Contents

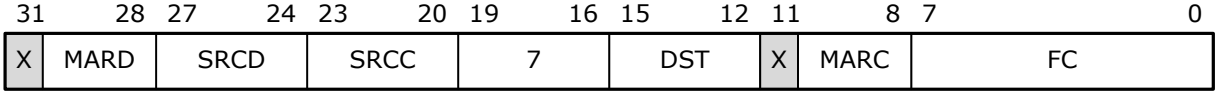
Instruction set reference

FFT – Fast Fourier Transform

Mnemonic:

FFT dst,marC:srcC,marD:srcD

Format:



Group: 10

Description:

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.

Altered flags in AFR[dst]:

CF[15:0]	Stay unchanged.
ZF	Set to 1 if FFT machine runs calculation.
SF	Stay unchanged.
OF	Stay unchanged.
IF	Stay unchanged.
NF	Stay unchanged.
DBF	Stay unchanged.
OpSize	Stay unchanged.
LICntr	Set to zero for AFR[dst], AFR[srcP] and AFR[srcD].
AMode	Stay unchanged.

Protection violations:

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

Example:

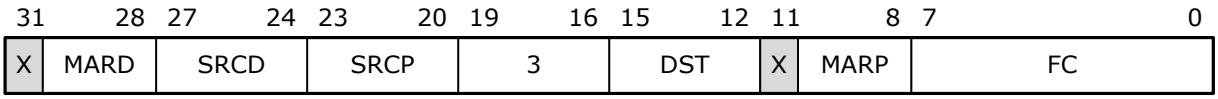
```
FFTStart:
        FFT   R4,MAR0:R5,MAR2:R11
        JC    R4:NZF,Displacement FFTStart
```

FMULACC – Floating point multiplication and accumulation

Mnemonic:

```
FMULACC dst,marP:srcP,marD:srcD
```

Format:



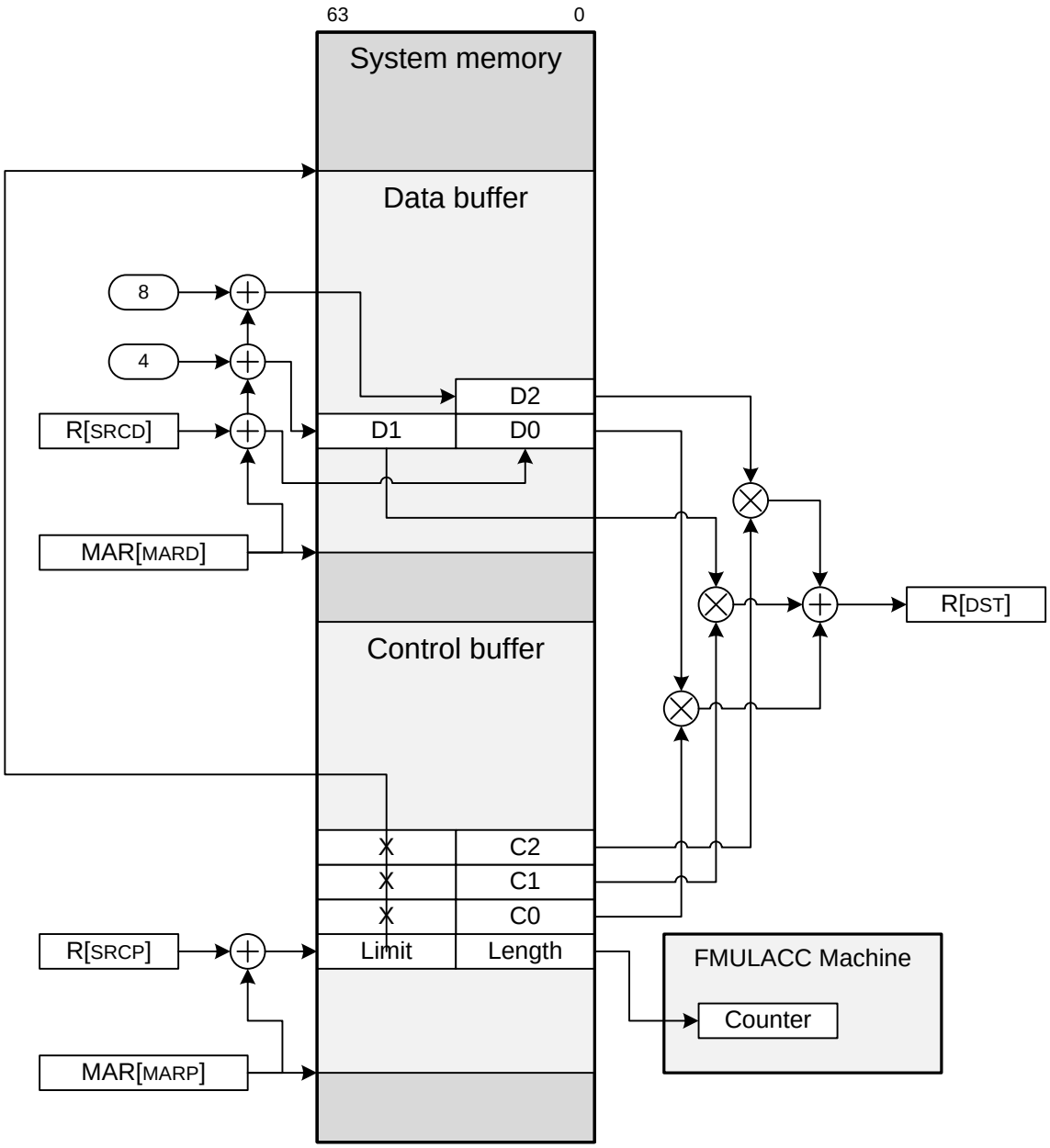
Group: 10

Description:

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.

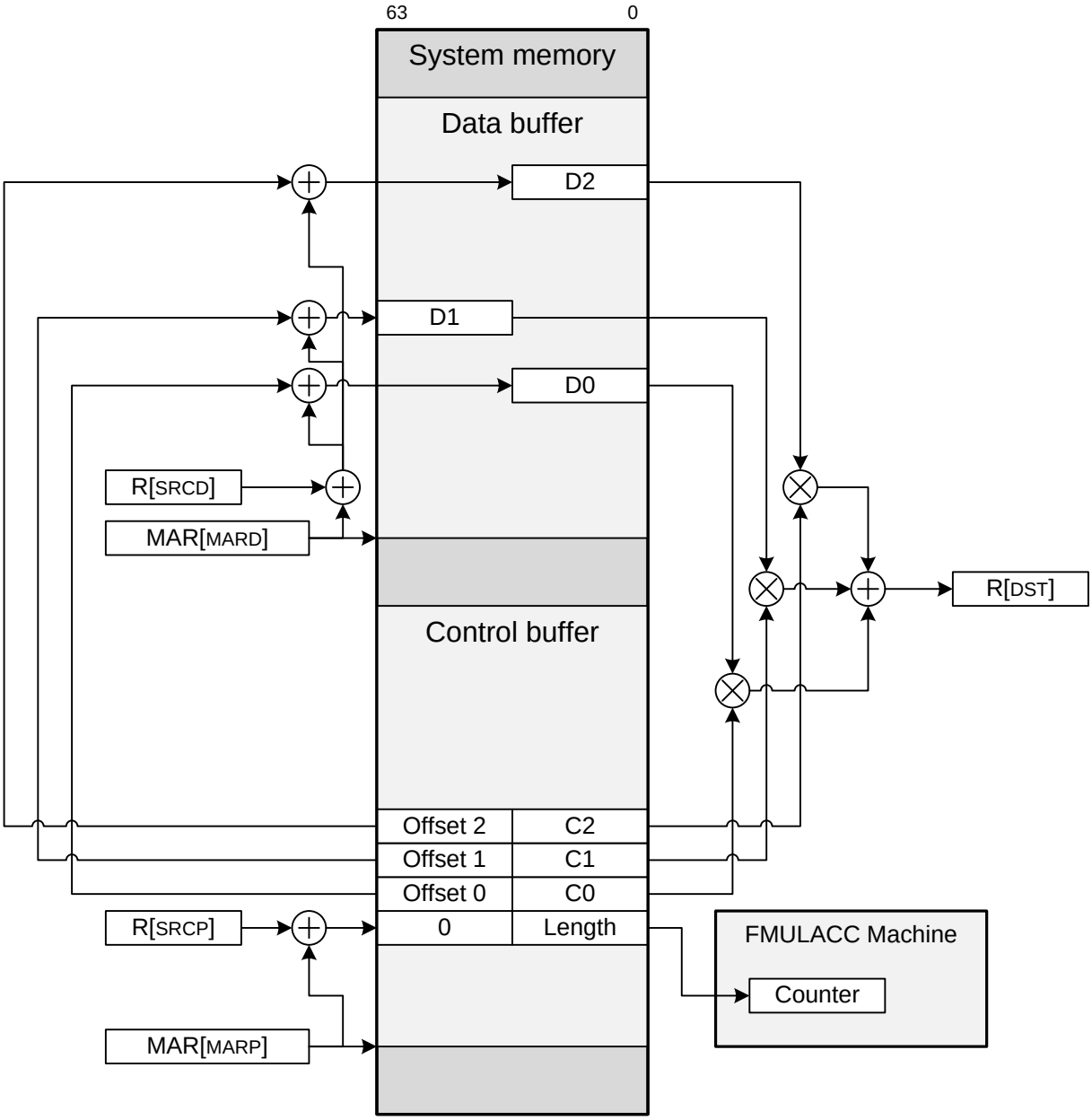
The group of address registers MARP indicates the base of the memory block where the coefficient table is placed. The SRCP general register defines an additional offset of the coefficient table in the block.

