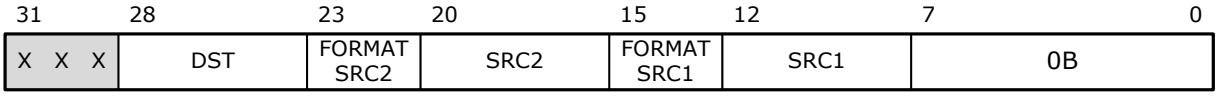
None.

DIVSX – integer division, signed

Mnemonic:

DIVSX idst,isrc1:isrc2

Format:



Description.

Integer signed division. SRC1 contains a dividend. SRC2 contains a divisor. Result's depth depends of the dividend depth.

Altered flags in AFR[dst]:

ZF, SF.

Example:

DIVSX R0,Rd1:Rw2

Exceptions:

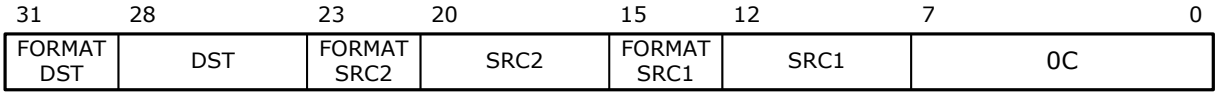
None.

FMIN – Floating point minimum

Mnemonic:

FMIN fdst,fsrc1:fsrc2

Format:



Description.

This instruction selects a minimum value from two source operands and place it to the destination register in a desired format.

Altered flags in AFR[dst]:

ZF, SF, IF, NF.

Example:

FMIN Rfe20,Rfs1:Rfd28

Exceptions:

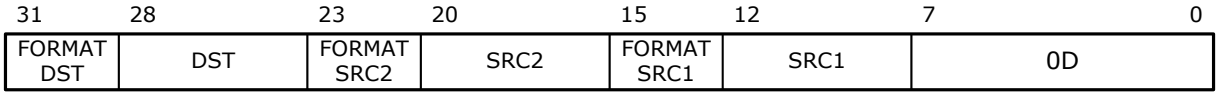
None.

FMAX – Floating point maximum

Mnemonic:

FMAX fdst,fsrc1:fsrc2

Format:



Description.

This instruction selects a maximum value from two source operands and place it to the destination register in a desired format.

Altered flags in AFR[dst]:

ZF, SF, IF, NF.

Example:

FMAX Rfs2,Rfd11:Rfd8

Exceptions:

None.

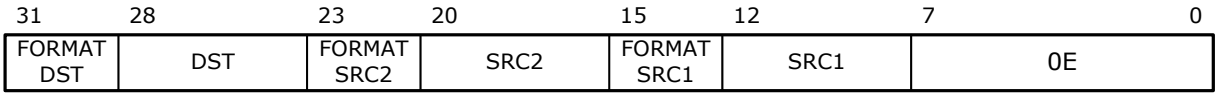
FADD – floating point addition

Mnemonic:

FADD fdst,fsrc1:fsrc2

FADD vdst,vsrc1:vsrc2

Format:



Description.

Floating point addition. Source operands could be in single, double or extended precision. Result can be written in any of these formats.

Altered flags in AFR[dst]:

ZF, SF, IF, NF.

Example:

FADD Rfs30,Rfe1:Rfd20

Exceptions:

None.

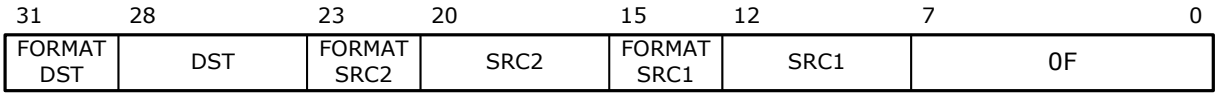
FSUB – floating point subtraction

Mnemonic:

FSUB fdst,fsrc1:fsrc2

FSUB vdst,vsrc1:vsrc2

Format:



Description.

Floating point subtraction. Fdst=fsrc1-fsrc2

Altered flags in AFR[dst]:

ZF, SF, IF, NF.

Example:

FSUB Rfs30,Rfs1:Rfd20

Exceptions:

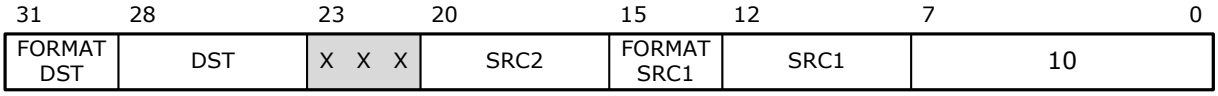
None.

LSL – logical shift left

Mnemonic:

LSL idst,isrc1:isrc2

Format:



Description.

Logical shift left by variable number of bits. Number of shifted bits determines by SRC2 content.

Altered flags in AFR[dst]:

ZF, SF, OF, DF.

Example:

LSL Rd26,Rd25:R2

Exceptions:

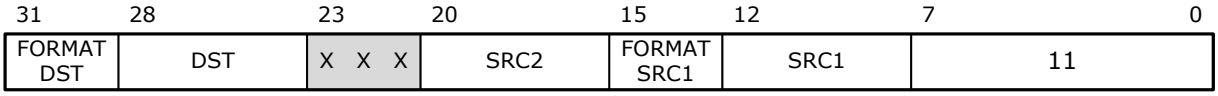
None.

LSR – logical shift right

Mnemonic:

LSR idst,isrc1:isrc2

Format:



Description.

Logical shift right by variable number of bits. Number of shifted bits determines by SRC2 content.

Altered flags in AFR[dst]:

ZF, SF, OF, DF.

Example:

LSR Rd26,Rd25:R2

Exceptions:

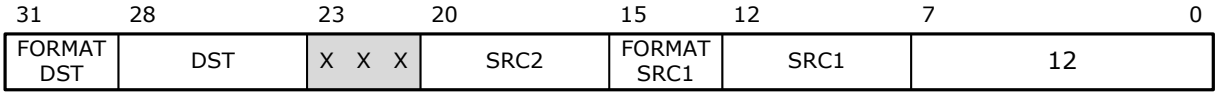
None.

CSL – cyclic shift left

Mnemonic:

CSL idst,isrc1:isrc2

Format:



Description.

Cyclic shift left.

Altered flags in AFR[dst]:

ZF, SF, OF, DF.

Example:

CSL Rd26,Rd25:R2

Exceptions:

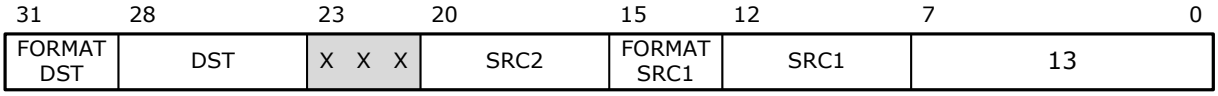
None.

CSR – cyclic shift right

Mnemonic:

CSR idst,isrc1:isrc2

Format:



Description.

Cyclic shift right.

Altered flags in AFR[dst]:

ZF, SF, OF, DF.

Example:

CSR Rd26,Rd25:R2

Exceptions:

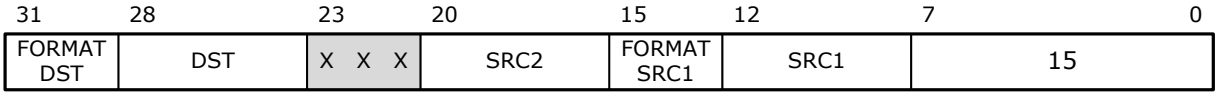
None.

ASR – arithmetic shift right

Mnemonic:

ASR idst,isrc1:isrc2

Format:



Description.

Arithmetic shift right.

Altered flags in AFR[dst]:

ZF, SF, OF, DF.

Example:

ASR Rd26,Rd25:R2

Exceptions:

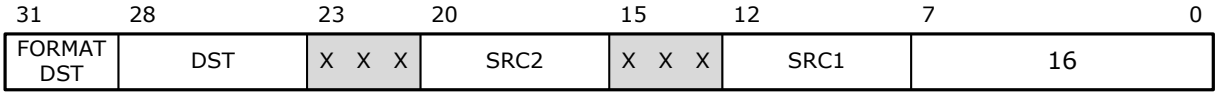
None.

FIELDSET – set field in the register

Mnemonic:

FIELDSET idst,isrc1:isrc2

Format:



Description.

