SIGMA Instruction Set Manual

Version 1.0

Editors: Suchir Mishra, Arohan Deshpande, Pavan Bellam

Table of Contents

---

10. Platform Level Specification**
11. Graphics Processing Extension
12. Virtualization
13. Assembly Language**

Introduction

Overview

SIGMA (standing for Scalable Integrated General-purpose computing Machine Architecture) is a brand new RISC processor architecture

created to be highly scalable and powerful while still remaining relatively simple. This entire architecture is completely open and available to any individual or group to freely implement.

Supported Data Formats

The word size for SIGMA is 128 bits. Including all of the extensions, SIGMA supports operations on these data formats:

- Bit
- Byte
- Halfword (32 bits)
- Word (64 bits)
- Doubleword (128 bits)

Floating point:

- IEEE Half precision floating point - 16 bits
- IEEE Single precision floating point - 32 bits
- IEEE Double precision floating point - 64 bits
- IEEE Quad precision floating point - 128 bits
- Brain Float 16 - 16 bits
- Tensorfloat

Endianness

SIGMA has support only for a little endian byte order.

SIGMA Base Instruction Set Architecture

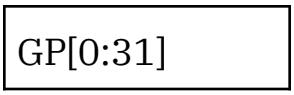
Registers

Program Counter

There is one program counter, which holds the memory address for the current instruction. Its width is dependent on the variant architecture.

General Purpose Registers

There is one general purpose register file, G, holding 32 general purpose registers that are each 128 bits wide. These can be used to store operands and results for arithmetic and logical operations, instruction pointers, or to hold the link to the current address before a jump. Their width is dependent on the variant of the architecture.



Flags Register

The Flags Register holds all the different flags describing the state of the processing core.

Carry Flag:
Overflow Flag:
Zero flag:

Condition flag:

Instructions

Integer Arithmetic Instructions

ADDR - Add Register

[0:6]	[7:11]	[12:16]	[17:21]	[22:31]
0000010	GP_1	GP_2	GP_DEST	SUB

The ADDR instruction adds two signed or unsigned integer operands held in any specified source GP registers and stores the result into a third specified destination GP register.

32 bit implementation:
This implementation will use a 32-bit instruction length, from bits 0 to 31. The carry flag is set if the result exceeds a width of 32 bits, and the lower 32 bits are stored in the destination register.

64 bit implementation:
This implementation will use a 64-bit instruction length, from bits 0 to 63. The carry flag is set if the result exceeds a width of 64 bits, and the lower 64 bits are stored in the destination register.

128 bit implementation:
This implementation will use a 128-bit instruction length, from bits 0 to 127. The carry flag is set if the result exceeds a width of 128 bits, and the lower 128 bits are stored in the destination register.

Exceptions:

ADDI - Add Immediate

[0:6]	[7:11]	[12:16]	[17:28]	[29:31]
0000000	GP_1	GP_DEST	IMMEDIATE	SUB

The ADDI instruction adds a 12-bit immediate value to an integer value held in any specified source GP register and stores the result into a third specified destination GP register.

32 bit implementation:
This implementation will use a 32 bit instruction length, from bits 0 to 31. The carry flag is set if the result exceeds a width of 32 bits, and the lower 32 bits are stored in the destination register.

64 bit implementation:

128 bit implementation:

Exceptions:

ADDRC - Add Register With Carry

[0:6]	[7:11]	[12:16]	[17:28]	[29:31]
OPCODE	GP_1	GP_DEST	IMMEDIATE	SUB

The ADDRC instruction adds two signed or unsigned integer operands held in any specified source GP registers, as well as the Carry Flag bit, and stores the result into a third specified destination GP register. In case of an overflow, the Carry Flag is set.

<u>32 bit implementation:</u>

This implementation will use a 32-bit instruction length, from bits 0 to 31. The carry flag is set if the result exceeds a width of 32 bits, and the lower 32 bits are stored in the GP_DEST register.