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

Altered flags in AFR[dst]:

CF[15:0]	Undefined state.
ZF	Set to 1 if zero result.
SF	MSB of the result.
OF	Undefined state.
IF	Set to 1 if the result is out of range of the number representation for the single precision format.
NF	Set to 1 if one of the operands is a non-a-number.
DBF	Set to an undefined state.

OpSize	Set to 2 (Dword).
LICntr	Set to zero for AFR[dst], AFR[srcP] and AFR[srcD].
AMode	Stay unchanged.

Protection violations:

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

Example:

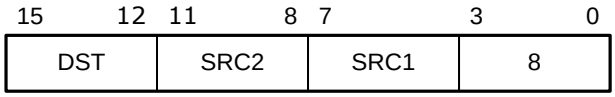
```
FMULACC   R4,MAR0:R5,MAR2:R11
```

FADD – Floating point Addition

Mnemonic:

```
FADD dst,src1:src2
```

Format:



Group: 1

Description:

Addition of numbers in a floating-point format.
R[dst]<=R[src1]+R[src2].

Altered flags in AFR[dst]:

CF[15:0]	Undefined state.
ZF	Set to 1 if zero result.
SF	MSB of the result.
OF	Undefined state.
IF	Set to 1 if the result is out of range of the number representation for the selected destination format.
NF	Set to 1 if one of the operands is a non-a-number.
DBF	Set to an undefined state.
OpSize	Determined by the maximum value of the width of the initial operands.
LICntr	Set to zero for AFR[dst], AFR[src1] and AFR[src2].
AMode	Stay unchanged.

Protection violations:

None.

Example:

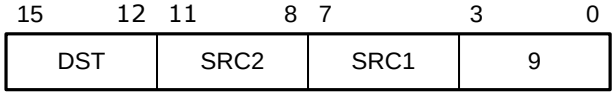
```
FADD  R3,R7:R15
```

FSUB – Floating point subtraction

Mnemonic:

FSUB dst,src1:src2

Format:



Group: 1

Description:

Subtraction of numbers in a floating-point format.

R[dst]<=R[src1]-R[src2].

Altered flags in AFR[dst]:

CF[15:0]	Undefined state.
ZF	Set to 1 if zero result.
SF	MSB of the result.
OF	Undefined state.
IF	Set to 1 if the result is out of range of the number representation for the selected destination format.
NF	Set to 1 if one of the operands is a non-a-number.
DBF	Undefined state.
OpSize	Determined by the maximum value of the width of the initial operands.
LICntr	Set to zero for AFR[dst], AFR[src1] and AFR[src2].
AMode	Stay unchanged.

Protection violations:

None.

Example:

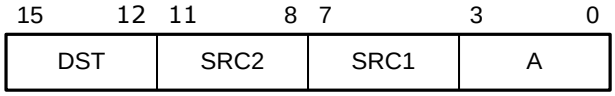
FSUB R13,R6:R7

FMUL – Floating point multiplication

Mnemonic:

FMUL dst,src1:src2

Format:



Group: 2

Description:

Multiplication of numbers in a floating-point format.

R[dst]<=R[src1]*R[src2].

Altered flags in AFR[dst]:

CF[15:0]	Undefined state.
ZF	Set to 1 if zero result.
SF	MSB of the result.
OF	Undefined state.
IF	Set to 1 if the result is out of range of the number representation for the selected destination format.
NF	Set to 1 if one of the operands is a non-a-number.
DBF	Undefined state.
OpSize	Determined by the maximum value of the width of the initial operands.
LICntr	Set to zero for AFR[dst], AFR[src1] and AFR[src2].
AMode	Stay unchanged.

Protection violations:

None.

Example:

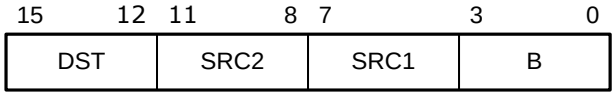
FMUL R10,R8:R8

FDIV – Floating point division

Mnemonic:

FDIV dst,src1:src2

Format:



Group: 3

Description:

Division of numbers in a floating-point format.

R[dst]<=R[src1]/R[src2].

Altered flags in AFR[dst]:

CF[15:0]	Undefined state.
ZF	Set to 1 if zero result.
SF	MSB of the result.
OF	Set to an undefined state.
IF	Set to 1 if the result is out of range of the number representation for the selected destination format or if divisor was zero.
NF	Set to 1 if one of the operands is a non-a-number.
DBF	Set to undefined state.
OpSize	Determined by the maximum value of the width of the initial operands.
LICntr	Set to zero for AFR[dst], AFR[src1] and AFR[src2].
AMode	Stay unchanged.

Protection violations:

None.

Example:

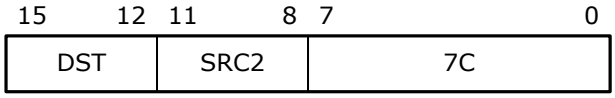
FDIV R0,R5:r10

SQRT – Square root

Mnemonic:

SQRT dst,src2

Format:



Group: 3

Description:

Extracting the square root from the contents of R[src2] and writing the result to the R[dst] register. R[dst]<=sqrt(R[src2]).

Altered flags in AFR[dst]:

CF[15:0]	Undefined state.
ZF	Set to 1 if zero result.
SF	Set to 0.
OF	Set to undefined state.
IF	Set to 1 if the initial operand was an infinite number and to 0, if not.
NF	Set to 1 if initial operand is a non-a-number.
DBF	Set to undefined state.
OpSize	Determined by value of the width of the initial operand.

LICntr	Set to zero for AFR[dst] and AFR[src2].
AMode	Stay unchanged.

Protection violations:

None.

Example:

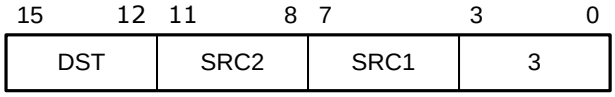
```
SQRT R2,R10
```

ADD – Integer addition

Mnemonic:

ADD dst,src1:src2

Format:



Group: 4

Description:

Addition of integer numbers.