Bits from the register R[src1], starting from zero, are copied to the bits of the register R[dst]. The register R[src2] contains in the low byte the index of the first bit in the register R[dst] where the bit field will be set, and the byte [15:8] contains the number of copied bits.

Altered flags in AFR[dst]:

ZF, SF.

Example:

FIELDSET Rd30,R1:R20

Exceptions:

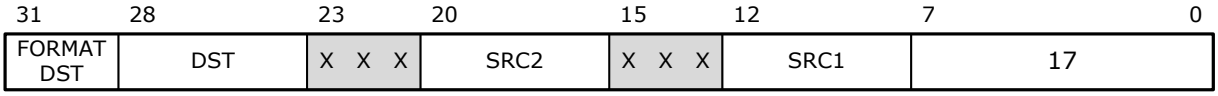
None.

FIELDGET – get the bit field

Mnemonic:

FIELDGET idst,isrc1:isrc2

Format:



Description.

The bits [N+L-1:N] from the register R[src1] are copied to the bits [L-1:0] of the register R[dst]. Byte [15:8] of register R[src2] contains the number of bits L to be copied, and byte [7:0] contains the position of the first copied bit N.

Altered flags in AFR[dst]:

ZF, SF.

Example:

FIELDGET Rd30,R1:R20

Exceptions:

None.

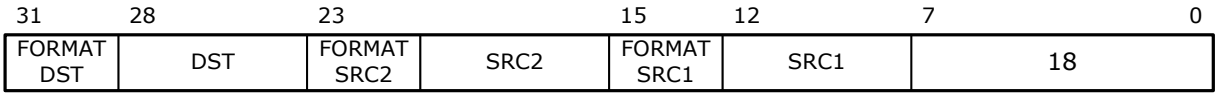
FMUL – floating point multiplication

Mnemonic:

FMUL fdst,fsrc1:fsrc2

FMUL vdst,vsrc1:vsrc2

Format:



Description.

Floating point multiplication.

Altered flags in AFR[dst]:

ZF, SF, IF, NF.

Example:

FMUL Rfs30,Rfs1:Rfd20

Exceptions:

None.

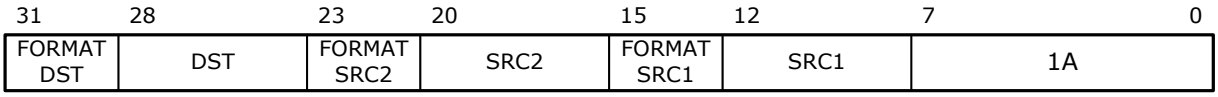
FDIV – floating point division.

Mnemonic:

FDIV fdst,fsrc1:fsrc2

FDIV vdst,vsrc1:vsrc2

Format:



Description.

Floating point division.

Altered flags in AFR[dst]:

ZF, SF, IF, NF.

Example:

FDIV Rfs30,Rfs1:Rfd20

Exceptions:

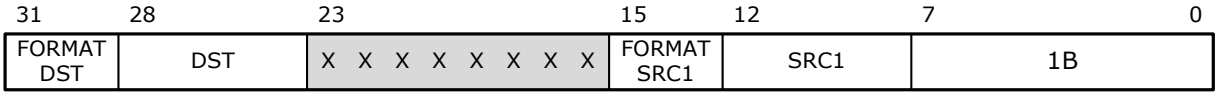
None.

SQRT – square root

Mnemonic:

SQRT fdst,fsrc1

Format:



Description.

Square root calculation. Instruction can be applied only to the numbers in a floating point representation.

Altered flags in AFR[dst]:

ZF, SF, IF, NF.

Example:

SQRT Rfs30,Rfs1

Exceptions:

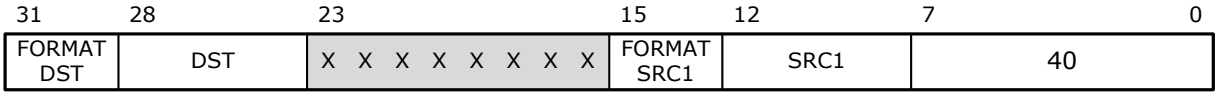
None.

DAA – decimal adjust after addition

Mnemonic:

DAA idst,isrc1

Format:



Description.

The instruction corrects the result of adding BCD numbers to obtain the correct result value.

Altered flags in AFR[dst]:

CF, ZF, SF.

Example:

DAA Rd3,Rd17

Exceptions:

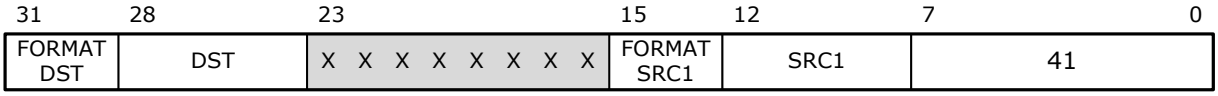
None.

DAS – decimal adjust after subtraction

Mnemonic:

DAS idst,isrc1

Format:



Description.

The instruction corrects the result of subtraction BCD numbers to obtain the correct result value.

Altered flags in AFR[dst]:

CF, ZF, SF.

Example:

DAS Rd23,Rd7

Exceptions:

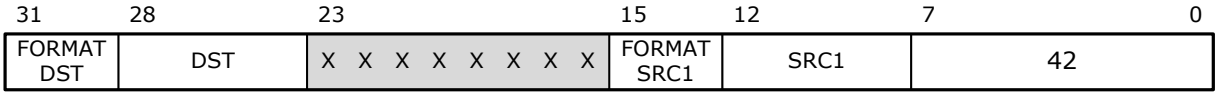
None.

NEG –negation

Mnemonic:

NEG idst,isrc1

Format:



Description.

Change the sign of an integer operand. If the receiver format is larger than the source format, then the missing bits on the left are filled with a result sign. If the receiver depth is less than the source depth, then only the selected result bits are written to the register.

Altered flags in AFR[dst]:

CF, ZF, SF.

Example:

NEG Rd2,Rd17

Exceptions:

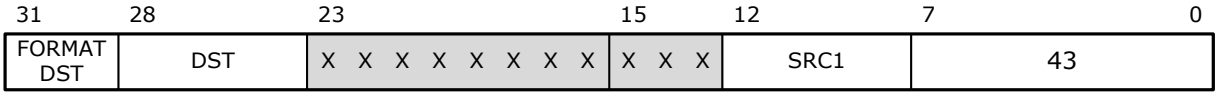
None.

BSWAP – bit swapping

Mnemonic:

BSWAP idst,isrc1

Format:



Description.

Bit swapping. For example, in a 16-bit operand, bits 0 and 15, 1 and 14, 2 and 13, etc. are interchanged.

Altered flags in AFR[dst]:

CF, ZF, SF.

Example:

BSWAP Rw5,R23

Exceptions:

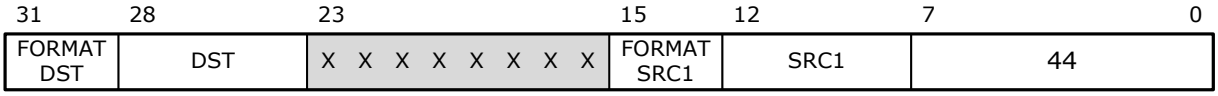
None.

RND – Round.

Mnemonic:

RND fdst,fsrc1

Format:



Description.

Round to the nearest integer. Operation can be performed only on floating point numbers.

Altered flags in AFR[dst]:

ZF, SF, IF, NF.

Example:

RND Rfd5,Rfe23

Exceptions:

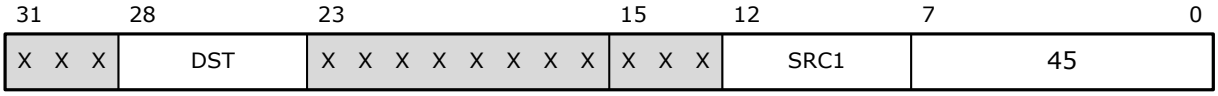
None.

POS – high bit position

Mnemonic:

POS idst,isrc1

Format:



Description.

Calculation of the position number of the high bit set to 1.

Altered flags in AFR[dst]:

ZF, SF.

Example:

POS Rfd5,Rfe23

Exceptions:

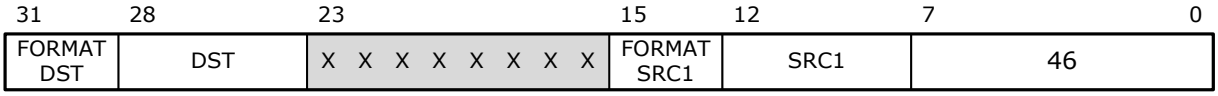
None.