64 bit implementation:
This implementation will use a 64-bit instruction length, from bits 0 to 63. The carry flag is set if the result exceeds a width of 64 bits, and the lower 64 bits are stored in the GP_DEST register.

128 bit implementation:
This implementation will use a 128-bit instruction length, from bits 0 to 127. The carry flag is set if the result exceeds a width of 128 bits, and the lower 128 bits are stored in the GP_DEST register.

Exceptions:

ADDIC - Add Immediate With Carry

[0:6]	[7:11]	[12:16]	[17:28]	[29:31]
OPCODE	GP_1	GP_DEST	IMMEDIATE	SUB

The ADDIC instruction adds a 12-bit immediate value to an integer value held in any specified source GP register, as well as the Carry Flag bit, and stores the result into a third specified destination GP register.

32 bit implementation:

This implementation will use a 32-bit instruction length, from bits 0 to 31. The carry flag is set if the result exceeds a width of 32 bits, and the lower 32 bits are stored in the GP_DEST register.

64 bit implementation:

128 bit implementation:

Exceptions:

SUBR - Subtract Register

[0:6]	[7:11]	[12:16]	[17:21]	[22:31]
OPCODE	GP_1	GP_2	GP_DEST	SUB

The SUBR instruction subtracts an integer value, held in any specified source GP register (indicated by GP_2), from another integer value, also held in any specified source GP register (indicated by GP_1). The result is then stored into a third specified destination GP register.

32 bit implementation:
This implementation will use a 32-bit instruction length, from bits 0 to 31. The carry flag is set if the result exceeds a width of 32 bits, and the lower 32 bits are stored in the destination register.

64 bit implementation:

128 bit implementation:

Exceptions:

SUBI - Subtract Immediate

[0:6]	[7:11]	[12:16]	[17:28]	[29:31]
OPCODE	GP_1	GP_DEST	IMMEDIATE	SUB

The SUBI instruction subtracts a 12-bit immediate value from an integer value held in any specified source GP register and stores the result into a third specified destination GP register.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

SUBRB - Subtract Register with Borrow

[0:6] [7:11] [12:16] [17:21] [22:31]

OPCODE	GP_1	GP_2	GP_DEST	SUB

The SUBRB instruction subtracts an integer value, held in any specified source GP register (indicated by GP_2), from another integer value, also held in any specified source GP register (indicated by GP_1). The result is then stored into a third specified destination GP register.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

SUBIB - Subtract Immediate with Borrow

[0:6] [7:11] [12:16] [17:28] [29:31]

OPCODE	GP_1	GP_DEST	IMMEDIATE	SUB

Possible description: subtraction operation that considers the borrow flag?

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

MULT - Multiply

[0:6] [7:11] [12:16] [17:21] [22:31]

OPCODE	GP_1	GP_2	GP_DEST	SUB

The MULT instruction multiplies a signed or unsigned integer operand to another signed or unsigned integer held in any specified source GP registers and stores the result in a third specified destination GP register.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

DIV - Divide

[0:6]	[7:11]	[12:16]	[17:21]	[22:31]
OPCODE	GP_1	GP_2	GP_DEST	SUB

The DIV instruction divides an integer operand by another integer operand held in any specified source GP registers and stores the value in a third specified destination GP register.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

MOV - Move

[0:6]	[7:11]	[12:16]	[17:21]	[22:31]
OPCODE	GP_1	GP_2	GP_DEST	SUB

The MOV instruction copies the value from a source GP register and stores the value into the specified destination GP register.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

Address Calculation

ADDIPC - Add Immediate to Program Counter

[0:6]	[7:11]	[12:16]	[17:28]	[29:31]
OPCODE	GP_1	GP_DEST	IMMEDIATE	SUB

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

ADDRPC - Add register to Program Counter

[0:6]	[7:11]	[12:16]	[17:21]	[22:31]
OPCODE	GP_1	GP_2	GP_DEST	SUB

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

Logical Instructions

ANDR - AND Register

[0:6]	[7:11]	[12:16]	[17:21]	[22:31]
OPCODE	GP_1	GP_2	GP_DEST	SUB

ANDR performs a bitwise AND operation on two operands held in any specified source GP registers and stores the result in a third specified destination GP register.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

