Slicudis RISC Machine

(SRM)

The SRM architecture is a flagless 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). 11 of the opcodes are reserved for the base instructions, leaving the 15 remaining spaces for extensions.

Index

Registers

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
0x00	ZR	Constant 0 (Read-Only)
0x01	G1	Caller
0x02	G2	Caller
0x03	G3	Caller
0x04	G4	Caller
0x05	G5	Caller
0x06	G6	Caller
0x07	G7	Callee
0x08	G8	Callee
0x09	G9	Callee
0x0a	G10	Callee
0x0b	G11	Callee
0x0c	G12	Callee
0x0d	G13	General purpose
0x0e	G14	General purpose
0x0f	G15	General purpose
0x10	G16	General purpose
0x11	G17	General purpose
0x12	G18	General purpose
0x13	G19	General purpose
0x14	G20	General purpose
0x15	G21	General purpose
0x16	G22	General purpose
0x17	G23	General purpose
0x18	G24	General purpose
0x19	PR	Process Register
0x1a	SP	Stack Pointer
0x1b	JDR	Long jump destination
0x1c	SRD	Subroutine return destination
0x1d	IRD	Interrupt return destination
0x1e	SR	Status Register (Read-Only)
0x1f	PC	Program counter (Read-Only)

ZR: Always holds 0x0 and can't be modified.

G1-G6: General purpose registers that are used by the user.

G7-G12: General purpose registers that the kernel uses for subroutines.

G13-G26: Global general purpose registers

SP: Used in operations related to the stack

JDR: Used for holding addresses to jump to

SDR: Used to hold the return address for subroutines

IRD: Used to hold the return address for interrupts

SR: Shows the contents of the status register

PC: Shows the contents of P.C

PR: Used by the Kernel to indicate the process that is currently being run. It can only be modified if the K bit is enabled.

Status Register

The Status Register contains the flags of the last ALU operations and control bits used by the processor. The status bits can only be set/reset by the kernel (except for K, which is only set/reset by interrupts and URT)

Status Register	Function
0 - Kernel mode (K)	Enable the control over the Status Register
1 - Enable hardware interrupts (I)	Enable hardware interrupts
2 - Protected memory mode (P)	Set the memory controller to protected mode
3 - 64-bit mode (Q)	Set the aritmethic operations to 64-bit mode

- **Enable hardware interrupts:** Enables the execution of interrupt subroutines
- **Protected mode:** Programs use virtual memory when this bit is active. (Only applied if the processor has the protected mode ext.)
- **64-bit mode:** ALU operations use 64 bits when this bit is set. (Only applied if the processor has the 64-bit extension)
- **Kernel mode:** Is set by interrupts or the INT instruction and reset by the URT instruction

Address locations

SRM processors must use these locations for defining the address of interrupt subroutines and using the virtual memory table.

Address	Label	Name	Size (bytes)
0xffffffe8-f	$rst_vec	Reset Vector	8
0xfffffff0-7	$s_int	Software INT vector	8
0xfffffff8-f	$h_int	Hardware INT vector	8
0x0 - 0xfffff	$vm	Virtual memory table	1048576

- **Reset vector:** Contains the starting point address of the P.C
- **Software INT vector:** Contains the subroutine location of system calls
- **Hardware INT vector:** Contains the subroutine location of hardware interrupts
- **Virtual memory table:** Contains the table that indicates the addresses for virtual memory. It can only be accessed by the kernel.

Virtual memory vector format: This is the format of all the vectors from virtual memory:

{Process [61:48], Virtual page [47:24], Physical page [23:0]}

A page contains 4 KiB (4096 bytes). The Kernel has to use the VM to prevent unwished data overwriting and other memory-related problems.
This format allows you to access 64 GiB of memory if you have the 64-bit extension. That means that the max address bus size of an SRM64 processor is 36 bits.

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					C					IMMEDIATE [21:0]																					
2	OPCODE					C					A					FN 1	IMMEDIATE [14:0]															
3	OPCODE					C					A					B					FN 1	IMMEDIATE [8:0]									
4	OPCODE					C					A					B					FN 1										
5	OPCODE					C					A					FN 1	IMMEDIATE [13:0]															
6	OPCODE					C					A																					
7	OPCODE																															
8	OPCODE					FN 1				F2																					

BASE INSTRUCTIONS:

OPCODE					Instruction	Mnemonic	Definition	Parameters	Format	Note
0	1	0	1	0	System Call Interrupt	SYSCAL	Trigger an interrupt and enable kernel mode	-	7	
0	0	0	1	1	Move upper immediate	MVU	C [31:10] = IMMEDIATE	-	1	
0	0	1	0	0	Store Double-word	STD	[A + IMM] = C [31:0]	FN 1 = 0b10		
0	0	1	0	0	Store Word	STW	[A + IMM] = C [15:0]	FN 1 = 0b01	2	
0	0	1	0	0	Store Byte	STB	[A + IMM] = C [7:0]	FN 1 = 0b00		
0	0	1	0	1	Load Double-word	LDD	C[31:0] = [A + IMM]	FN 1 = 0b10		
0	0	1	0	1	Load word	LDW	C[15:0] = [A + IMM]	FN 1 = 0b01	2	
0	0	1	0	1	Load byte	LDB	C[7:0] = [A + IMM]	FN 1 = 0b00		
0	0	1	1	0	Jump	JMP	PC = [C + IMM]	-	2	Signed imm
0	0	1	1	1	Jump if Zero	JZ	If Z in A-B: PC = [C + IMM]	FN 1 = 0b000		
0	0	1	1	1	Jump if Carry (Borrow)	JC	If C in A-B: PC = [C + IMM]	FN 1 = 0b001		
0	0	1	1	1	Jump if Sign	JS	If S in A-B: PC = [C + IMM]	FN 1 = 0b010		
0	0	1	1	1	Jump if Overflow	JV	If V in A-B: PC = [C + IMM]	FN 1 = 0b011	3	Signed imm
0	0	1	1	1	Jump if Not Zero	JNZ	If !Z in A-B: PC = [C + IMM]	FN 1 = 0b100		
0	0	1	1	1	Jump if Not Carry	JNC	If !C in A-B: PC = [C + IMM]	FN 1 = 0b101		
0	0	1	1	1	Jump if Not Sign	JNS	If !S in A-B: PC = [C + IMM]	FN 1 = 0b110		
0	0	1	1	1	Jump if Not Overflow	JNV	If !V in A-B: PC = [C + IMM]	FN 1 = 0b111		
0	0	0	0	0	Addition	ADD	C = A + B	FN 1 = 0b000		
0	0	0	0	0	Subtraction	SUB	C = A - B	FN 1 = 0b001		
0	0	0	0	0	Bitwise AND	AND	C = A & B	FN 1 = 0b010	4	
0	0	0	0	0	Bitwise OR	OR	C = A | B	FN 1 = 0b011		
0	0	0	0	0	Bitwise XOR	XOR	C = A ^ B	FN 1 = 0b100		
0	0	0	0	1	Addition (IMM)	ADDI	C = A + IMM	FN 1 = 0b000		
0	0	0	0	1	Subtraction (IMM)	SUBI	C = A - IMM	FN 1 = 0b001		
0	0	0	0	1	Bitwise AND (IMM)	ANDI	C = A & IMM	FN 1 = 0b010	5	
0	0	0	0	1	Bitwise OR (IMM)	ORI	C = A | IMM	FN 1 = 0b011		
0	0	0	0	1	Bitwise XOR (IMM)	XORI	C = A ^ IMM	FN 1 = 0b100		
0	0	0	1	0	Logical Right Shift	SHR	C = A >> IMM	FN 1 = 0b000		
0	0	0	1	0	Aritmethic Right Shift	ASR	C = A >>> IMM	FN 1 = 0b001		
0	0	0	1	0	Logical Left Shift	SHL	C = A << IMM	FN 1 = 0b010	5	
0	0	0	1	0	Aritmethic Left Shift	ASL	C = A <<< IMM	FN 1 = 0b011		
0	0	0	1	0	Right Rotation	ROR	C = A ↻ IMM	FN 1 = 0b100		
0	0	0	1	0	Left Rotation	ROL	C = A ↺ IMM	FN 1 = 0b101		
0	1	0	0	0	Set status bit	SSB	Status bit [FN 2] = 1	FN 1 = 1	8	
0	1	0	0	0	Clear status bit	CSB	Status bit [FN 2] = 0	FN 1 = 0	8	
0	1	0	0	1	User mode return	URT	Return from an interrupt in user mode	-	7	

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 SP, SP, 4
	STD [SOURCE], SP, 0
POP [DEST]	LDD [DEST], SP
	ADDI SP, SP, 4

64-BIT EXTENSION

This extension adds 64-bit variants for 2 opcodes and extends 2 instructions (giving 4 new instructions in total) and support for 64-bit arithmetic operations (which are enabled by the Q flag). SRM processors with this extension still support the 32-bit instructions (If Q = 0)

OPCODE					Instruction	Mnemonic	Definition	Parameters	Format	Note
0	0	1	0	0	Store Quad-Word	STQ	[A + IMM] = C [63:0]	FN 1 = 0b11	2	
0	0	1	0	1	Load Quad-Word (Only for the 64b ext.)	LDQ	C[63:0] = [A + IMM]	FN 1 = 0b11	2	
0	1	0	1	1	Move upper immediate (High: High)	MVU.HH	C [63:42] = IMM	-	1	
0	1	1	0	0	Move upper immediate (High: Low)	MVU.HL	C [53:32] = IMM	-	1	

MUL/DIV EXTENSION

This extension implements signed multiplication, division and modulo. No new opcodes are implemented. The extension implements more operations to the ALU.

OPCODE					Instruction	Mnemonic	Definition	Parameters	Format	Note
0	0	0	0	0	Multiply	MUL	C = A * B	FN 1 = 0b101		
0	0	0	0	0	Divide	DIV	C = A / B	FN 1 = 0b110	4	
0	0	0	0	0	Modulo / Remainder	MOD	C = A % B	FN 1 = 0b111		
0	0	0	0	1	Multiply (IMM)	MULI	C = A * IMM	FN 1 = 0b101		
0	0	0	0	1	Divide (IMM)	DIVI	C = A / IMM	FN 1 = 0b110	4	Signed IMM
0	0	0	0	1	Modulo / Remainder (IMM)	MODI	C = A % IMM	FN 1 = 0b111		

LEGAL

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