FP2INT – floating point to integer

Mnemonic:

FP2INT idst,fsrc1

Format:



Description.

Number's conversion from a floating point format to the integer format. Source values less than 1.0 gives zero's as a result. The overflow flag is set to 1 if the number cannot be represented in integer format due to the large value.

Altered flags in AFR[dst]:

ZF, SF, IF.

Example:

FP2INT Rq5,Rfe23

Exceptions:

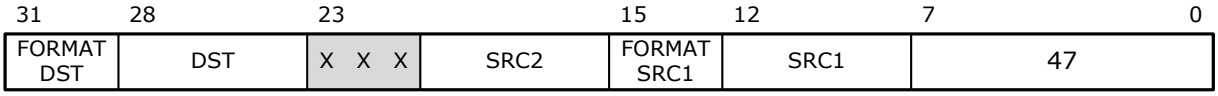
None.

INT2FP – integer to floating point conversion

Mnemonic:

INT2FP fdst,isrc1:isrc2

Format:



Description.

The instruction is intended for converting numbers from a signed integer format to a floating point format. R [src1] contains the value of the original integer operand, and the register R [src2] contains an integer value that is added to the order of the exponent after conversion.

Altered flags in AFR[dst]:

ZF, SF.

Example:

INT2FP Rfs7,Rw23:R22

Exceptions:

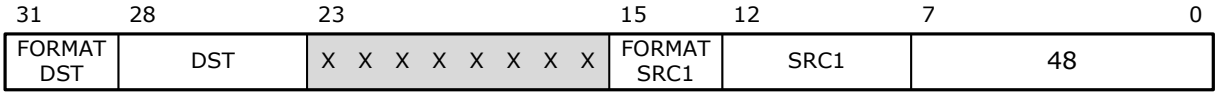
None.

COPYZX – copy register to register with zero extension

Mnemonic:

COPYZX dst,src1

Format:



Description.

The contents of register R[src1] are copied to register R[dst]. The contents of the companion flag register are also copied. If the format of the source operand is smaller than the format of the receiver of the result, then the original number is padded with zeros in the missing positions of the most significant bits.

Altered flags in AFR[dst]:

All flags are copied from the source AFR.

Example:

COPYZX Rq12,Rw14

Exceptions:

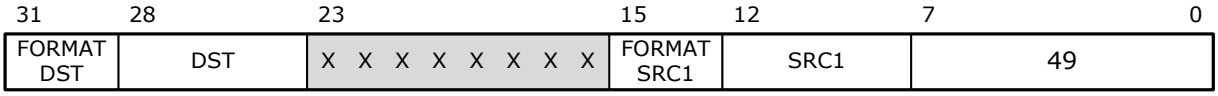
None.

COPYSX – copy with sign extension

Mnemonic:

COPYSX dst,src1

Format:



Description.

Copy data with a sign bit extension if the recipient format is wider than the source format.

Altered flags in AFR[dst]:

All flags are copied from the source AFR.

Example:

COPYSX Rq12,Rw14

Exceptions:

None.

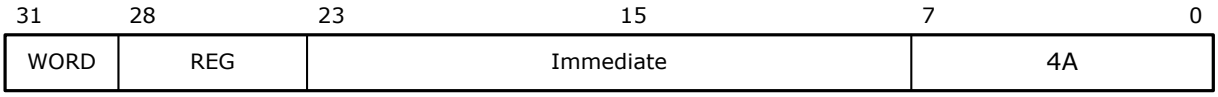
LID – load immediate value into data register

Mnemonic:

LID dst:WN,data

Where WN may be w0,w1,w2,w3,w4,w5,w6 or w7.

Format:



Description.

Load 16 bits immediate data to the register. The word number /WN/ is determined in bits [31:29] of the instruction. Downloading constants longer than 16 bits is performed in several steps. For example, to load a 128-bit constant into the register, you need to execute 8 LID commands sequence.

When writing 16-bit data to a register, all register bits to the left of the most significant one being modified are set to 0 or 1, depending on bit 15 of the immediate data. For example, if the code 8000h is written to word w1 of register, then bits 127:32 will be set to 1. This sign extension allows you to quickly load short signed constants into the register. For example, to set the constant -100 in the register, you need to write the value 0FF9Ch into word 0 of the register and all register bits from 16 to 127 will be set to 1.

Altered flags in AFR[dst]:

AFR[dst] doesn’t altered.

Example:

LID R12:w2,0FEDCh

Exceptions:

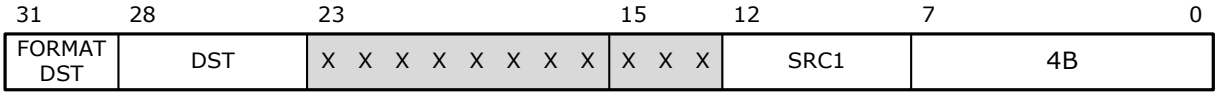
None.

NOT – inversion

Mnemonic:

NOT idst,isrc1

Format:



Description.

Bitwise inversion of the operand. The depth of the original operand does not matter.

Altered flags in AFR[dst]:

ZF, SF.

Example:

NOT Rq7,R23

Exceptions:

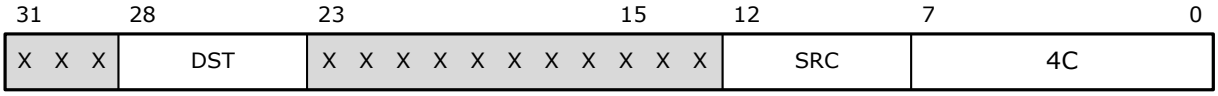
None.

SFR – store flag register

Mnemonic:

SFR idst,src1

Format:



Description.

Content of the flag register AFR[src] stores in the GPR[dst].

Altered flags in AFR[dst]:

AFR[dst] doesn’t altered.

Example:

SFR R12,R12

Exceptions:

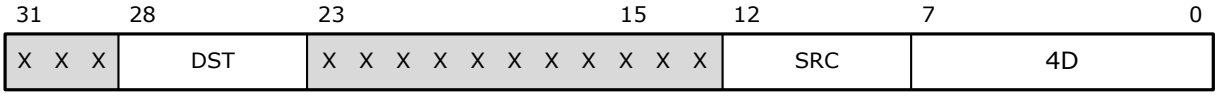
None.

LFR – load flag register

Mnemonic:

LFR dst,isrc1

Format:



Description.

Instruction loads flag register from the general-purpose register.

Altered flags in AFR[dst]:

All flags loaded from R[src].

Example:

LFR R12,R11

Exceptions:

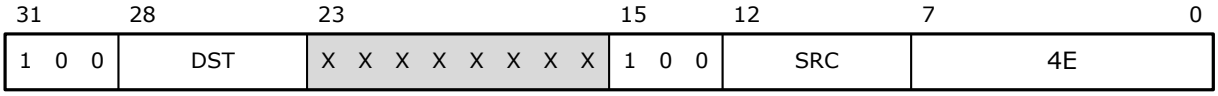
None.

XCHG – exchange low and hi 64-bit words in the 128-bit register

Mnemonic:

XCHG dst,src1

Format:



Description.

The instruction swaps bits 63:0 and 127:64. Can be used to simplify work with vector data that uses the upper 64 bits of registers.

Altered flags in AFR[dst]:

ZF, SF, OF, DF.

Example:

XCHG R26,R25

Exceptions:

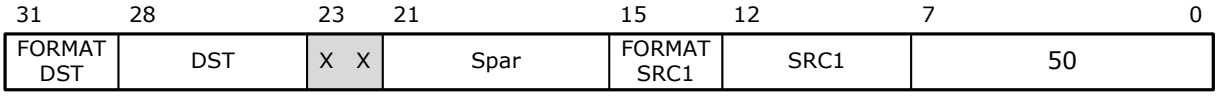
None.

LSLI – logical shift left by immediate shift parameter

Mnemonic:

LSLI idst,isrc1:Spar

Format:



Description.

Logical shift left. Shift parameter pointed in the instruction code.

Altered flags in AFR[dst]:

ZF, SF, OF, DF.

