Slicudis RISC Machine

(SRM)

The SRM architecture is a RISC ISA that was created by Santiago Licudis with the aim of surpassing current RISC architectures.
The base architecture consists of a processor with a bank of 32 registers (not all of them can be written to), each of 32 bits. The sizes of the address bus and the data bus are 32 bits each, giving access to 4 GiB of memory and reading/writing 4 bytes of data in a single clock cycle. Instructions have the size of 32 bits, making the fetch process faster and more efficient.

Instructions have an opcode size of 5 bits (32 opcodes). 17 of the opcodes are reserved for the base instructions, leaving the 15 remaining spaces for extensions.

INDEX

- 2-3 - Registers and the Status Register
- 4 - Vectors
- 5 - Base instruction set and instruction formats.
- 6 - Pseudo-Instructions and Extra data
- 7 - 64-bit extension.
- 8 - Multiplication and division extension.
- 8 - Floating point extension
- 9 - LEGAL

REGISTERS AND STATUS BITS

SRM processors contain 32 registers that are used for arithmetic operations, holding data, addresses, etc. These registers can be used for general purpose or for specific operations.

Address	Register	Function	Note
0x00	E0X	Constant Zero	Read-Only
0x01	EAX	Caller	Mainly used by the user
0x02	EBX	Caller	Mainly used by the user
0x03	ECX	Caller	Mainly used by the user
0x04	EDX	Caller	Mainly used by the user
0x05	EEX	Caller	Mainly used by the user
0x06	EFX	Caller	Mainly used by the user
0x07	EGX	Caller	Mainly used by the user
0x08	EHX	Caller	Mainly used by the user
0x09	EIX	Caller	Mainly used by the user
0x0a	EJX	Caller	Mainly used by the user
0x0b	EKX	Caller	Mainly used by the user
0x0c	EMX	Caller	Mainly used by the user
0x0d	ENX	Caller	Mainly used by the user
0x0e	EOX	Callee	Mainly used by the kernel
0x0f	EPX	Callee	Mainly used by the kernel
0x10	EQX	Callee	Mainly used by the kernel
0x11	ERX	Callee	Mainly used by the kernel
0x12	ESX	Callee	Mainly used by the kernel
0x13	ETX	Callee	Mainly used by the kernel
0x14	EUX	Callee	Mainly used by the kernel
0x15	EVX	Callee	Mainly used by the kernel
0x16	EWX	Callee	Mainly used by the kernel
0x17	EXX	Callee	Mainly used by the kernel
0x18	ERT	Subroutine Return Address	Automatically set by interrupts, should be globally used by subroutines
0x19	ESR	Status Register	Read-Only
0x1a	ECR	Count	Should be globally used by programmers for its function
0x1b	EDR	Destination register	Should be globally used by programmers for its function
0x1c	EIR	Index Register	Should be globally used by programmers for its function
0x1d	EBP	Base pointer	Should be globally used by programmers for its function
0x1e	ESP	Stack pointer	Should be globally used by programmers for its function
1x1f	EPC	Program Counter	Read-Only

E0X: Always holds 0x0 and can't be modified. It can be used for pseudo instructions like CMP, where the result is unused and only the flags are needed.

EAX-ENX: General purpose registers that are used by the user.

EOX-EPX: General purpose registers that the kernel uses for subroutines.

ERT: Holds the return address for subroutines.

ESR: Shows the contents of the status register, but you can only read it.

ECR: Used for program loops

EDR: Used for string operations

EIR: Used for string operations

EBP: Used to point to the base of the stack frame during function calls

ESP: Used for stack operations

EPC: Shows the contents of the program counter, but can't be modified

STATUS REGISTER: The Status Register contains the flags of the last ALU operations and control bits used by the processor.