R[dst]<=R[src1]+R[src2].

Altered flags in AFR[dst]:

CF[15:0]	Register carry signals from tetrads.
ZF	Set to 1 if zero result.
SF	MSB of the result.
OF	Set to 1 if integer overflow occurs.
IF	Set to undefined state.
NF	Set to undefined state.
DBF	Set to undefined state.
OpSize	Determined by the maximum value of the width of the initial operands.
LICntr	Set to zero for AFR[dst], AFR[src1] and AFR[src2].
AMode	Stay unchanged.

Protection violations:

None.

Example:

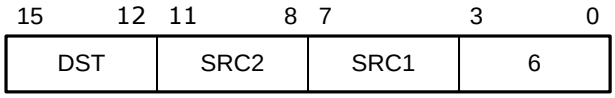
ADD R6,R6:R10

AND – Logical AND

Mnemonic:

AND dst,src1:src2

Format:



Group: 4

Description:

Logical AND.

R[dst]<=R[src1] &R[src2].

Altered flags in AFR[dst]:

CF[15:0]	Undefined state.
ZF	Set to 1 if zero result.
SF	MSB of the result.
OF	Set to 0.
IF	Set to undefined state.
NF	Set to undefined state.
DBF	Set to undefined state.
OpSize	Determined by the maximum value of the width of the initial operands.
LICntr	Set to zero for AFR[dst], AFR[src1] and AFR[src2].
AMode	Stay unchanged.

Protection violations:

None.

Example:

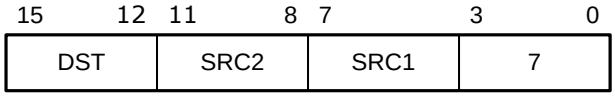
```
AND R15,R2:R3
```

OR – Logical OR

Mnemonic:

OR dst,src1:src2

Format:



Group: 4

Description:

Logical OR.

R[dst]<=R[src1] | R[src2].

Altered flags in AFR[dst]:

CF[15:0]	Set to undefined state.
ZF	Set to 1 if zero result.
SF	MSB of the result.
OF	Set to 0.
IF	Set to undefined state.
NF	Set to undefined state.
DBF	Set to undefined state.
OpSize	Determined by the maximum value of the width of the initial operands.
LICntr	Set to zero for AFR[dst], AFR[src1] and AFR[src2].
AMode	Stay unchanged.

Protection violations:

None.

Example:

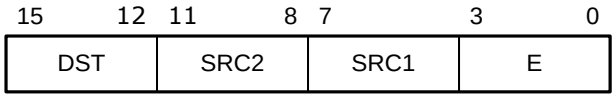
OR R9,R9:R0

XOR – Exclusive OR

Mnemonic:

XOR dst,src1:src2

Format:



Group: 4

Description:

Exclusive OR.

R[dst]<=R[src1] ^ R[src2].

Altered flags in AFR[dst]:

CF[15:0]	Set to undefined state.
ZF	Set to 1 if zero result.
SF	MSB of the result.
OF	Set to 0.
IF	Set to undefined state.
NF	Set to undefined state.
DBF	Set to undefined state.
OpSize	Determined by the maximum value of the width of the initial operands.
LICntr	Set to zero for AFR[dst], AFR[src1] and AFR[src2].
AMode	Stay unchanged.

Protection violations:

None.

Example:

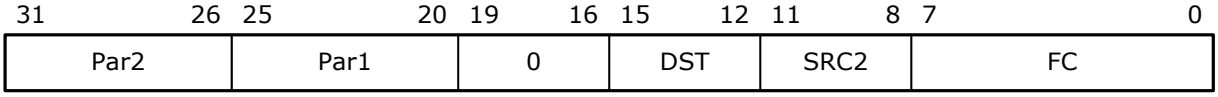
XOR R2,R13:R11

FIELDCOPYI – Copying bit field

Mnemonic:

FIELDCOPYI dst,src2:par1:par2

Format:



Group: 4

Description:

Copying the bit field. Several bits (par2), starting with the zero, extracted from the src2 register, are copied to the dst register. Par1 determines the position of the first bit in the dst register where the copied bit group will be placed. The remaining bits of the dst register remain unchanged. For example, if dst register R5 holds 16-bit value 0F007h and a value in src2 register R7 is byte 55h. After the fieldcopyi r5,r7:5:7 instruction is executed, the R5 register will contain a 16-bit value of 0FAA7h.

R[dst]<={R[dst][63:par1+par2], R[src2][par2-1:0], R[dst][par1-1:0]}

Altered flags in AFR[dst]:

CF[15:0]	Set to undefined state.
ZF	Set to undefined state.
SF	Set to undefined state.
OF	Set to undefined state.
IF	Set to undefined state.
NF	Set to undefined state.
DBF	Set to undefined state.
OpSize	Determined by the OpSize AFR[dst].
LICntr	Set to zero for AFR[dst] and AFR[src2].
AMode	Stay unchanged.

Protection violations:

None.

Example:

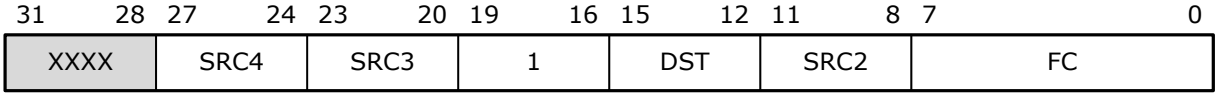
FIELDCOPYI R5,R7:30:7

FIELDCOPY – Copying bit field

Mnemonic:

FIELDCOPY dst,src2:src3:src4

Format:



Group: 4

Description:

Copying the bit field. Several bits, starting with the zero bit, extracted from the src2 register, are copied to the dst register. The register R[src3] contains a 6-bit value that determines the number of the first bit of the register R[dst], from which the copied group of bits will be located. The bits [5:0] of the register R[src4] contain the number of bits to be copied.

R[dst]<={R[dst][63:R[src3][5:0]+R[src4][5:0]], R[src2][R[src4][5:0]-1:0], R[dst][R[src3][5:0]-1:0]}

Altered flags in AFR[dst]:

CF[15:0]	Set to undefined state.
ZF	Set to undefined state.
SF	Set to undefined state.
OF	Set to undefined state.
IF	Set to undefined state.
NF	Set to undefined state.
DBF	Set to undefined state.
OpSize	Determined by the OpSize AFR[dst].
LICntr	Set to zero for AFR[dst], AFR[src3], AFR[src4] and AFR[src2].
AMode	Stay unchanged.

Protection violations:

None.

Example:

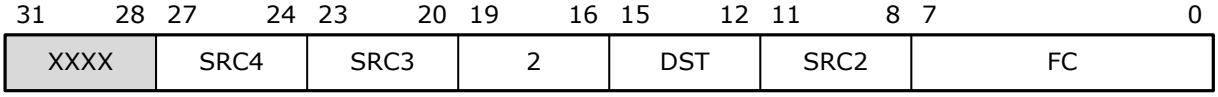
FIELDCOPY R4,R5:R10:R0

MASKCOPY – Masked bits copying

Mnemonic:

MASKCOPY dst,src2:src3:src4

Format:



Group: 4

Description:

Masked bit copying.

R[dst]<=(R[src2] & ~R[src4]) | (R[src3] & R[src4])

Altered flags in AFR[dst]:

CF[15:0]	Set to undefined state.
ZF	Set to undefined state.
SF	Set to undefined state.
OF	Set to undefined state.
IF	Set to undefined state.
NF	Set to undefined state.
DBF	Set to undefined state.
OpSize	Determined by maximum value from AFR[src2], AFR[src3] and AFR[src4].
LICntr	Set to zero for AFR[dst], AFR[src3], AFR[src4] and AFR[src2].
AMode	Stay unchanged.