Example:

LSLI Rd26,Rd25:22

Exceptions:

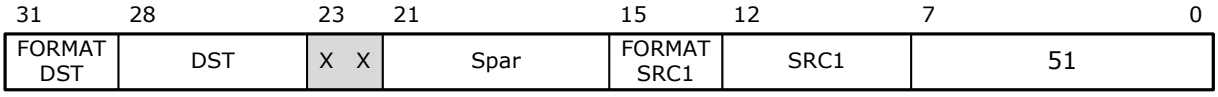
None.

LSRI – logical shift right by immediate shift parameter

Mnemonic:

LSRI idst,isrc1:Spar

Format:



Description.

Logical shift right. Shift parameter pointed in the instruction code.

Altered flags in AFR[dst]:

ZF, SF, OF, DF.

Example:

LSRI Rd26,Rd25:22

Exceptions:

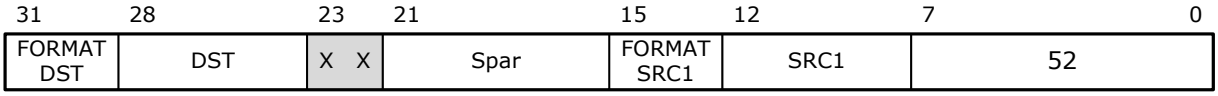
None.

CSLI – cyclic shift left by immediate parameter

Mnemonic:

CSLI idst,isrc1:Spar

Format:



Description.

Cyclic shift left. Shift parameter pointed in the instruction code.

Altered flags in AFR[dst]:

ZF, SF, OF, DF.

Example:

CSLI Rb6,Rb5:2

Exceptions:

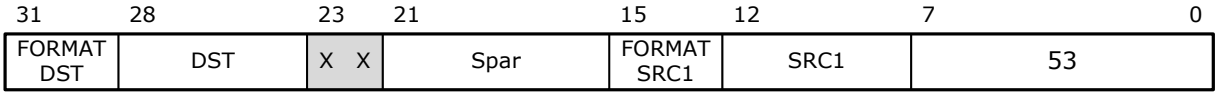
None.

CSRI – cyclic shift right by immediate parameter

Mnemonic:

CSRI idst,isrc1:Spar

Format:



Description.

Cyclic shift right. Shift parameter pointed in the instruction code.

Altered flags in AFR[dst]:

ZF, SF, OF, DF.

Example:

CSRI Rw13,Rw3:12

Exceptions:

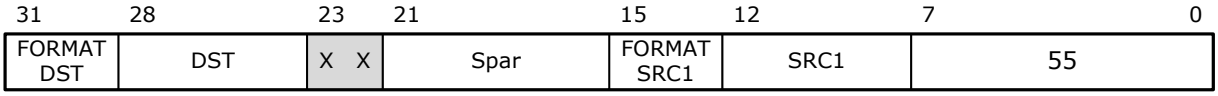
None.

ASRI – arithmetic shift right by immediate parameter

Mnemonic:

ASRI idst,isrc1:Spar

Format:



Description.

Arithmetic shift right. Shift parameter pointed in the instruction code. Sign bit of the source operand copied into all shifted-in bits.

Altered flags in AFR[dst]:

ZF, SF, OF, DF.

Example:

ASRI Rw13,Rw3:12

Exceptions:

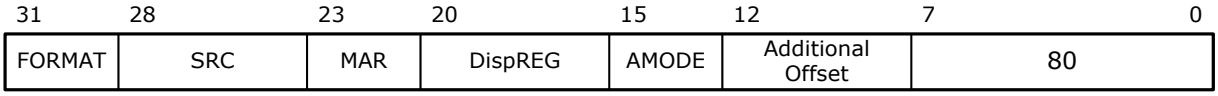
None.

ST – store data

Mnemonic:

ST MAR:DispREG:AdditionalOffset,AMODE,SRC

Format:



Description.

Instruction stores data from GPR into memory location.

Additional offset is expressed not in bytes, but in data elements - bytes, words, double words, 64-bit words or 128-bit words, depending on the specified bit depth of the transmitted data element. The additional offset is a signed number and allows you to adjust the offset both upward and downward. For example, if a 32-bit value is written to the memory and the instruction contains the 1Ah code in bits [12:8], then the value of the additional offset will be -24.

The mode of formation of the resulting offset is indicated directly in the instruction code and encoded in accordance with the table:

AMODE	***Mnemonic***	***Reference***
0	[AR]	address register fully determines the offset
1	[AR] AR=AR+R	The offset is formed only from the contents of the address register, and the address register is incremented by the value from the general-purpose register.
2	[R]	An offset is the contents of a general-purpose register.
3	[AR+R]	The offset is formed by adding the contents of the address register and the general-purpose register.
4	[AR] AR=AR+OS	The offset is retrieved from the address register. The contents of the address register are increased by the number of bytes that make up the data element.
5	[AR] AR=AR-OS	The offset is retrieved from the address register. The contents of the address register are reduced by the number of bytes that make up the data element.
6	[AR+R] AR=AR+OS	The offset is formed by adding the contents of the address register and the general-purpose register. After the operation, the contents of the address register is increased by the number of bytes that make up the data element.
7	[AR+R] AR=AR-OS	The offset is formed by adding the contents of the address register and the general-purpose register. After the operation is completed, the contents of the address register are reduced by the number of bytes constituting the data element.

An additional offset in the table is not indicated, since it always participates in the formation of the resulting offset.

The DispREG field defines the general-purpose register, the contents of which are used to form the resulting offset or to obtain a new value in the address register.

Altered flags in AFR[dst]:

Any AFRs don’t alter.

Example:

```
; instruction parameters: mar:DispREG:Additional Offset,AMODE,SRC

ST          mar3:r13:2,2,Rd23
```

Exceptions:

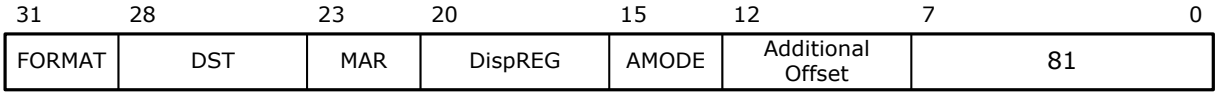
1. Object limits violation.
2. Illegal object selector.
3. Illegal object type.
4. Privilege level violation.
5. Read or write access violation.
6. TaskID violation.
7. Object can’t be accessed through multiprocessor network.
8. Processor is absent in the multiprocessor network.

LD – load data

Mnemonic:

LD DST,MAR:DispREG:AdditionalOffset,AMODE

Format:



Description.

Loading a data element from memory into a register.

Altered flags in AFR[dst]:

Any AFRs don’t alter.

Example:

```
; instruction parameters: DST,mar:DispREG:Additional Offset,AMODE
LD      Rw4,mar4:r23:0,2
```

Exceptions:

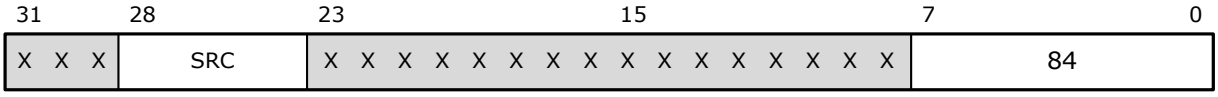
1. Object limits violation.
2. Illegal object selector.
3. Illegal object type.
4. Privilege level violation.
5. Read or write access violation.
6. TaskID violation.
7. Object can’t be accessed through multiprocessor network.
8. Processor is absent in the multiprocessor network.