ANDI - AND Immediate

[0:6]	[7:11]	[12:16]	[17:28]	[29:31]
OPCODE	GP_1	GP_DEST	IMMEDIATE	SUB

ANDI performs a bitwise AND operation on two operands held in any specified source GP registers and stores the result in a third specified destination GP register.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

NOTR - NOT Register

[0:6]	[7:11]	[12:16]	[17:21]	[22:31]
OPCODE	GP_1	GP_2	GP_DEST	SUB

NOTR performs a bitwise NOT operation on two operands held in any specified source GP registers and stores the result in a third specified destination GP register.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

NOTI - NOT Immediate**

[0:6]	[7:11]	[12:16]	[17:28]	[29:31]
OPCODE	GP_1	GP_DEST	IMMEDIATE	SUB

NOTI performs a bitwise NOT operation on two operands, one held in any specified source GP register and the other in the 12-bit immediate field, and stores the result in another specified destination GP register.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

ORR - OR Register

[0:6] [7:11] [12:16] [17:21] [22:31]

OPCODE	GP_1	GP_2	GP_DEST	SUB

ORR performs a bitwise OR operation on two operands held in any two specified source GP registers, and stores the result in a third specified destination GP register.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

ORI - OR Immediate **

[0:6] [7:11] [12:16] [17:28] [29:31]

OPCODE	GP_1	GP_DEST	IMMEDIATE	SUB

ORI performs a bitwise OR operation on two operands, one held in any specified source GP register, and the other in the 12-bit immediate field, and stores the result in another specified destination GP register.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

XORR - XOR Register

[0:6] [7:11] [12:16] [17:21] [22:31]

OPCODE	GP_1	GP_2	GP_DEST	SUB

XORR performs a bitwise XOR operation on two operands held in any two specified source GP registers, and stores the result in a third specified destination GP register.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

XORI - XOR Immediate

[0:6] [7:11] [12:16] [17:28] [29:31]

OPCODE	GP_1	GP_DEST	IMMEDIATE	SUB

XORI performs a bitwise XOR operation on two operands, one held in any specified source GP register, and the other in the 12-bit immediate field, and stores the result in another specified destination GP register.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

LLS - Logical Left Shift

[0:6]	[7:11]	[12:16]	[17:28]	[29:31]
OPCODE	GP_1	GP_DEST	IMMEDIATE	SUB

LLS is a logical instruction that shifts an operand, held in any specified source GP register, to the left a number of times specified in the 12-bit immediate field. The result is stored in another specified destination GP register. This is done by removing the most-significant bit and appending a 0 as the least-significant bit.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

Logical Right Shift

[0:6]	[7:11]	[12:16]	[17:28]	[29:31]
OPCODE	GP_1	GP_DEST	IMMEDIATE	SUB

LLS is a logical instruction that shifts an operand, held in any specified source GP register, to the right a number of times specified in the 12-bit immediate field. The result is stored in another specified destination GP register. This is done by removing the least-significant bit and appending a 0 as the most-significant bit.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

ARS - Arithmetic Right Shift

[0:6]	[7:11]	[12:16]	[17:28]	[29:31]
OPCODE	GP_1	GP_DEST	IMMEDIATE	SUB

ARS is a logical instruction that shifts an operand, held in any specified source GP register, to the right a number of times specified in the 12-bit immediate field. The result is stored in another specified destination GP register. The shift is performed by removing the least-significant bit, and appending the same most significant bit as in the original operand as the most-significant bit.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

Bitfield Move/Move Instructions **

Load/Store Instructions

LWR - Load Word Register

[0:6]	[7:11]	[12:16]	[17:28]	[29:31]
OPCODE	GP_ADDR	GP_DEST	IMMEDIATE	001

The LWR instruction loads a word from memory into a specified GP register using an address held in another specified GP register, with an optional immediate offset.