Protection violations:

None.

Example:

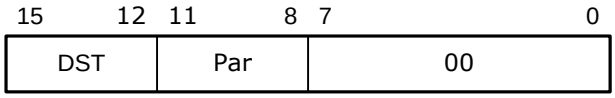
MASKCOPY R1,R9:R10:R15

LSLI – Logical shift left on immediately specified number of bits

Mnemonic:

LSLI dst:Par

Format:



Group: 5

Description:

Shift the contents of the DST register to the left via the DBF flag by the number of bits specified by the parameter Par. After executing the command, the last extended bit is stored in the DBF. The shift is possible by a maximum of 15 bits.

R[dst]<=R[dst] << Par

Altered flags in AFR[dst]:

CF[15:0]	Set to undefined state.
ZF	Set to 1 if result is zero.
SF	MSB of the result.
OF	Set to 1 if arithmetic shift left overflow.
IF	Set to undefined state.
NF	Set to undefined state.
DBF	Set to the last shifted out data bit.
OpSize	Determined by the OpSize AFR[dst].
LICntr	Set to zero for AFR[dst].
AMode	Stay unchanged.

Protection violations:

None.

Example:

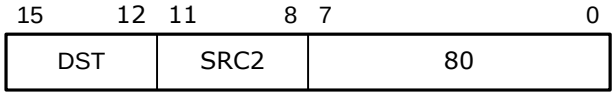
LSLI R0:9

LSL – Logical shift left

Mnemonic:

LSL dst:src2

Format:



Group: 5

Description:

Shifts the contents of the R[dst] register to the left via the DBF flag by the number of bits specified in the R[src2] register. The bit [5:0] of the R[src2] register is used as the shift parameter.

R[dst]<=R[dst] << R[src2][5:0]

Altered flags in AFR[dst]:

CF[15:0]	Set to undefined state.
ZF	Set to 1 if result is zero.
SF	MSB of the result.
OF	Set to 1 if arithmetic shift left overflow.
IF	Set to undefined state.
NF	Set to undefined state.
DBF	Set to the last shifted out data bit.
OpSize	Determined by the OpSize AFR[dst].
LICntr	Set to zero for AFR[dst] and AFR[src2].
AMode	Stay unchanged.

Protection violations:

None.

Example:

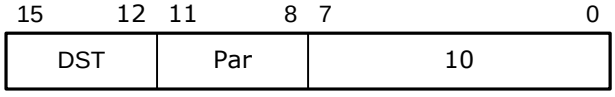
LSL R5:R8

LSRI – Logical shift right on immediately specified number of bits

Mnemonic:

LSRI dst:Par

Format:



Group: 5

Description:

Shift the contents of the DST register to the right via the DBF flag by the number of bits specified by the parameter Par. After executing the command, the last extended bit is stored in the DBF. The shift is possible by a maximum of 15 bits.

R[dst]<=R[dst] >> Par

Altered flags in AFR[dst]:

CF[15:0]	Set to undefined state.
ZF	Set to 1 if result is zero.
SF	MSB of the result.
OF	Set to 0.
IF	Set to undefined state.

NF	Set to undefined state.
DBF	Set to the last shifted out data bit.
OpSize	Determined by the OpSize AFR[dst].
LICntr	Set to zero for AFR[dst].
AMode	Stay unchanged.

Protection violations:

None.

Example:

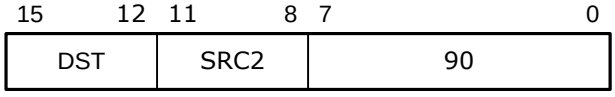
LSRI R10:8

LSR – Logical shift right

Mnemonic:

LSR dst:src2

Format:



Group: 5

Description:

Shifts the contents of the R[dst] register to the right via the DBF flag by the number of bits specified in the R[src2] register. The bit [5:0] of the R[src2] register is used as the shift parameter.

R[dst]<=R[dst] >> R[src2][5:0]

Altered flags in AFR[dst]:

CF[15:0]	Set to undefined state.
ZF	Set to 1 if result is zero.
SF	MSB of the result.
OF	Set to 0.
IF	Set to undefined state.
NF	Set to undefined state.
DBF	Set to the last shifted out data bit.
OpSize	Determined by the OpSize AFR[dst].
LICntr	Set to zero for AFR[dst] and AFR[src2].
AMode	Stay unchanged.

Protection violations:

None.

Example:

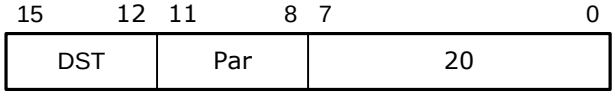
LSR R6:R0

CSLI – Cyclic shift left on immediately specified number of bits

Mnemonic:

CSLI dst:Par

Format:



Group: 5

Description:

Cyclic left shift of the contents of the register R[dst] by the number of bits determined by the directly specified Par parameter.

R[dst]<=R[dst] << Par

Altered flags in AFR[dst]:

CF[15:0]	Set to undefined state.
ZF	Set to 1 if result is zero.
SF	MSB of the result.
OF	Set to 0.
IF	Set to undefined state.
NF	Set to undefined state.
DBF	Set to the last shifted out data bit.
OpSize	Determined by the OpSize AFR[dst].
LICntr	Set to zero for AFR[dst].
AMode	Stay unchanged.

Protection violations:

None.

Example:

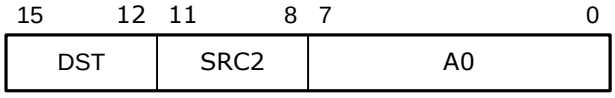
CSLI R10:10

CSL – Cyclic shift left

Mnemonic:

CSL dst:src2

Format:



Group: 5

Description:

Cyclic shift left of the contents of the register R[dst] by the number of bits specified in the bits [5: 0] of the register R[src2].

R[dst]<=R[dst] << R[src2]

Altered flags in AFR[dst]:

CF[15:0]	Set to undefined state.
ZF	Set to 1 if result is zero.
SF	MSB of the result.
OF	Set to 0.
IF	Set to undefined state.
NF	Set to undefined state.
DBF	Set to the last shifted out data bit.
OpSize	Determined by the OpSize AFR[dst].
LICntr	Set to zero for AFR[dst] and AFR[src2].
AMode	Stay unchanged.

Protection violations:

None.

Example:

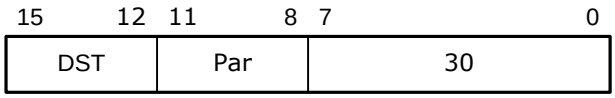
CSL R5:R11

CSRI – Cyclic shift right by immediately specified number of bits

Mnemonic:

CSRI dst:Par

Format:



Group: 5

Description:

Cyclic right shift of the contents of register R [dst] to the specified number of bits.
R[dst]<={R[dst][Par-1:0], R[dst][MSB:Par]}

Altered flags in AFR[dst]:

CF[15:0]	Set to undefined state.
ZF	Set to 1 if result is zero.
SF	MSB of the result.
OF	Set to 0.
IF	Set to undefined state.
NF	Set to undefined state.
DBF	Set to the last shifted out data bit.
OpSize	Determined by the OpSize AFR[dst].
LICntr	Set to zero for AFR[dst].
AMode	Stay unchanged.

Protection violations:

None.

Example:

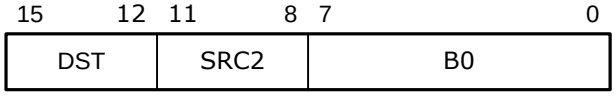
CSRI R0:10

CSR – Cyclic shift right

Mnemonic:

CSR dst:src2

Format:



Group: 5

Description:

Cyclic right shift of the contents of R[dst] by the number of bits specified in the register R[src2].