PUSHD – push data register into stack

Mnemonic:

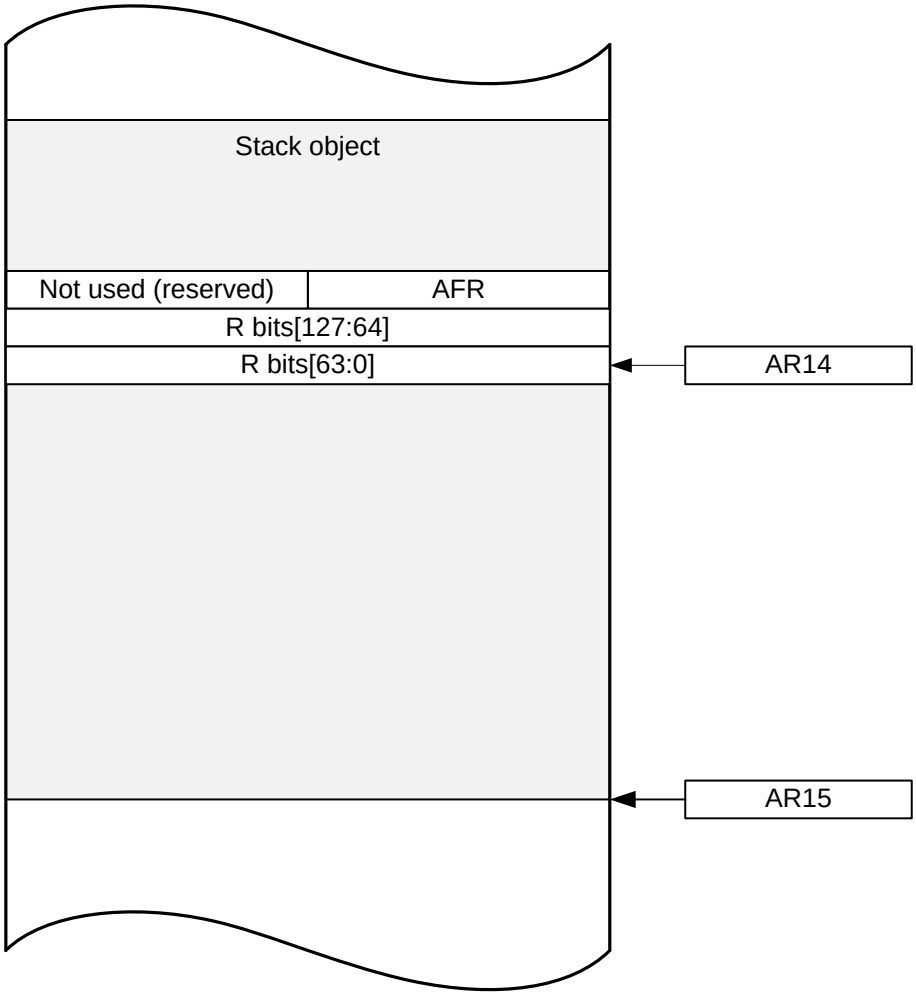
PUSD src

Format:



Description.

Push data register content into stack. 16 bytes of a 128-bit data register and 4 bytes of the corresponding AFR register are always written to the stack. Since the stack is always aligned to the border of 8 byte words, the contents of the AFR are complemented by 4 unused bytes. The format of the top of the stack after executing the PUSD instruction:



Altered flags in AFR[src]:

Any AFR bits don’t alter.

Example:

```
PUSHD        R4
```

Exceptions:

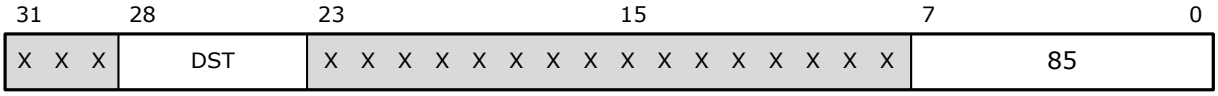
1. Stack object limits violation.

POPD – pop data register from stack

Mnemonic:

POPD dst

Format:



Description.

Reading the data register and its accompanying flag register from the stack.

Altered flags in AFR[src]:

AFR loads from the stack.

Example:

POPD R22

Exceptions:

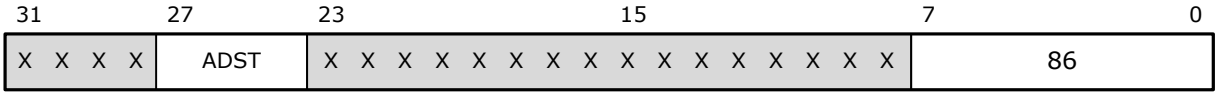
1. Stack object limits violation.

PUSHA – push address register

Mnemonic:

PUSA adst

Format:



Description.

writing the contents of the address register onto the stack.

Altered flags in AFR[src]:

Any AFR's don’t alter.

Example:

PUSHA AR4

Exceptions:

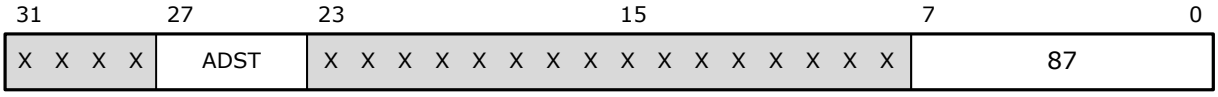
1. Stack object limits violation.

POPA – pop address register from stack

Mnemonic:

POPA adst

Format:



Description.

Reading address register from the stack.

Altered flags in AFR[src]:

Any AFR's don’t alter.

Example:

POPA AR4

Exceptions:

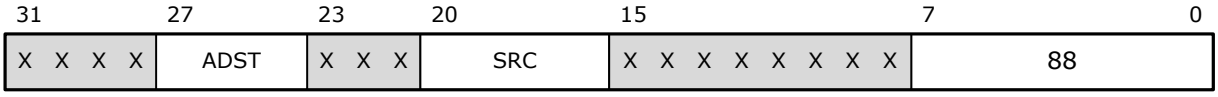
1. Stack object limits violation.

LAR – load address register

Mnemonic:

LAR adst,src

Format:



Description.

Download the address register from the general-purpose register. Registers AR13 and AR15 can only be changed with CPL = 0.

Altered flags in AFR:

Any AFR's don’t alter.

Example:

LAR AR4,R17

Exceptions:

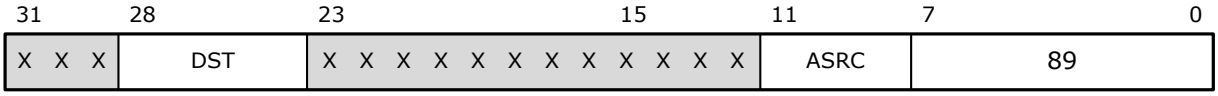
None.

SAR – store address register

Mnemonic:

SAR dst,asrc

Format:



Description.

Sending the contents of the address register to the general-purpose register.

Altered flags in AFR:

Any AFR's don’t alter.

Example:

SAR R4,AR7

Exceptions:

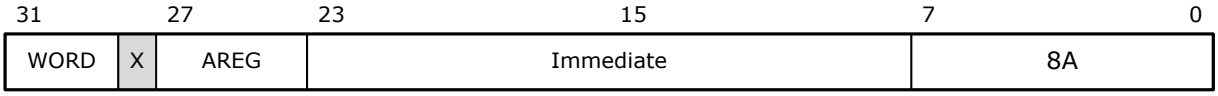
None.

LIA – load immediate offset to the address register

Mnemonic:

LIA AREG:WORD,DataImmediate

Format:



Description.

Loading a 16-bit value into the address register. Since the address registers containing the offset are 37-bit, a full load of such a register is possible with three consecutive instructions. If a word is loaded into bits [15: 0] or [31:16], then the most significant bits of the register are set according to the state of bit 15 of the word specified in the instruction.

Altered flags in AFR:

Any AFR's don’t alter.

Example:

LIA AR6:w0,Offset DataString shl 2

Exceptions:

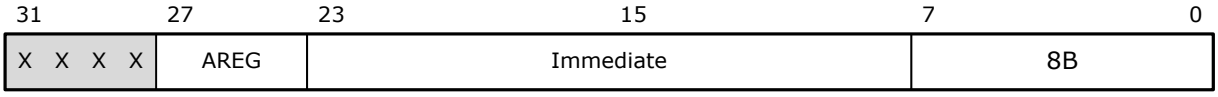
None.

IAR – increment address register

Mnemonic:

IAR AREG:DataImmediate

Format:



Description:

Increase the contents of the address register by the value specified in the instruction. The 16-bit value is supplemented with up to 37 bits with its signed bit before adding to the contents of the address register.

Altered flags in AFR:

Any AFR's don’t alter.

Example:

IAR AR6:-592

Exceptions:

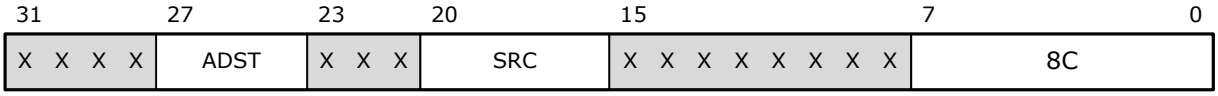
None.

AAR – add value to address register

Mnemonic:

AAR ADST:SRC

Format:



Description.

Adding to the contents of the address register the value from the general-purpose register.

Altered flags in AFR:

Any AFR's don’t alter.