Status Register	NOTE
0 - Zero (Z)	ALU flag
1 - Carry (C)	ALU flag
2 - Sign (S)	ALU flag
3 - Overflow (V)	ALU flag
4 - Enable hardware interrupts (IE)	Only modified by Kernel
5 - Virtual memory mode (VM)	Only modified by Kernel
6 - 64-bit mode (QWM)	Only for the 64-bit extension
7 - Kernel mode (KM)	Set/Reset by INT and URT, Sets to 1 at the start
8 - Floating point mode (FPM)	Only for the floating point extension

- **Zero:** Is set when the last ALU operation resulted in zero
- **Carry:** Is set when the last ALU operation had carry out or borrow out
- **Sign:** MSB of the result of the last ALU operation
- **Overflow:** Is set when an overflow happens
- **Enable hardware interrupts:** Enables the execution of interrupt subroutines. It can only be set/reset by the Kernel
- **Virtual memory mode:** Programs use virtual memory when this bit is active. It can only be set/reset by the Kernel
- **64-bit mode:** It can only be set/reset by kernel. ALU operations use 64 bits when this bit is set.
- **Kernel mode:** Is set by interrupts or the INT instruction and reset by the URT instruction
- **Floating point mode:** ALU integer operations switch to floating point operations when this bit is set. It can be set/reset by both the user and Kernel.

VECTORS FOR ADDRESSES

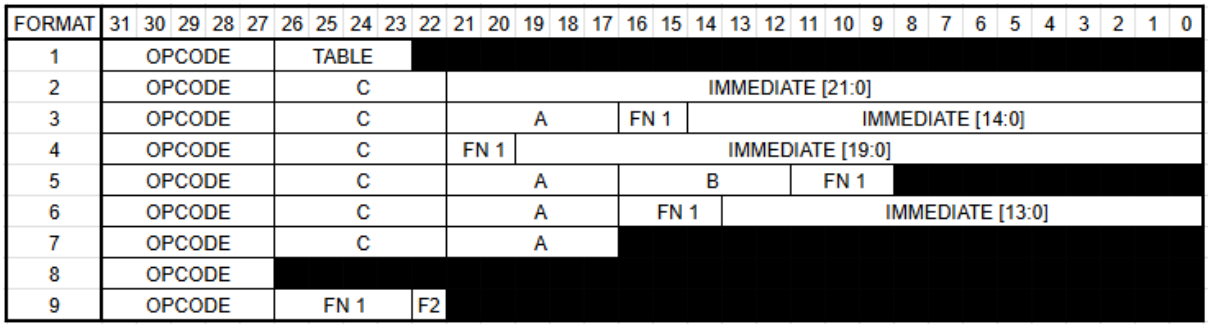
The address locations of the interrupt subroutines and the starting point of the execution of the processor's instructions are defined by the vectors, which are parts of memory that contain the addresses. The Kernel and BIOS should be the only program to be able to access those locations.

Address	Label	Name
0xfffffffc	rst_vec	Reset Vector
0x0	int_0	INT vector 0
0x4	int_1	INT vector 1
0x8	int_2	INT vector 2
0xc	int_3	INT vector 3
0x10	int_4	INT vector 4
0x14	int_5	INT vector 5
0x18	int_6	INT vector 6
0x1c	int_7	INT vector 7
0x20	int_8	INT vector 8
0x24	int_9	INT vector 9
0x28	int_10	INT vector 10
0x2c	int_11	INT vector 11
0x30	int_12	INT vector 12
0x34	int_13	INT vector 13
0x38	int_14	INT vector 14
0x3c	int_15	INT vector 15

BASE INSTRUCTION SET

Every SRM processor must be able to execute the base instructions to keep global compatibility between all SRM processors.

INSTRUCTION FORMATS: The base instruction set uses 10 formats to allocate the parameters and operands.