R[dst]<={R[dst][R[src2][5:0]-1:0], R[dst][MSB:R[src2][5:0]]}

Altered flags in AFR[dst]:

CF[15:0]	Set to undefined state.
ZF	Set to 1 if result is zero.
SF	MSB of the result.
OF	Set to 0.
IF	Set to undefined state.
NF	Set to undefined state.
DBF	Set to the last shifted out data bit.
OpSize	Determined by the OpSize AFR[dst].
LICntr	Set to zero for AFR[dst] and AFR[src2].
AMode	Stay unchanged.

Protection violations:

None.

Example:

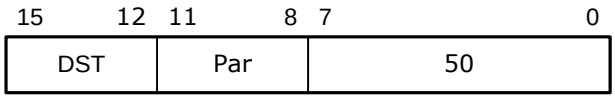
CSR R15:R0

ASRI – Arithmetic shift right by immediately specified number of bits

Mnemonic:

ASRI dst:Par

Format:



Group: 5

Description:

Arithmetic right shift of the contents of R[dst] by the number of bits determined by the specified parameter.

R[dst]<={{Par{R[dst][MSB]}}, R[dst][MSB:Par]}

Altered flags in AFR[dst]:

CF[15:0]	Set to undefined state.
ZF	Set to 1 if result is zero.
SF	MSB of the result.
OF	Set to 0.
IF	Set to undefined state.
NF	Set to undefined state.
DBF	Set to the last shifted out data bit.
OpSize	Determined by the OpSize AFR[dst].
LICntr	Set to zero for AFR[dst].

AMode Stay unchanged.

Protection violations:

None.

Example:

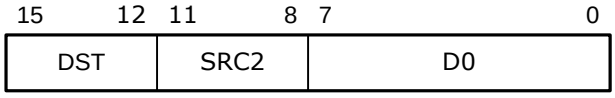
ASRI R2:5

ASR – Arithmetic shift right

Mnemonic:

ASR dst:src2

Format:



Group: 5

Description:

Arithmetic shift right of the contents of the register R[dst] by the number of bits determined by the content of the register R[src2].

R[dst]<={{R[src2][5:0]{R[dst][MSB]}}, R[dst][MSB:R[src2][5:0]]}

Altered flags in AFR[dst]:

CF[15:0] Set to undefined state.
ZF Set to 1 if result is zero.
SF MSB of the result.
OF Set to 0.
IF Set to undefined state.
NF Set to undefined state.
DBF Set to the last shifted out data bit.
OpSize Determined by the OpSize AFR[dst].
LICntr Set to zero for AFR[dst] and AFR[src2].
AMode Stay unchanged.

Protection violations:

None.

Example:

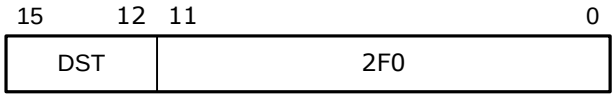
ASR R6:R7

BSWAP – Bitwise swap

Mnemonic:

BSWAP dst

Format:



Group: 6

Description:

Swap bits in operand.

If the operand is byte, then bits 7 and 0, 6 and 1 and so on change in places. For 16-bit operands, the bits 15<>0, 14<>1, etc. are being interchanged, 32-bit operands are swapped bit 31<>0, 30<>1, etc. For 64-bit operands, the bit 63<>0, 62<>1, etc. is swapped.

Altered flags in AFR[dst]:

CF[15:0] Set to undefined state.
ZF Set to 1 if result is zero.

SF	MSB of the result.
OF	Set to 0.
IF	Set to undefined state.
NF	Set to undefined state.
DBF	Set to undefined state.
OpSize	Determined by the OpSize AFR[dst].
LICntr	Set to zero for AFR[dst].
AMode	Stay unchanged.

Protection violations:

None.

Example:

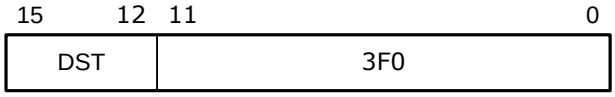
BSWAP R9

NEG – Negation

Mnemonic:

NEG dst

Format:



Group: 6

Description:

Generate additional number code. Used in the operation of subtracting binary numbers. Prepares the operand before the addition operation.

R[dst]<=(~R[dst])+1

Altered flags in AFR[dst]:

CF[15:0]	Set to undefined state.
ZF	Set to 1 if result is zero.
SF	MSB of the result.
OF	Set to 0.
IF	Set to undefined state.
NF	Set to undefined state.
DBF	Set to undefined state.
OpSize	Determined by the OpSize AFR[dst].
LICntr	Set to zero for AFR[dst].
AMode	Stay unchanged.

Protection violations:

None.

Example:

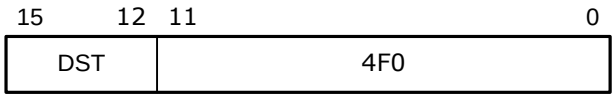
NEG R0

DAA – Decimal adjust after addition

Mnemonic:

DAA dst

Format:



Group: 6

Description:

Decimal correction of the result of the addition operation.

Altered flags in AFR[dst]:

CF[15:0]	Set to undefined state.
ZF	Set to 1 if result is zero.
SF	MSB of the result.
OF	Set to 0.
IF	Set to undefined state.
NF	Set to undefined state.
DBF	Set to undefined state.
OpSize	Determined by the OpSize AFR[dst].
LICntr	Set to zero for AFR[dst].
AMode	Stay unchanged.

Protection violations:

None.

Example:

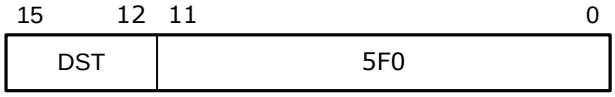
```
DAA R2
```

DAS – Decimal adjust after subtraction

Mnemonic:

DAS dst

Format:



Group: 6

Description:

Decimal correction of the result of the operation of subtraction of BCD-numbers.

Altered flags in AFR[dst]:

CF[15:0]	Set to undefined state.
ZF	Set to 1 if result is zero.
SF	MSB of the result.
OF	Set to 0.
IF	Set to undefined state.
NF	Set to undefined state.
DBF	Set to undefined state.
OpSize	Determined by the OpSize AFR[dst].
LICntr	Set to zero for AFR[dst].
AMode	Stay unchanged.

Protection violations:

None.

Example:

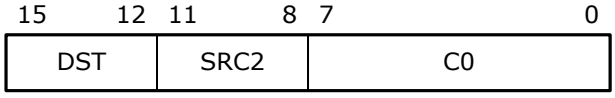
DAS R11

POS – Highest bit position calculation

Mnemonic:

POS dst,src2

Format:



Group: 6

Description:

Calculate the position of the last bit on the left side, set to 1.

Pos=0

For (i=0; i<OperandSize; i=i+1) If (R[src2][i] != 0) Pos=i

Altered flags in AFR[dst]:

CF[15:0]	Set to undefined state.
ZF	Set to 1 if result is zero.
SF	Set to 0.
OF	Set to 0.
IF	Set to undefined state.
NF	Set to undefined state.
DBF	Set to undefined state.
OpSize	Determined by the OpSize AFR[dst].
LICntr	Set to zero for AFR[dst] and AFR[src2].
AMode	Stay unchanged.

Protection violations:

None.

Example:

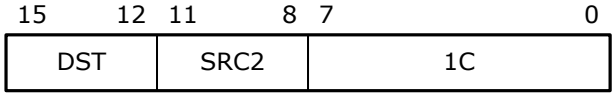
POS R0,R14

CFZ – Change data Format with rounding to Zero

Mnemonic:

CFZ dst,src2

Format:



Group: 6

Description:

Change the format of floating-point data. The number from the SRC2 register is copied to the DST register with the format change. The new number format is determined by the OpSize field of the AFR[dst]. When converting numbers to a format with less accuracy, rounding is used by discarding the lower bits of the mantissa.