Example:

AAR AR6:R3

Protection violations:

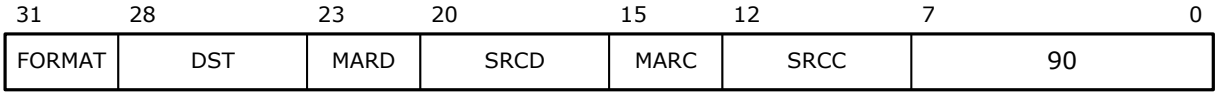
None.

FMULACC – multiplication and accumulation.

Mnemonic:

FMULACC FDST,MARC:SRCC,MARD:SRCD

Format:



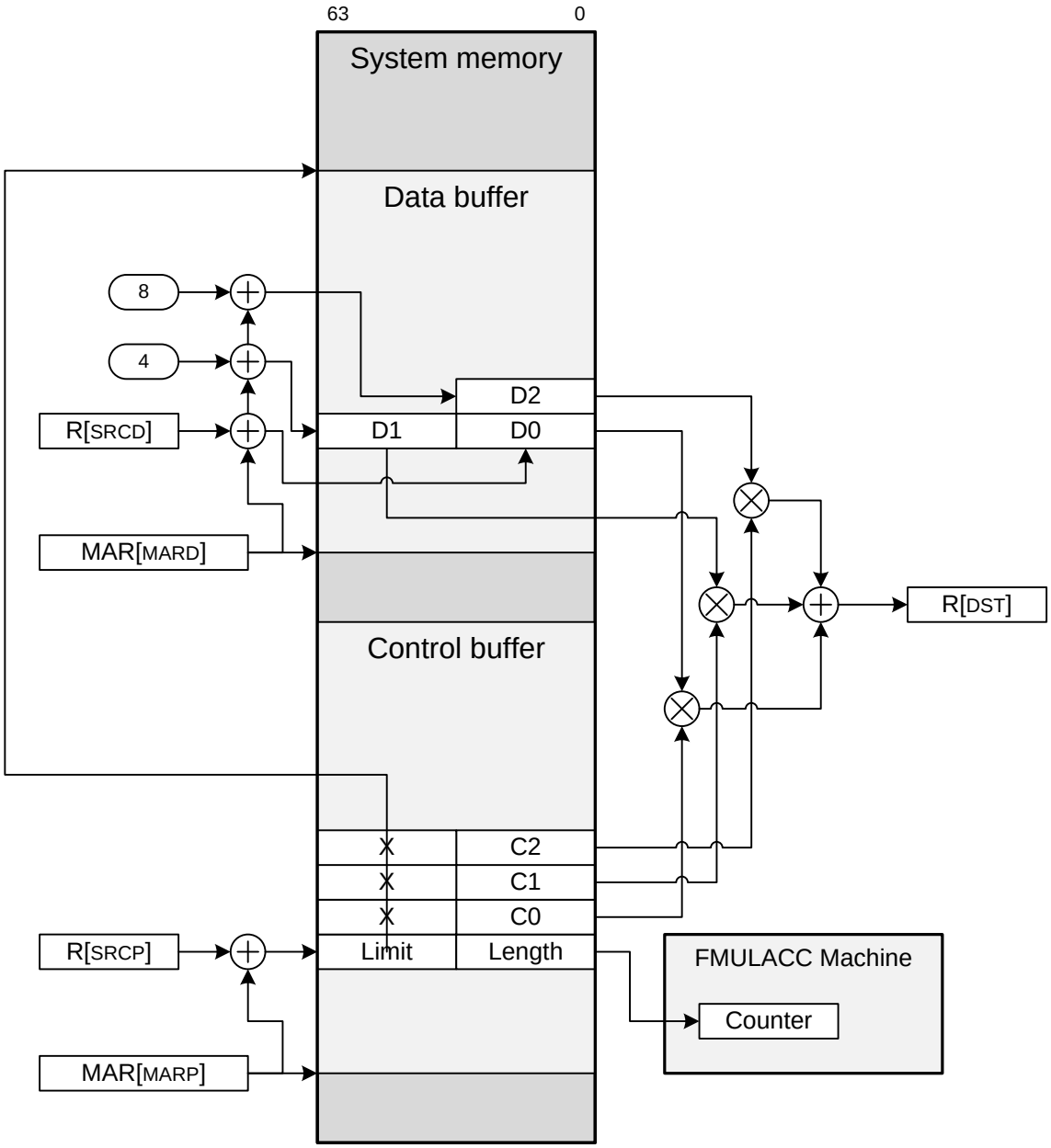
Description.

This instruction is not supported by Carrier-type cores.

Mutiplication and accumulation. The instruction performs processing of two arrays - an array of initial data and an array of coefficients. The DST register contains the sum of the results of multiplying the data elements by their corresponding coefficients. The data array contains data represented in a single-precision floating-point format. The coefficient array contains scale factors in the single-precision format and control information. The first 64-bit word in the coefficient array contains the counter of the data elements to be processed (bits [31: 0]), and the bits [63:32] can contain the length of the data buffer. The length is expressed in 32-bit words. If the bits [63:32] are zero, then the offset of each data element is set separately, in the coefficient list, next to the corresponding coefficient. If the block length is nonzero, then this indicates the sequential arrangement of the data in the array.

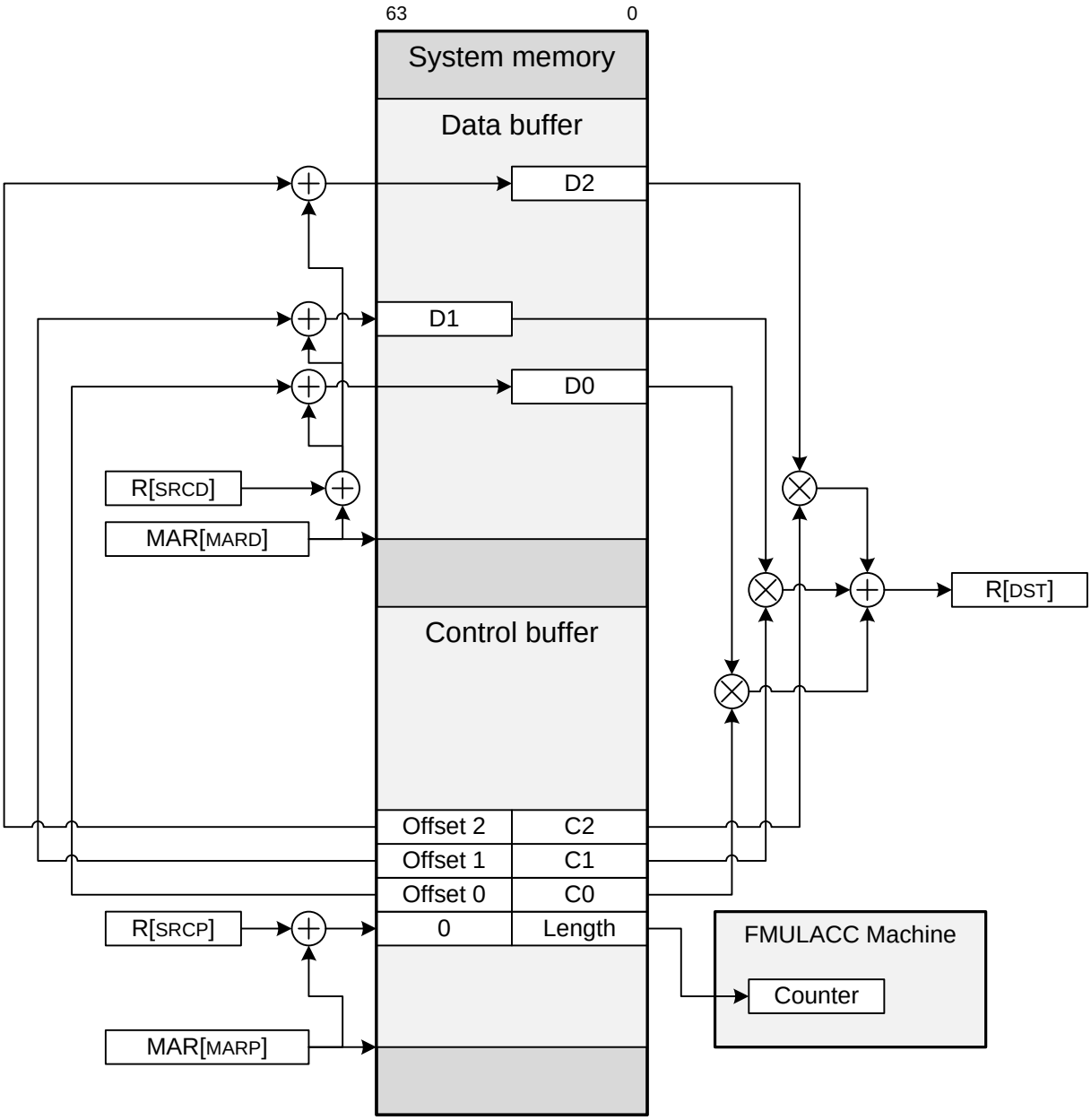
The group of address registers MARC indicates the base of the memory block where the coefficient table is placed. The SRCC general-purpose register defines an additional offset of the coefficient table in the block.