Exceptions:
Misaligned address

SWR - Store Word Register

[0:6]	[7:11]	[12:16]	[17:28]	[29:31]
OPCODE	GP_WORD	GP_DEST	IMMEDIATE	001

The SWR instruction stores a word from a specified GP register into memory using an address held in another specified GP register, with an optional immediate offset.

LDR - Load Doubleword Register

[0:6]	[7:11]	[12:16]	[17:28]	[29:31]
OPCODE	GP_ADDR	GP_DEST	IMMEDIATE	001

The LWR instruction loads a doubleword from memory into a specified GP register using an address held in another specified GP register, with an optional immediate offset.

SDR - Store Doubleword Register

[0:6]	[7:11]	[12:16]	[17:28]	[29:31]
OPCODE	GP_WORD	GP_DEST	IMMEDIATE	001

The SWR instruction stores a doubleword from a specified GP register into memory using an address held in another specified GP register, with an optional immediate offset.

LHR - Load Halfword Register

[0:6]	[7:11]	[12:16]	[17:28]	[29:31]
OPCODE	GP_ADDR	GP_DEST	IMMEDIATE	010

The LHR instruction loads a halfword from memory into a specified GP register using an address held in another specified GP register, with an optional immediate offset.

SHR - Store Halfword Register

[0:6]	[7:11]	[12:16]	[17:28]	[29:31]
OPCODE	GP_HALFW.	GP_DEST	IMMEDIATE	010

The SHR instruction stores a halfword from a specified GP register into memory using an address held in another specified GP register, with an optional immediate offset.

LBR - Load Byte Register

[0:6]	[7:11]	[12:16]	[17:28]	[29:31]

OPCODE	GP_ADDR	GP_DEST	IMMEDIATE	SUB

The LDBR instruction loads a byte from memory into a specified GP register using an address held in another GP register, with an optional immediate offset.

SBR - Store Byte Register

[0:6] [7:11] [12:16] [17:28] [29:31]

OPCODE	GP_Byte	GP_DEST	IMMEDIATE	SUB

The SBR instruction stores a byte from a specified GP register into memory using an address held in another specified GP register, with an optional immediate offset.

LWRRO - Load Word Register with Register Offset

The LWRRO instruction loads a word from memory into a specified GP register using an address held in another GP register, with an optional register offset.

SWRRO - Store Word Register with Register Offset

The SWRRO instructions stores a word from a specified GP register into memory using an address held in another specified GP Register, with an optional register offset.

LDRRO - Load Doubleword Register with Register Offset

The LDRRO instruction loads a doubleword from memory into a specified GP register using an address held in another GP register, with an optional register offset.

SDRRO - Store Doubleword Register with Register Offset

The SDRRO instructions stores a doubleword from a specified GP register into memory using an address held in another specified GP Register, with an optional register offset.

LHRRO - Load Halfword Register with Register Offset

The LHRRO instruction stores a halfword from memory into a specified GP register using an address held in another GP register, with an optional register offset.

SHRRO - Store Halfword Register with Register Offset

The SHRRO instruction stores a halfword from a specified GP register into memory using an address held in another specified GP register, with an optional register offset.

LBRRO - Load Byte Register with Register Offset

The LQRRO instruction stores a byte from memory into a specified GP register using an address held in another GP register, with an optional register offset.

SBRRO - Store Byte Register with Register Offset

The SHRRO instruction stores a byte from a specified GP register into memory using an address held in another specified GP register, with an optional register offset.

LWPRO - Load Word PC with Register Offset

The LWPRO instruction loads a word from memory into a specified GP register using the current value of the PC, with an offset from a specified GP register, as the effective memory address.

SWPRO - Store Word PC with Register Offset

The SWPRO instruction stores a word from a specified GP register into memory using the current value of the PC, with an offset from a specified GP register, as the effective memory address.