Altered flags in AFR[dst]:

CF[15:0]	Set to undefined state.
ZF	Set to 1 if result is zero.
SF	Set to MSB of the result.
OF	Set to 0.
IF	Set to 1 if the number is outside the range of the number representation for the selected format.
NF	Set to 1 if source operand is a NaN.
DBF	Set to undefined state.
OpSize	Determined by the OpSize AFR[dst]
LICntr	Set to zero for AFR[dst] and AFR[src2].
AMode	Stay unchanged.

Protection violations:

None.

Example:

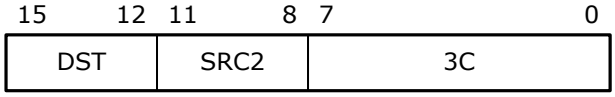
CFZ R1,R6

CFN – Change data Format with rounding to Nearest

Mnemonic:

CFN dst,src2

Format:



Group: 6

Description:

Change the format of floating-point data. The number from the SRC2 register is copied to the DST with the format change. When converting numbers to a lower-precision format, rounding to the nearest value is applied.

Altered flags in AFR[dst]:

CF[15:0]	Set to undefined state.
ZF	Set to 1 if result is zero.
SF	Set to MSB of the result.
OF	Set to 0.
IF	Set to 1 if the number is outside the range of the number representation for the selected format.
NF	Set to 1 if source operand is a NaN.
DBF	Set to undefined state.
OpSize	Determined by the OpSize AFR[dst]

LICntr Set to zero for AFR[dst] and AFR[src2].
AMode Stay unchanged.

Protection violations:

None.

Example:

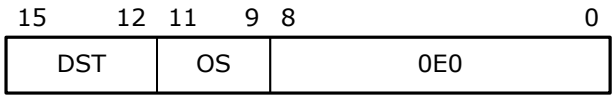
CFN R8,R6

SIZE – Resize operand in register

Mnemonic:

SIZE dst,OS

Format:



Group: 7

Description:

Three OS bits are written in the OpSize field of the register AFR [dst]. The instruction is used to replace the width of operands. The OS parameter can take the values BYTE, WORD, DWORD, QWORD, OWORD and NAN.

AFR[dst][24:22]<=OS

Altered flags in AFR[dst]:

CF[15:0] Stay unchanged.
ZF Stay unchanged.
SF Stay unchanged.
OF Stay unchanged.
IF Stay unchanged.
NF Stay unchanged.
DBF Stay unchanged.
OpSize Accepts the value of OS.
LICntr Set to zero for AFR[dst].
AMode Stay unchanged.

Protection violations:

None.

Example:

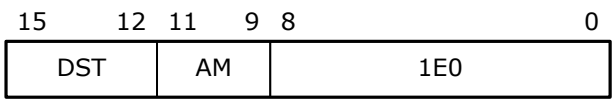
SIZE R11,DWORD

AMODE – Setup Address processing MODE

Mnemonic:

AMODE dst,AM

Format:



Group: 7

Description:

Three AM bits are recorded in the AMode field of the register AFR[dst]. AM can take values from 0 to 7.

AFR[dst][27:25]<=AM

Altered flags in AFR[dst]:

CF[15:0] Stay unchanged.
ZF Stay unchanged.
SF Stay unchanged.

OF	Stay unchanged.
IF	Stay unchanged.
NF	Stay unchanged.
DBF	Stay unchanged.
OpSize	Stay unchanged.
LICntr	Set to zero for AFR[dst].
AMode	Accepts the value of AM.

Protection violations:

None.

Example:

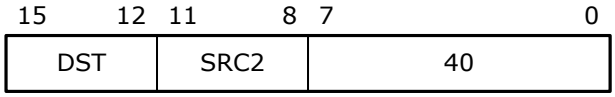
AMODE R9,5

COPY – Copy register-register

Mnemonic:

COPY dst,src2

Format:



Group: 7

Description:

The command copies the contents of the register R[src2] to the register R[dst], and copies the contents of the AFR[src2] register to the register AFR[dst].

R[dst]<=R[src2]

Altered flags in AFR[dst]:

AFR[dst]<=AFR[src2] without LICntr field.

LICntr sets to zero in AFR[dst] and AFR[src2].

Protection violations:

None.

Example:

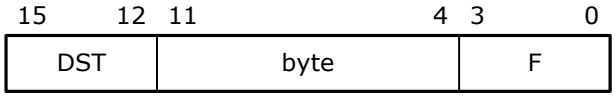
COPY R11,R1

LI – Load Immediate

Mnemonic:

LI dst,byte

Format:



Group: 7

Description:

Load the byte into the 8-bit part of the general-purpose register. The first LI instruction always loads the byte [7: 0] of register R[dst]. In this case, bit 7 of the byte is copied to all other bits of the register [127: 8]. The next LI instruction with the same dst value puts the data byte in the [15: 8] bits of the R [dst] register, and the most significant bit is copied into the [127: 16] bits of the register. Accordingly, the third instruction will load the third byte into bits [23:16] by expanding the character to bits [127: 24]. 16 LI instructions fully load a 128-bit register. For example, the sequence LI R3,78h; LI R3,56h; LI R3,34h; LI R3,12h will result in the formation of the number 12345678h in the register R3. A sequence of LI R9,0EFh; LI R9,0CDh will form in the register R9 the value 0FFFFFFFFFFFFFFFFFFFFFFFFFFFFCDEFh. LI commands can alternate with other commands, as well as with LI commands that have different values in

the dst field. The LICntr counter is zeroed out by any command other than LI and using the register as the source of the source operand or result receiver.

Altered flags in AFR[dst]:

All flags stay unchanged, but only LICntr receives an increment of 1

Protection violations:

None.

Example:

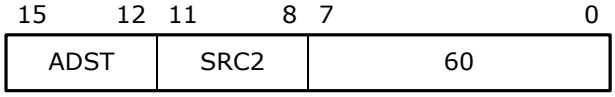
```
LI R8,0B4h
LI R9,11
LI R8,34          ; R8 loaded with 16-bit value 22B4h, R9 holds byte 0Bh
```

LAR – Load Address Register

Mnemonic:

LAR adst,src2

Format:



Group: 8

Description:

Load address register.

AR[adst]<=R[src2]

Registers AR[13], AR[14] and AR[15] can't be modified when CPL<>0.

Altered flags:

Field LICntr in AFR[src2] cleared.

Protection violations:

None.

Example:

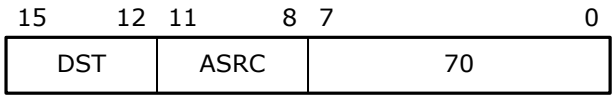
LAR AR6,R12

SAR – Store Address Register

Mnemonic:

SAR dst,asrc

Format:



Group: 8

Description:

Read address register.

R[dst]<=AR[asrc]

Altered flags:

Field LICntr in R[dst] cleared.

Protection violations:

None.

Example:

SAR R10,AR0

LDB/LDW/LDD/LDQ/LDO – Load data from memory

Mnemonic:

LDB dst,mar:src2
LDW dst,mar:src2
LDD dst,mar:src2
LDQ dst,mar:src2
LDO dst,mar:src2

Format:

15 12	11 8	7 5	4 0	
DST	SRC2	MAR	05	LDB
DST	SRC2	MAR	0D	LDW
DST	SRC2	MAR	15	LDD
DST	SRC2	MAR	1D	LDQ
DST	SRC2	MAR	14	LDO

Group: 8

Description:

Instruction loads data into register from local memory space or memory space of the another processor, or from data stream. The MAR address registers pair describes the object selector and the base offset. If the [31: 0] bits of the object selector are 0 or equal to the CPUNR number, then reading is done either from the local memory or from the stream controller, depending on the type of descriptor. The register R[src2] contains either an additional offset summed with an offset from the address register, or an increment of the base offset, or does not participate in the address generation at all. The register AFR [src2] contains the AMODE field, which defines the final offset generation mode and the basic offset modification mode.

Altered flags:

LICntr field of AFR[src2] and AFR[dst] set to 0.
The Size field in the AFR [dst] register is set depending on the operation applied, however, if reading data from an empty stream or from a non-existent processor in the network, the Size field is set to NaN and the NF bit is set to 1.

Protection violations:

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

Example:

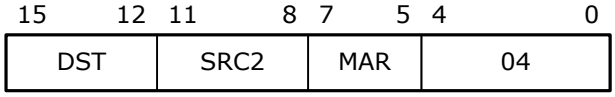
LDO R10,MAR3:R8

ST – Store data

Mnemonic:

ST mar:src2,dst

Format:



Group: 8

Description:

The command writes a data element either to local memory, or to the memory of another processor or to a local stream, or to a stream located in another processor. The bit size of the data element is determined by the Size field of the register AFR [dst]. A pair of address registers MAR contains an object selector and a base offset. If the [31: 0] bits of the object selector are 0 or the current CPUNR value, the data is transferred to either the local memory or the local flow controller, depending on the type of the object descriptor. If the selector refers to another processor, the transaction is transferred to the FPU. In the case of the CoreQuad processor, a transaction can be sent either to TMUX or to the stream controller of the neighboring core.

Altered flags:

Fields LICntr resets in the AFR[dst] and AFR[src2] registers.

Protection violations:

Violation of the object's limit.
Bad data selector, if the selector is zero or it exceeds the limit of the descriptor table.
Attempt to write to a write-protected object.
Object not accessible on a current privilege level.
Violation of object protection mechanism by TaskID value occurs.
Invalid descriptor type occurs.
Object can’t be accessible for any other cores in the multiprocessor network.
The processor with the specified number is not on the network.

Example:

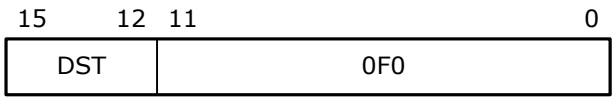
ST MAR2:R10,R3

PUSHD – Push data register into the stack

Mnemonic:

PUSHD dst

Format:



Group: 8

Description:

Pushing the contents of the general register and the corresponding flags register into the stack. During the operation, the stack pointer in AR14 is reduced by 24. First, a 32-bit flag register is pushed to the stack, supplemented with up to 64 bits, then the high-order 64 bits of the general register and then the lower 64 bits.

Altered flags:

Field LICntr in AFR[dst] cleared.

Protection violations:

Violation of the object's limit.

Bad data selector, if the selector is zero or it exceeds the limit of the descriptor table.
Attempt to write to a write-protected object.
Object not accessible on a current privilege level.
Violation of object protection mechanism by TaskID value occurs.
Invalid descriptor type occurs.
Object can’t be accessible for any other cores in the multiprocessor network.
The processor with the specified number is not on the network.

Example:

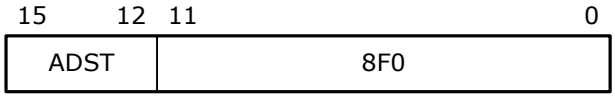
PUSHD R13

PUSHA – Push address register into the stack

Mnemonic:

PUSHA adst

Format:



Group: 8

Description:

Pushing the contents of the address register into the stack. The stack pointer is decremented by 8 and the value of the address register is written to the stack.

Altered flags:

None

Protection violations:

Violation of the object's limit.
Bad data selector, if the selector is zero or it exceeds the limit of the descriptor table.
Attempt to write to a write-protected object.
Object not accessible on a current privilege level.
Violation of object protection mechanism by TaskID value occurs.
Invalid descriptor type occurs.
Object can’t be accessible for any other cores in the multiprocessor network.
The processor with the specified number is not on the network.

Example:

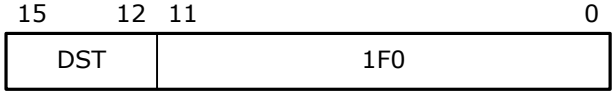
PUSHA AR12

POPD – Recover data register from the stack

Mnemonic:

POPD dst

Format:



Group: 8

Description:

Recovery from the stack of the general register and its register of flags. First, the lower 64 bits of the data register are restored, the stack pointer is incremented by 8, then the higher 64 bits of the data register are restored and the stack pointer is incremented by 8 and then the AFR[dst] is loaded and the stack pointer is incremented by 8 again.

Altered flags AFR[dst]:

All flags loaded from the stack, but only LICntr cleared

Protection violations:
Violation of the object's limit.
Bad data selector, if the selector is zero or it exceeds the limit of the descriptor table.
Attempt to write to a write-protected object.
Object not accessible on a current privilege level.
Violation of object protection mechanism by TaskID value occurs.
Invalid descriptor type occurs.
Object can’t be accessible for any other cores in the multiprocessor network.
The processor with the specified number is not on the network.
Example:

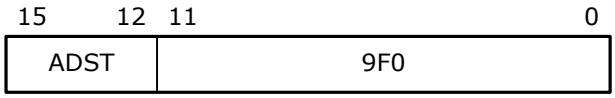
```
POPD R2
```

POPA – Recover address register from the stack

Mnemonic:

```
POPA adst
```

Format:



Group: 8
Description:
Recover address register from the stack and increase the stack pointer by 8.
Altered flags AFR[dst]:
None.
Protection violations:
Violation of the object's limit.
Bad data selector, if the selector is zero or it exceeds the limit of the descriptor table.
Attempt to write to a write-protected object.
Object not accessible on a current privilege level.
Violation of object protection mechanism by TaskID value occurs.
Invalid descriptor type occurs.
Object can’t be accessible for any other cores in the multiprocessor network.
The processor with the specified number is not on the network.
Example:

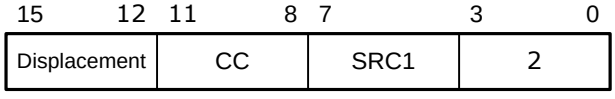
```
POPA AR13
```

JC – Jump if condition true

Mnemonic:

```
JC src1:cc,displacement
```

Format:



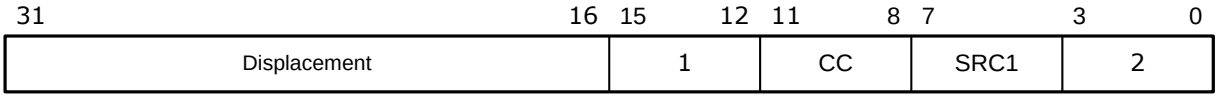
Group: 9
Description:
The instruction performs conditional transfer of control within the current code object to a distance of +7 and -8 16-bit words. The command specifies the register R[src1] whose flags are checked, the jump condition code CC and the 4-bit displacement of the first command relative to the JC command. The offset is expressed in 16-bit words, since the lengths of all commands are a multiple of 16-bits.

CC	*Branch condition*
0	ZF = 1
1	CF15 = 1
2	SF = 1
3	OF = 1
4	IF = 1
5	NF = 1
6	DBF = 1
7	Always executed
8	ZF = 0
9	CF15 = 0
A	SF = 0
B	OF = 0
C	IF = 0
D	NF = 0
E	DBF = 0
F	Always executed

Altered flags AFR[dst]:
LICntr zeroed.
Protection violations:
None.
Example:
NaNDetected:
JC R5:NF,displacement NaNDetected

JCL – Jump if condition true. Long form

Mnemonic:
JCL src1:cc,displacement
Format:



Group: 9
Description:
The instruction performs conditional transfer of control within the current code object to a distance of +32767 and -32768 16-bit words. The command specifies the register R[src1] whose flags are checked, the jump condition code CC and the 4-bit displacement of the first command relative to the JC command. The offset is expressed in 16-bit words, since the lengths of all commands are a multiple of 16-bits. This instruction must be aligned to the four-byte paragraph in the memory.