The address register group MARD points to the database of the data block. The general-purpose register SRCD determines the offset of the first data element in the data block.



The data is placed with a constant step

The sequential data addressing mode can be used to implement FIR and IIR filters. In this mode, if during data processing the data pointer reaches the limit value, then it is set to 0 and the next data element will be read from the data buffer with a zero offset from the beginning of the buffer. By changing the starting value in the R[SRCD] register, it is possible to simulate the operation of the delay line of the FIR/IIR filter with the aid of a ring buffer.



The data is placed in a predefined locations

X32Carrier core do not have a FMULACC instruction.

Altered flags in AFR[dst]:

ZF, SF, IF, NF

Example:

```
FMULACC     R4,mar1:r4,mar2:R9
```

Protection violations:

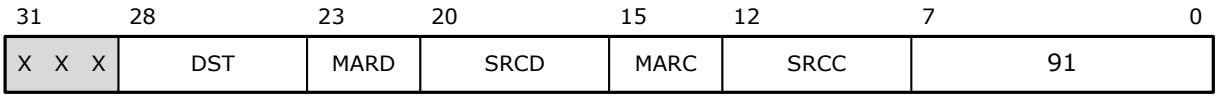
1. Violation of the object's limit.
2. Bad data selector, if the selector is zero or it exceeds the limit of the descriptor table.

3. Attempt to read from an object that is not readable.
4. Object not accessible on a current privilege level.
5. Violation of object protection mechanism by TaskID value occurs.
6. Invalid descriptor type occurs.
7. Object can’t be accessible for any other cores in the multiprocessor network.
8. The processor with the specified number is not on the network.

FFT – fast fourier transform

Mnemonic:

FFT idst,MARC:SRCC,MARD:SRCD



Description.

This instruction is not supported by Carrier-type cores.

The instruction initiates the process of calculating the fast Fourier transform. The instruction belongs to the FlyBy class of instructions and allows the processor to continue with the following instructions, without waiting for the completion of the FFT calculation. Data and twiddle factors are complex numbers, the real and imaginary parts of which are represented in floating-point format of single precision. The source data are:

- The size of the data array. The 5-bit data length code is located in bits [4: 0] of the general-purpose register, addressed by the DST field. The parameter can take values from 0 (data length - 2 numbers) and up to 19 (data length - 1048576 complex numbers).
- Pointer to a twiddle factors array. The pointer consists of an object selector, a block offset (both parameters are placed in MAR[MARC]), and an additional block offset (retrieved from the register R[SRCC]). The length of the array of twiddle factors is 2 times less than the length of the data block.
- Pointer to a data block. The pointer consists of an object selector, a block offset (both parameters are placed in MAR[MARD]), and an additional block offset (retrieved from the register R[SRCD]).

At the time of receipt of the FFT instruction, the machine may be busy processing the data array, initiated earlier. In this case, the new instruction is ignored. To control the start of the FFT calculation process, the ZF AFR [DST] flag allows. ZF [AFR [DST]] = 1 indicates that the instruction has successfully started the FFT machine. If ZF = 0, then the command must be repeated after some time.

The completion of the processing of the data array is accompanied by the recording of the code 544646464F444E45h (string "ENDOFFFT") instead of the last complex number in the data array. Periodically scanning the last 8 bytes of the data array, you can determine the completion of the calculation of the FFT.

X32Carrier core do not have a FFT instruction.

Altered flags in AFR[dst]:

ZF sets to 1 if FFT Machine starts calculations.

Example:

FFTStart:

```
FFT   R4,MAR0:R5,MAR2:R11

JNC    R4:ZF,Displacement FFTStart
```

Protection violations:

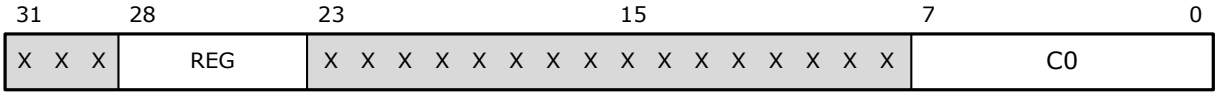
1. Violation of the object's limit.
2. Bad data selector, if the selector is zero or it exceeds the limit of the descriptor table.
3. Attempt to read from an object that is not readable.
4. Object not accessible on a current privilege level.
5. Violation of object protection mechanism by TaskID value occurs.
6. Invalid descriptor type occurs.
7. Object can’t be accessible for any other cores in the multiprocessor network.
8. The processor with the specified number is not on the network.

SENDMSG – send message

Mnemonic:

SENDMSG REG

Format:



Description.

Sending a message. The message identifier is located in the [15: 0] bits of the R[dst] general-purpose register. A 32-bit parameter that is passed to the message handler is placed in the bits [63:32] of the R[dst] register.

Altered flags:

None.

Example:

SENDMSG R9

Protection violations:

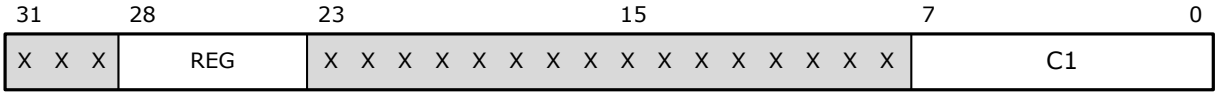
1. The message index is outside the table of imported procedures.
2. Invalid PSO selector to which the message is sent.
3. The index goes beyond the table of exported procedures.
4. Violation of access to the message handler by privilege level.
5. The type of the message handler does not match the mode of access. For example, if a software attempt is made to call a hardware interrupt handler.
6. There is no space in the message queue to write a message.

GETPAR – get message parameter

Mnemonic:

GETPAR REG

Format:



Description.

Getting the message parameter into the general-purpose register. The main process code can also get the parameter with which the process was launched.

Altered flags:

None.

Example:

GETPAR R14

Protection violations:

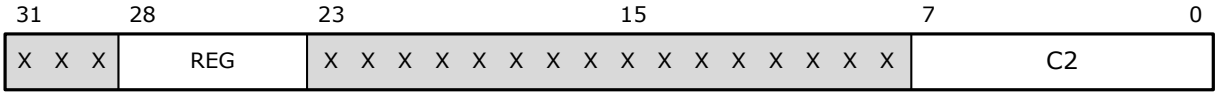
None.

JUMPR – jump by register content

Mnemonic:

JUMPR REG

Format:



Description.

Unconditional jump by the contents of the general-purpose register. The contents of the register determines the offset of the first command in a new instruction flow.

Altered flags:

None.

Example:

JUMPR R16

Protection violations:

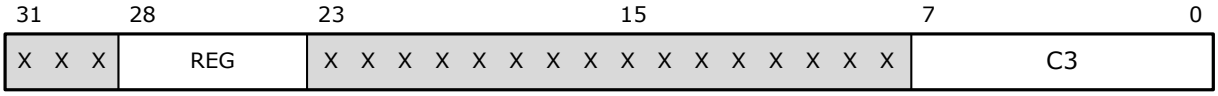
Code object limits violation.

CALLR – subroutine call by register content

Mnemonic:

CALLR REG

Format:



Description.

Unconditional call of the subroutine by the contents of the register. The contents of the register determines the offset of the first subroutine instruction in the code object.

Altered flags:

None.

Example:

CALLR R23

Protection violations:

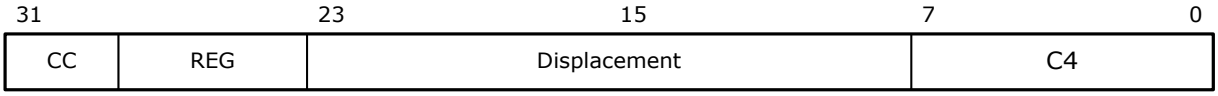
1. Code object limits violation.
2. Stack limits violation.