LDPRO - Load Doubleword PC with Register Offset

The LDPRO instruction loads a doubleword from memory into a specified GP register using the current value of the PC, with an offset from a specified GP register, as the effective memory address.

SDPRO - Store Doubleword PC with Register Offset

The SDPRO instruction stores a word from a specified FP register into memory using the current value of the PC, with an offset from a specified GP register, as the effective memory address.

LHPRO - Load Halfword PC with Register Offset

The LHPRO instruction loads a halfword from memory into a specified GP register using the current value of the PC, with an offset from a specified GP register, as the effective memory address.

SHPRO - Store Halfword PC with Register Offset

The SHPRO instruction stores a halfword from a specified GP register into memory using the current PC value, with an offset from a specified GP register, as the effective memory address.

LBPRO - Load Byte PC with Register Offset

The LBPRO instruction loads a byte from memory into a specified GP register using the current value of the PC, with an offset from a specified GP register, as the effective memory address.

SBPRO - Store Byte PC with Register Offset

The SBPRO instruction stores a byte from a specified FP register into memory using the current value of the PC, with an offset from a specified GP register, as the effective memory address.

LWPIO - Load Word PC with Immediate Offset

The LWPIO instruction loads a word from memory into a specified GP register using the current value of the PC, with a 12-bit immediate offset, as the effective memory address.

SWPIO - Store Word PC with Immediate Offset

The SWPIO instruction stores a word from a specified GP register into memory using the current PC value, with a 12-bit immediate offset, as the effective memory address.

LDPIO - Load Doubleword PC with Immediate Offset

The LDPIO instruction loads a doubleword from memory into a specified GP register using the current value of the PC, with a 12-bit immediate offset, as the effective memory address.

SDPIO - Store Doubleword PC with Immediate Offset

The SDPIO instruction stores a doubleword from a specified GP register into memory using the current PC value, with a 12-bit immediate offset, as the effective memory address.

LHPIO - Load Halfword PC with Immediate Offset

The LHPIO instruction loads a halfword from memory into a specified GP register using the current value of the PC, with a 12-bit immediate offset, as the effective memory address.

SHPIO - Store Halfword PC with Immediate Offset

The SHPIO instruction stores a halfword from a specified GP register into memory using the current PC value, with a 12-bit immediate offset, as the effective memory address.

LBPIO - Load Byte PC with Immediate Offset

The LBPIO instruction loads a byte from memory into a specified GP register using the current value of the PC, with a 12-bit immediate offset, as the effective memory address.

SBPIO - Store Byte PC with Immediate Offset

The SBPIO instruction stores a byte from a specified GP register into memory using the current PC value, with a 12-bit immediate offset, as the effective memory address.

LWP - Load Word Pair

SWP - Store Word Pair

LHP - Load Halfword Pair

SHP - Store Halfword Pair

LBP - Load Byte Pair

SBP - Store Byte Pair

SWAP - Swap

The SWAP instruction exchanges a value in a specified GP register with a value from a specific memory address using an atomic load, store, and move operation.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

CAS - Compare and Swap

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

Jumps and Conditional Branches

JALR - Jump and link Register
The JAL instruction jumps to the address in memory supplied by any specified GP register, saving the value of the current PC + 4 to any specified GP register in order for returns.

[0:6] [7:11] [12:16] [17:31]

OPCODE	GP_1	GP_DEST	IMMEDIATE

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

JWLR - Jump Without Link Register

The JAL instruction jumps to the address in memory supplied by any specified GP register, without storing a return address into a register.

[0:6]	[7:11]	[12:16]	[17:31]
OPCODE	GP_1	GP_DEST	IMMEDIATE

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

JALI - Jump and Link Immediate

The JALI instruction jumps to a PC relative immediate offset, provided by the IMMEDIATE field in the instruction format, saving the value of the current PC + 4 to any specified GP register.

