The System from Scratch Instruction Set Architecture Specification

Version 0.0.0

Contents

1 Introduction

This document defines the System from Scratch Instruction Set Architecture (or the SfS ISA), which aims to describe a basic but fully-functional computer architecture.

2 Registers

The SfS ISA uses 16 registers. A brief summary of these registers is shown in Table ref.

Number	`sfsasm` Names	Description
0	`%0`	Always zero. Cannot be written.
1	`%1`, `%a`	
2	`%2`, `%b`	
3	`%3`, `%c`	
4	`%4`, `%d`	
5	`%5`, `%e`	
6	`%6`, `%f`	
7	`%7`, `%g`	
8	`%8`, `%h`	
9	`%9`, `%i`	
10	`%10`, `%c1`	
11	`%11`, `%c2`	
12	`%12`, `%ip`	
13	`%13`, `%st`	
14	`%14`, `%in`	
15	`%15`, `%pt`	

2.1 Zero Register

Register 0, or the zero register, will always have all of its bits read zero. Any instructions that write to this register will silently fail.

2.2 General Purpose Registers

Registers 1–9, or the general purpose registers A–I,

2.3 Comparison Registers

2.4 Instruction Pointer Register

2.5 Status Register

Register 13, or the status register, contains bits that represent the current status of the processor, including cartain bits which can be set to affect how the processor operates.

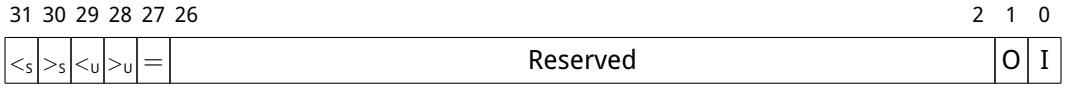


Figure 1: The bit format for the contents of the status register.

The five most sigificant bits of the status register shown in Figure 1 represent the results of comparison of the comparison registers. The bits each represent, in decreasing significance,

2.6 Interrupt Register

Register 14, or the interrupt register,

2.7 Page Table Pointer Register

Register 15, or the page table pointer register,

3 Data Types

The SfS ISA uses two primary data types: words and bytes. Words are 32 bits in size, and bytes are 8 bits in size. Bytes are only supported in memory accesses and are implicitly converted to and from words in these operations.

4 Instruction Format

The SfS ISA uses four formats for instructions: one each for operations on registers with registers, operations on registers with immediate values, conditional jumps, and memory access. The most significant two bits of each instruction indicate its format.

4.1 Register-Register Operations

The register-register operation instruction format is used to conditionally perform arithmetic or bitwise operations on registers.

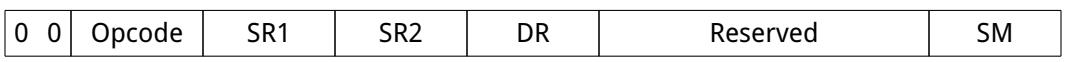


Figure 2: The bit format for register-register operation instructions.

For the instruction shown in Figure 2, the instruction would perform the operation specified by the opcode on the values in the register specified by SR1 and SR2 (for non-commutative operations like shifting, SR1 represents the first argument and SR2 represents the second) and store the result in the register specified by DR if the bits in SM match the flags in the status register.

4.2 Register-Immediate Operations

The register-immediate operation instruction format is used to perform arithmetic or bitwise operations on a register and an immediate value.

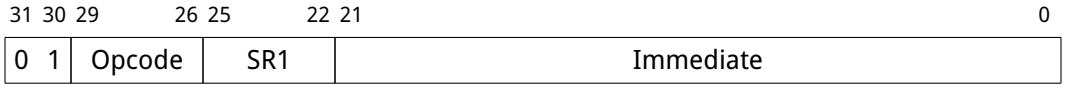


Figure 3: The bit format for register-immediate operation instructions.

For the instruction shown in Figure 3, the instruction would perform the operation specified by the opcode on the values in the register specified by SR1 and the immediate value, interpreted as either a zero- or sign-extended 32-bit value, depending on the opcode, (for non-commutative operations like shifting, SR1 represents the first argument and the immediate value represents the second) and store the result in the register specified by SR1.

4.3 Conditional Jumps

The conditional jump instruction format is used to conditionally change the value of the instruction pointer.

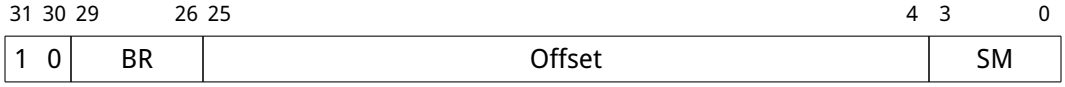


Figure 4: The bit format for conditional jump instructions.