JC – jump conditional if flag set to 1

Mnemonic:

JC REG:CC,Displacement

Format:



Description.

Conditional jump if the selected flag is set to 1. The relative offset is supplemented by the sign bit on the left and is summed with the address of the JC instruction.

CC	***Flag***	***Description***
0	ZF	Zero flag
1	CF	Carry flag
2	SF	Sign flag
3	OF	Overflow flag /integer operations/
4	IF	Infinity flag /floating point operations/
5	NF	Not a number flag
6	DF	Data flag
7	1	Always “true” jump condition

Altered flags:

None.

Example:

JC R16:SF,Displacement Lab1

Protection violations:

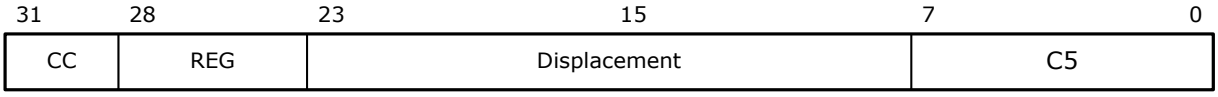
Code object limits violation.

JNC – jump conditional if flag set to 0

Mnemonic:

JNC REG:CC,Displacement

Format:



Description.

Conditional jump if flag is cleared.

Altered flags:

None.

Example:

JNC R16:SF,Displacement Lab1

Protection violations:

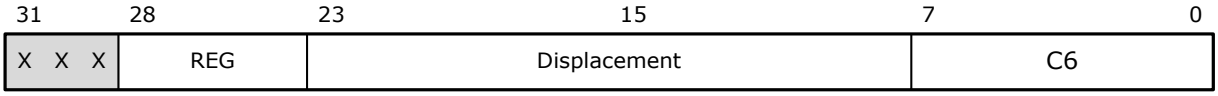
Code object limits violation.

LOOP – loop

Mnemonic:

LOOP REG,Displacement

Format:



Description.

The instruction is designed to organize the cycle. The loop counter is located in the general-purpose register. The offset from the instruction is supplemented by 19 bits on the left and is summed with the address of the LOOP instruction itself to obtain the address of the first instruction in the loop.

Altered flags:

None.

Example:

LOOP R6,Displacement Cycle0

Protection violations:

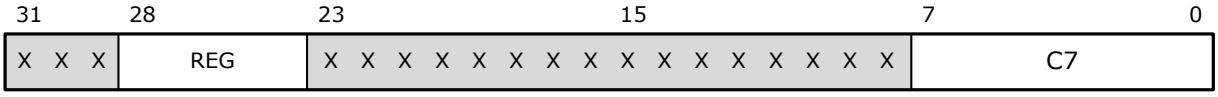
Code object limits violation.

MEMALLOC – memory allocation request

Mnemonic:

MEMALLOC REG

Format:



Description.

Request a block of memory or free a block of memory. The parameter specified in the general-purpose register determines whether a request will be made for a new block or release of the old one.

To obtain a new memory block, it is necessary to indicate the required block size in bits [63:32]. Size must be expressed in 32-byte paragraphs. Bits [31: 0] must be in the zero state for the operation of allocating a new block.

If bits [31: 0] contain a value other than 0, then the block is freed, and this value is interpreted as the selector of the object that is freed. Object can be released when owner field from object's descriptor is equal to PSO selector. If CPL=0 owner's verification skips by processor.

Altered flags:

None.

Example:

MEMALLOC R7

Protection violations:

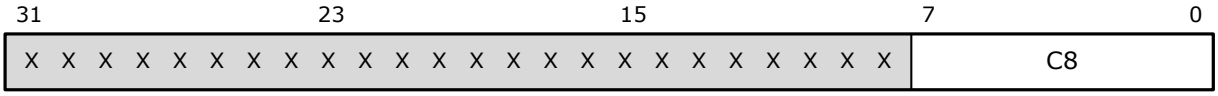
None.

RET – return from subroutine

Mnemonic:

RET

Format:



Description.

Return from the subroutine.

Altered flags:

None.

Example:

RET

Protection violations:

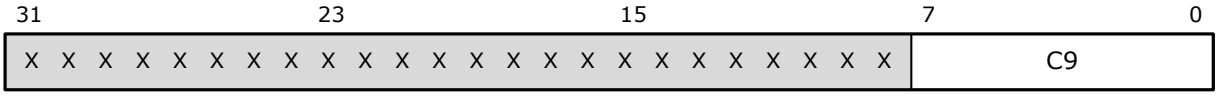
1. Code object limits violation.
2. Stack limits violation.

ENDMSG – end of message

Mnemonic:

ENDMSG

Format:



Description.

The instruction terminated the processing of the hardware interrupt or message and restores the execution of the interrupted process.

Altered flags:

None.

Example:

ENDMSG

Protection violations:

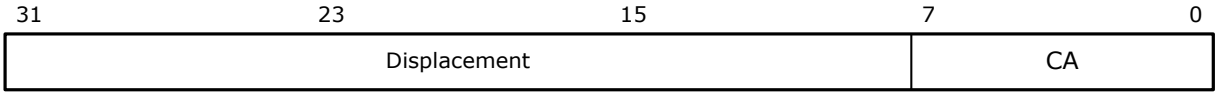
1. Empty context stack.
2. Invalid return PSO selector in context stack.

JUMPI – jump by immediate displacement

Mnemonic:

JUMPI Displacement

Format:



Description.

Unconditional jump with immediate value of relative displacement. 24 bits of the offset specified in the command are complemented by 11 bits on the left and are summed with the address of the JUMPI command itself.

Altered flags:

None.

Example:

JUMPI Displacement Label11

Protection violations:

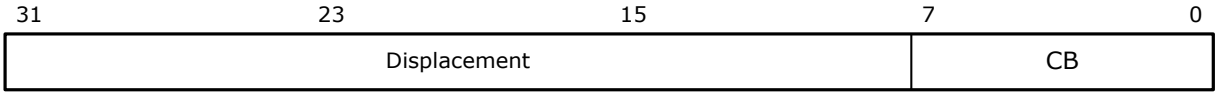
Code object limits violation.

CALLI – subroutine call by immediate displacement

Mnemonic:

CALLI Displacement

Format:



Description.

Unconditional subroutine call with immediate value of relative displacement.

Altered flags:

None.

Example:

CALLI Displacement Label11

Protection violations:

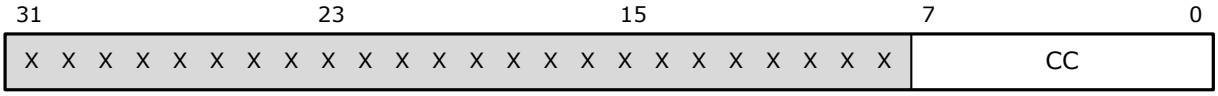
1. Code object limits violation.
2. Stack limit violation.

BKPT – breakpoint

Mnemonic:

BKPT

Format:



Description.

Breakpoint generation.

Altered flags:

None.

Example:

BKPT

Protection violations:

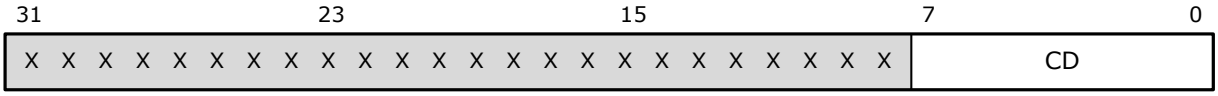
None.

SLEEP – process sleeps

Mnemonic:

SLEEP

Format:



Description.

A process frees the core to execute other processes before the activity timer for the current process located in the PTR register ends. Instruction SLEEP resets PTR counter to value 2 what causes switch of processes in short time.

Altered flags:

None.

Example:

SLEEP

Protection violations:

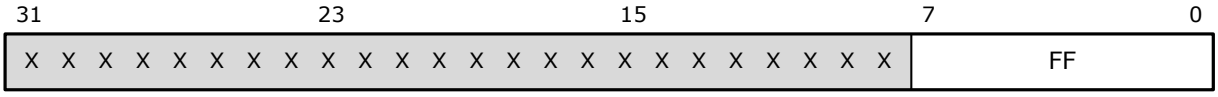
None.

NOP – no operations

Mnemonic:

NOP

Format:



Description.

Do nothing instruction.

Altered flags:

None.

Example:

NOP

Protection violations:

None.