FORMAT	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	OPCODE					TABLE																										
2	OPCODE					C					IMMEDIATE [21:0]																					
3	OPCODE					C					A					FN 1			IMMEDIATE [14:0]													
4	OPCODE					C					FN 1	IMMEDIATE [19:0]																				
5	OPCODE					C					A					B					FN 1											
6	OPCODE					C					A					FN 1			IMMEDIATE [13:0]													
7	OPCODE					C					A																					
8	OPCODE																															
9	OPCODE					FN 1			F2																							

BASE INSTRUCTIONS:

OPCODE					Instruction	Mnemonic	Definition	Parameters	Format	Note
0	1	0	1	1	Interrupt	INT	Trigger an interrupt and enable kernel mode	-	1	
0	1	0	1	0	Move upper immediate	MVU	DEST [31:10] = IMMEDIATE	-	2	
0	1	0	0	0	Store Quad-Word	STQ	[A + IMM] = C [63:0]	FN 1 = 0b11		
0	1	0	0	0	Store Double-word	STD	[A + IMM] = C [31:0]	FN 1 = 0b10		
0	1	0	0	0	Store Word	STW	[A + IMM] = C [15:0]	FN 1 = 0b01	3	
0	1	0	0	0	Store Byte	STB	[A + IMM] = C [7:0]	FN 1 = 0b00		
0	1	0	1	0	Load Quad-Word (Only for the 64b ext.)	LDQ	C[63:0] = [A + IMM]	FN 1 = 0b11		
0	1	0	0	1	Load Double-word	LDD	C[31:0] = [A + IMM]	FN 1 = 0b10		
0	1	0	0	1	Load word	LDW	C[15:0] = [A + IMM]	FN 1 = 0b01	3	
0	1	0	0	1	Load byte	LDB	C[7:0] = [A + IMM]	FN 1 = 0b00		
0	0	1	0	0	Jump	JMP	PC = [C + IMM]	-	3	Signed imm
0	0	1	1	0	Jump if Zero	JZ	If Z: PC = [C + IMM]	FN 1 = 0b000		
0	0	1	1	0	Jump if Carry	JC	If C: PC = [C + IMM]	FN 1 = 0b001		
0	0	1	1	0	Jump if Sign	JS	If S: PC = [C + IMM]	FN 1 = 0b010		
0	0	1	1	0	Jump if Overflow	JV	If V: PC = [C + IMM]	FN 1 = 0b011	4	Signed imm
0	0	1	1	0	Jump if Not Zero	JNZ	If !Z: PC = [C + IMM]	FN 1 = 0b100		
0	0	1	1	0	Jump if Not Carry	JNC	If !C: PC = [C + IMM]	FN 1 = 0b101		
0	0	1	1	0	Jump if Not Sign	JNS	If !S: PC = [C + IMM]	FN 1 = 0b110		
0	0	1	1	0	Jump if Not Overflow	JNV	If !V: PC = [C + IMM]	FN 1 = 0b111		
0	0	0	0	0	Addition	ADD	C = A + B	FN 1 = 0b000		
0	0	0	0	0	Addition with carry	ADC	C = A + B + Cin	FN 1 = 0b001		
0	0	0	0	0	Subtraction	SUB	C = A - B	FN 1 = 0b010		
0	0	0	0	0	Subtraction with carry as borrow	SBC	C = A - B - Cin	FN 1 = 0b011	5	
0	0	0	0	0	Bitwise AND	AND	C = A & B	FN 1 = 0b100		
0	0	0	0	0	Bitwise OR	OR	C = A | B	FN 1 = 0b101		
0	0	0	0	0	Bitwise XOR	XOR	C = A ^ B	FN 1 = 0b110		
0	0	0	0	1	Addition (IMM)	ADDI	C = A + IMM	FN 1 = 0b000		
0	0	0	0	1	Addition with carry (IMM)	ADCI	C = A + IMM + Cin	FN 1 = 0b001		
0	0	0	0	1	Subtraction (IMM)	SUBI	C = A - IMM	FN 1 = 0b010		
0	0	0	0	1	Subtraction with carry as borrow (IMM)	SBCI	C = A - IMM - Cin	FN 1 = 0b011	6	
0	0	0	0	1	Bitwise AND (IMM)	ANDI	C = A & IMM	FN 1 = 0b100		
0	0	0	0	1	Bitwise OR (IMM)	ORI	C = A | IMM	FN 1 = 0b101		
0	0	0	0	1	Bitwise XOR (IMM)	XORI	C = A ^ IMM	FN 1 = 0b110		
0	0	0	1	0	Logical Right Shift	SHR	C = A >> 1	FN 1 = 0b000		
0	0	0	1	0	Aritmethical Right Shift	ASR	C = A >>> 1	FN 1 = 0b001		
0	0	0	1	0	Logical Right Shift by Nibble	SRN	C = A >> 4	FN 1 = 0b010	7	
0	0	0	1	0	Logical Right Rotation	ROR	C = Cin >> A >> 1	FN 1 = 0b011		
0	0	0	1	0	Logical Left Shift by Nibble	SLN	C = A << 4	FN 1 = 0b100		
0	1	1	1	0	Set status bit	SSB	Status bit [FN 2] = 1	FN 1 = 1	9	
0	1	1	1	0	Clear status bit	CSB	Status bit [FN 2] = 0	FN 1 = 0	9	
0	1	1	1	1	User mode return	URT	Return from an interrupt in user mode	-	8	