[0:6]	[7:11]	[12:16]	[17:31]
OPCODE	GP_1	GP_DEST	IMMEDIATE

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

BE - Branch if Equal

[0:6]	[11:15]	[16:20]	[21:25]	[26:31]
OPCODE	GP_1	GP_2	GP_3	SUB

The BE instruction compares two operands held in specified GP registers and then branches to the memory address held in the third specified GP register if the two operands are equal.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

- BRANCH_ADDRESS_MISALIGNED
- HIGHER_PRIVELEGE_NEEDED**

BNE - Branch if Not Equal

[0:6]	[11:15]	[16:20]	[21:25]	[26:31]
OPCODE	GP_1	GP_2	GP_3	SUB

The BNE instruction compares an operand held in a specified GP register to an immediate value and then branches to the memory address held in the third specified GP register if the two operands are not equal.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

- BRANCH_ADDRESS_MISALIGNED

BGT - Branch if Greater Than

[0:6]	[11:15]	[16:20]	[21:25]	[26:31]
OPCODE	GP_1	GP_2	GP_3	SUB

The BGT instruction compares two operands held in specified GP registers and then branches to the memory address held in the third specified GP register if the first operand is greater than the second operand.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

- BRANCH_ADDRESS_MISALIGNED

BGTE - Branch if Greater Than or Equal

[0:6]	[11:15]	[16:20]	[21:25]	[26:31]
OPCODE	GP_1	GP_2	GP_3	SUB

The BGTE instruction compares two operands held in specified GP registers and then branches to the memory address held in the third specified GP register if the first operand is greater than or equal to the second operand.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

- BRANCH_ADDRESS_MISALIGNED

BLT - Branch if Less Than

[0:6]	[11:15]	[16:20]	[21:25]	[26:31]
OPCODE	GP_1	GP_2	GP_3	SUB

The BLT instruction compares two operands held in specified GP registers and then branches to the memory address held in the third specified GP register if the first operand is less than the second operand.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

- BRANCH_ADDRESS_MISALIGNED

BLTE - Branch if Less Than or Equal Register

[0:6]	[11:15]	[16:20]	[21:25]	[26:31]
OPCODE	GP_1	GP_2	GP_3	SUB

The BLTE instruction compares two operands held in specified GP registers and then branches to the memory address held in the third specified GP register if the first operand is less than or equal to the second operand.

32 bit implementation:

64 bit implementation:

128 bit implementation:

Exceptions:

- BRANCH_ADDRESS_MISALIGNED

BZ - Branch if Zero

The BZ instruction checks the Zero Flag and branches to a given memory address if the flag is set.

Exceptions:

- BRANCH_ADDRESS_MISALIGNED

BNZ - Branch if Nonzero

The BNZ instruction checks the Zero Flag and branches to a given memory address if the flag is not set.

Exceptions:

- BRANCH_ADDRESS_MISALIGNED

BC - Branch on Carry

The BC instruction checks the Carry Flag and branches to a given memory address if the flag is set.

BO - Branch on Overflow

The BO instruction checks the overflow flag and branches to a given memory address if the flag is set.

Hints and System Instructions

NOP - No Operation

The NOP instruction is a hint that allows the processing core to increment the PC without performing any other operation.

WFI - Wait for Interrupt

The WFI instruction is a hint that halts the processing core until the next hardware interrupt is triggered. This instruction behaves as a NOP to any software running on the processing core.

WFE - Wait for Event

The WFE instruction is a hint that halts the processing core until an event is received. This instruction behaves as a NOP to any software running on the processing core.

SE - Send Event

The SE instruction is a hint that resumes the processes of all processing cores halted by the WFE instruction. This instruction behaves as a NOP to any software running on the processing core.

SLE - Send Local Event

The SLE instruction is a hint that resumes the processes of only the processing core that receives it. This instruction behaves as a NOP to any software running on the processing core.

Cache Management Instructions

Memory Barriers

Floating Point Extension

Global Exceptions

INSTRUCTION_MISALIGNED:

Description: The instruction being fetched is not at a properly aligned byte boundary.

ILLEGAL_INSTRUCTION:

Description: The current fetched instruction cannot be interpreted by the decoder.