For the instruction shown in Figure 4, the instruction would move the value specified by adding the value in the register specified by BR to the offset shifted left by 2 bits, interpreted as a sign-extended 32-bit value, if the bits in SM match the flags in the status register.

4.4 Memory Access

The memory access instruction format is used to encode instructions which move words and bytes between registers and memory.

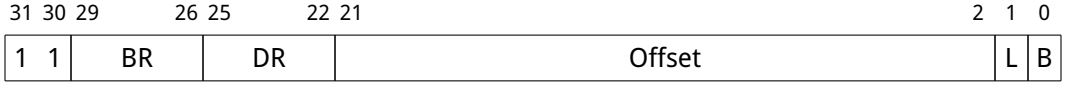


Figure 5: The bit format for memory access instructions.

For the instruction shown in Figure 5, the instruction would operate on the register specified by DR and the memory at the address specified by adding the value in the register specified by BR to the offset (interpreted as a sign-extended 32-bit value). The instruction will either store (move the register value into memory, if bit L is cleared) or load (move the value in memory into the register, if bit L is set) a word (if bit B is cleared) or a byte (if bit B is set).

4.5 Opcodes

The four opcode bits in the operation instruction formats encode information about the type of arithmetic or bitwise operation to be performed, as well as the interpretation of the immediate value (if there is one).

Opcode	`sfsasm` RRO Name	`sfsasm` RIO Name	Description
0000	`add`	`addi`	Adds the arguments, interpreting both arguments as 32-bit unsigned integers, setting the overflow status flag if the result cannot be represented as a 32-bit unsigned integer.
0001	`adds`	`addsi`	Adds the arguments, interpreting both arguments as 32-bit signed integers, setting the overflow status flag if the result cannot be represented as a 32-bit signed integer.
0010	`sub`	`subi`	Subtracts the second argument from the first argument, interpreting both arguments as 32-bit unsigned integers, setting the overflow status flag if the result cannot be represented as a 32-bit unsigned integer.
0011	`subs`	`subsi`	Subtracts the second argument from the first argument, interpreting both arguments as 32-bit signed integers, setting the overflow status flag if the result cannot be represented as a 32-bit signed integer.
0100	`shl`	`shli`	Shifts the first argument left by the number of bits specified in the second argument, filling the least significant bits with zeros, setting the overflow status flag if any of the discarded bits are non-zero.
0101	`shls`	`shlsi`	Shifts the first argument left by the number of bits specified in the second argument, filling the least significant bits with zeros, setting the overflow status flag if the result cannot be represented as a 32-bit signed integer.
0110	`shr`	`shri`	Shifts the first argument right by the number of bits specified in the second argument, filling the most significant bits with zeros, setting the overflow status flag if any of the discarded bits are non-zero.
0111	`shrs`	`shrsi`	Shifts the first argument right by the number of bits specified in the second argument, filling the least significant bits with the value of the original most significant bit.

1000	%and	%andi	
1001	%xor	%xori	
1010	%or	%ori	
1011	%nor	%nori	
1100	Reserved		
1101	Reserved		
1110	Reserved		
1111	Reserved		

4.6 Status Mask

The four SM bits in the conditionally-executing instruction formats encode information about the desired flags in the status register (which reflect the comparison of the values in the two comparison registers) to execute an instruction.

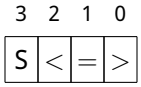


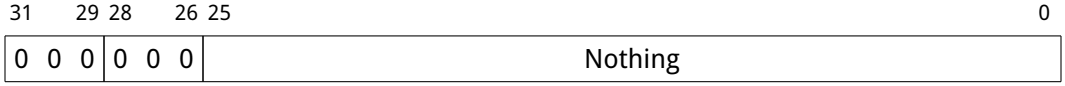
Figure 6: The bit format for the status mask bits in conditionally-executing instructions.

The most significant bit in the status mask shown in Figure 6, the S bit, will affect the interpretation of the values in the comparison registers. If the S bit is cleared, both of the values in the comparison registers will be interpreted as unsigned 32-bit integers. If the S bit is set, both of the values in the comparison registers will be interpreted as signed 32-bit integers.

The following three least significant bits in the status mask shown in Figure 6 indicate the type of comparisons that are to be evaluated. If the $<$ bit is set and the value in comparison register 1 is less than the value in comparison register 2 (when interpreted in the way that the S bit dictates), then the instruction containing the status mask will execute. Similarly, the instruction ontaining the status mask will also execute if the $=$ bit is set and the values in the comparison registers are equal or if the $>$ bit is set and the value in comparison register 1 is greater than the value in comparison register 2 (again, when interpreted in the way that the S bit dictates).

5 Instruction Set Reference

5.1 NOP: Null Operation



`sfsasm` Syntax: NOP (0x00000000), RTL: IP←IP