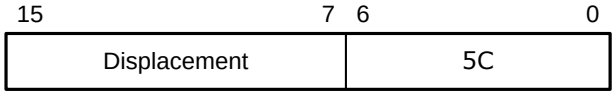
CC	*Branch condition*
0	ZF = 1
1	CF15 = 1
2	SF = 1
3	OF = 1
4	IF = 1
5	NF = 1
6	DBF = 1

7	Always executed
8	ZF = 0
9	CF15 = 0
A	SF = 0
B	OF = 0
C	IF = 0
D	NF = 0
E	DBF = 0
F	Always executed

Altered flags AFR[dst]:
LICntr zeroed.
Protection violations:
None.
Example:
NaNDetected:
JCL R5:NF,displacement NaNDetected

JNEAR – Near jump

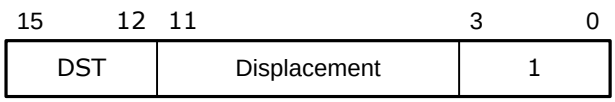
Mnemonic:
JNEAR displacement
Format:



Group: 9
Description:
The command transfers control within the limits of +255 and -256 of 16-bit words.
Altered flags:
None.
Protection violations:
None.
Example:
JNEAR displacement Label_A1

LOOP.

Mnemonic:
LOOP dst,displacement
Format:



Group: 9
Description:
The counter in the R[dst] register is decremented by 1. If a new value in the register R[dst] is not equal to 0, then a jump is performed using the specified displacement to calculate the address of the instruction that starts the loop. The transition is possible up to 256 instructions back (with displacement = 00h). With displacement = 0FFh, the transition will be made to 1 instruction back, to the instruction located in front of LOOP.
Altered flags:

None.

Protection violations:

None.

Example:

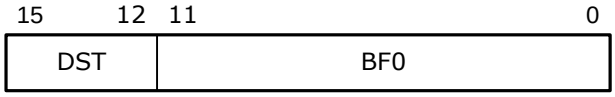
LOOP R7,displacement A0

JUMP – Jump by value from general register

Mnemonic:

JUMP dst

Format:



Group: 9

Description:

Unconditional jump to location specified by register content.

Altered flags:

None.

Protection violations:

None.

Example:

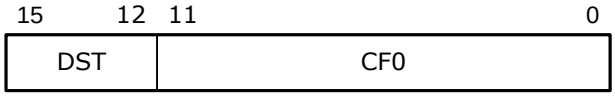
JUMP R9

CALL – Subprogram call

Mnemonic:

CALL dst

Format:



Group: 9

Description:

Call the subroutine who location specified by register content.

Altered flags:

None.

Protection violations:

Stack object violations:

Violation of the object's limit.

Bad data selector, if the selector is zero or it exceeds the limit of the descriptor table.

Attempt to write to a write-protected object.

Object not accessible on a current privilege level.

Violation of object protection mechanism by TaskID value occurs.

Invalid descriptor type occurs.

Object can’t be accessible for any other cores in the multiprocessor network.

The processor with the specified number is not on the network.

Example:

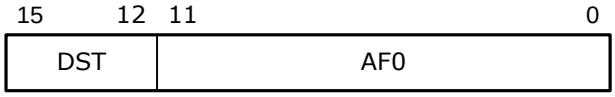
CALL R4

SENDMSG – Send message

Mnemonic:

SENDMSG dst

Format:



Group: 9

Description:

Sending a message. The message identifier is located in the [15: 0] bits of the R[dst] general 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.

Protection violations:

The message index is outside the table of imported procedures.
Invalid PSO selector to which the message is sent.
The index goes beyond the table of exported procedures.
Violation of access to the message handler by privilege level.
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.
There is no space in the message queue to write a message.

Example:

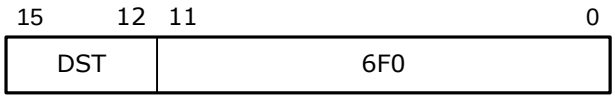
SENDMSG R9

MEMALLOC – Allocate memory block

Mnemonic:

MEMALLOC dst

Format:



Group: 9

Description:

The register R[dst] contains the original instruction parameter and takes the result of its execution. If the [23:0] bits of the register R[dst] contain zeros, then a new memory block is allocated. If the bits are non-zero, they are used as an object selector for the object deletion procedure. Bits [63:32] contain the value of the required length of the object, expressed in 32-byte paragraphs if the operation of allocating a new block is performed. The distributed block selector is returned to the R[dst] register if the selection was successful. If the block was not allocated or was deleted, the register will contain 0.

Altered flags:

None.

Protection violations:

None.

Example:

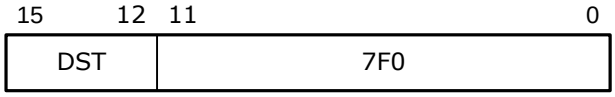
MEMALLOC R2

GETPAR – Get message parameter

Mnemonic:

GETPAR dst

Format:



Group: 9

Description:

The instruction is used to get the message handler a 32-bit message call parameter that was previously sent in the SENDMSG instruction.

Altered flags:

None.

Protection violations:

None.

Example:

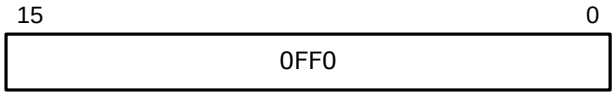
GETPAR R11

RET – Return from subroutine

Mnemonic:

RET

Format:



Group: 9

Description:

Returns from the subroutine within the current code object.

Altered flags:

None.

Protection violations:

Stack object violations:

Violation of the object's limit.

Bad data selector, if the selector is zero or it exceeds the limit of the descriptor table.

Attempt to write to a write-protected object.

Object not accessible on a current privilege level.

Violation of object protection mechanism by TaskID value occurs.

Invalid descriptor type occurs.

Object can’t be accessible for any other cores in the multiprocessor network.

The processor with the specified number is not on the network.

Example:

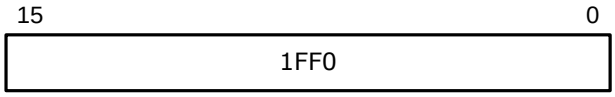
RET

ENDMSG – End of message processing

Mnemonic:

ENDMSG

Format:



Group: 9

Description:

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

Altered flags:

None.

Protection violations:

Empty context stack.

Invalid return PSO selector in context stack.

Example:

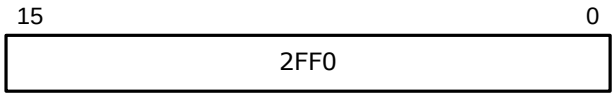
ENDMSG

BKPT – Breakpoint

Mnemonic:

BKPT

Format:



Group: 9

Description:

Instruction generates breakpoint interrupt. Used for software debugging.

Altered flags:

None.

Protection violations:

None.

Example:

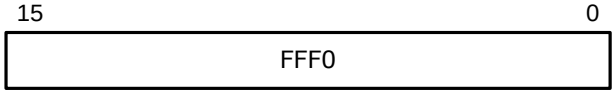
BKPT

NOP – No operations

Mnemonic:

NOP

Format:



Group: none

Description:

Instruction is used to align the placement of instructions with a 32-bit format, to the address that is a multiple of 32-bit double word.

Altered flags:

None.

Protection violations:

None.

Example:

NOP

Note.

Any instruction can cause object limit violation, if processor detects attempt to execute instruction, that located out of code object range.

MSB – most significant bit

Group. Instructions from different groups can be executed simultaneously if they are independent of each other by the source operands and the result receivers. X16E can execute two instructions from groups 1, 2 and 4 simultaneously if they are independent of each other.