PSEUDO-INSTRUCTIONS

Pseudo-instructions are groups of instructions to make the equivalent of an instruction. These are some of the most used pseudoinstructions.

Pseudoinstructions	Equivalents
MVI [DEST], [IMM]	MVU [DEST], [31:10]
	ADDI [DEST], [DEST], [9:0]
PSH [SOURCE]	SUBI ESP, ESP, 4
	STD [SOURCE], ESP, 0
POP [DEST]	LDD [DEST], ESP
	ADDI ESP, ESP, 4
CAL [Location]	SUBI ESP, ESP, 4
	STD EPC, ESP, 0
	JMP (Location)
RET	LDD ERT, SP, 0
	JMP ERT, 0
CMP [A], [B]	SUB E0X, [A], [B]
TST [A], [B]	AND E0X, [A], [B]
NOP	ADD E0X, E0X, E0X
SYSCAL	INT 0x0
SHL	ADD [DEST], [A], [A]

EXTRAS

Extras	Code
Recommended interrupt subroutine	CSB
	PSH EDR
	PSH ESR
	SSB IE
	[CODE]
	CSB IE
	POP ESR
	POP EDR
	SSB IE
	URT

64-BIT EXTENSION

This extension adds 64-bit variants for 6 opcodes (22 instructions) and support for 64-bit arithmetic operations (which are enabled by the QWM flag). SRM processors with this extension still support the 32-bit instructions.

FORMATS:

FORMAT	63 62 61 60 59	58 57 56 55 54	53 52 51 50	49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
10	OPCODE	C	IMMEDIATE [53:0]	
11	OPCODE	C	A	FN 1 IMMEDIATE [46:0]
12	OPCODE	C	FN 1	IMMEDIATE [51:0]
13	OPCODE	C	A	FN 1 IMMEDIATE [45:0]

INSTRUCTIONS:

OPCODE					Instruction	Mnemonic	Definition	Parameters	Format	Note
1	0	0	0	0	Addition (IMM 64B)	ADDI	C = A + IMM	FN 1 = 0b000		
1	0	0	0	0	Addition with carry (IMM 64B)	ADCI	C = A + IMM + Cin	FN 1 = 0b001		
1	0	0	0	0	Subtraction (IMM 64B)	SUBI	C = A - IMM	FN 1 = 0b010		
1	0	0	0	0	Subtraction with carry as borrow (IMM 64B)	SBCI	C = A - IMM - Cin	FN 1 = 0b011	13	
1	0	0	0	0	Bitwise AND (IMM 64B)	ANDI	C = A & IMM	FN 1 = 0b100		
1	0	0	0	0	Bitwise OR (IMM 64B)	ORI	C = A | IMM	FN 1 = 0b101		
1	0	0	0	0	Bitwise XOR (IMM 64B)	XORI	C = A ^ IMM	FN 1 = 0b110		
1	0	0	0	1	Move Upper Immediate (64B)	MVU	C [54:10] = IMM	-	10	
1	0	0	1	0	Load Quad-Word (IMM 64B)	LDQ	C[63:0] = [A + IMM]	FN 1 = 0b11		
1	0	0	1	0	Load Double-word (IMM 64B)	LDD	C[31:0] = [A + IMM]	FN 1 = 0b10		
1	0	0	1	0	Load word (IMM 64B)	LDW	C[15:0] = [A + IMM]	FN 1 = 0b01	11	
1	0	0	1	0	Load byte (IMM 64B)	LDB	C[7:0] = [A + IMM]	FN 1 = 0b00		
1	0	0	1	1	Store Quad-Word (IMM 64B)	STQ	[A + IMM] = C [63:0]	FN 1 = 0b11		
1	0	0	1	1	Store Double-word (IMM 64B)	STD	[A + IMM] = C [31:0]	FN 1 = 0b10		
1	0	0	1	1	Store Word (IMM 64B)	STW	[A + IMM] = C [15:0]	FN 1 = 0b01	11	
1	0	0	1	1	Store Byte (IMM 64B)	STB	[A + IMM] = C [7:0]	FN 1 = 0b00		
1	0	1	0	0	Jump (IMM 64B)	JMP	PC = [C + IMM]	-	10	Signed imm
1	0	1	0	1	Jump if Zero (64B IMM)	JZ	If Z: PC = [C + IMM]	FN 1 = 0b000		
1	0	1	0	1	Jump if Carry (64B IMM)	JC	If C: PC = [C + IMM]	FN 1 = 0b001		
1	0	1	0	1	Jump if Sign (64B IMM)	JS	If S: PC = [C + IMM]	FN 1 = 0b010		
1	0	1	0	1	Jump if Overflow (64B IMM)	JV	If V: PC = [C + IMM]	FN 1 = 0b011	12	Signed imm
1	0	1	0	1	Jump if Not Zero (64B IMM)	JNZ	If !Z: PC = [C + IMM]	FN 1 = 0b100		
1	0	1	0	1	Jump if Not Carry (64B IMM)	JNC	If !C: PC = [C + IMM]	FN 1 = 0b101		
1	0	1	0	1	Jump if Not Sign (64B IMM)	JNS	If !S: PC = [C + IMM]	FN 1 = 0b110		
1	0	1	0	1	Jump if Not Overflow (64B IMM)	JNV	If !V: PC = [C + IMM]	FN 1 = 0b111		

MUL/DIV EXTENSION

This extension allows the SRM processors to perform signed multiplication, division and modulo. Multiplication with carry in is supported. The multiplication carry register is separate from the 1-bit carry flag from the status register.

OPCODE					Instruction	Mnemonic	Definition	Parameters	Format	Note
1	0	1	1	0	Multiply	MUL	C = A * B	FN 1 = 0b000		
1	0	1	1	0	Multiply with multiplication carry in	MLC	C = A * B + Carry in	FN 1 = 0b001		
1	0	1	1	0	Divide	DIV	C = A / B	FN 1 = 0b010	5	
1	0	1	1	0	Modulo / Remainder	MOD	C = A % B	FN 1 = 0b100		

FLOATING POINT EXTENSION

This extension allows SRM processors to perform floating points operations by enabling the FM flag and using the new instructions: ITF, FTI and SQR.

OPCODE					Instruction	Mnemonic	Definition	Parameters	Format	Note
1	0	1	1	1	Integer to float	ITF	C = float(A)	FN 1 = 0b000		
1	0	1	1	1	Float to integer	FTI	C = float(A)	FN 1 = 0b001	5	
1	0	1	1	1	Square root	SQR	C = Sqrt A	FN 1 = 0b010		

LEGAL

This project is under the MIT license, meaning that companies/people have to follow the terms the license: