Detailed Constraints and Soundness Analysis for zkVM Instructions

This document provides a comprehensive breakdown of constraint systems and soundness analyses for arithmetic and comparison instructions used in the zkVM, based on 8-bit limb decomposition and secure computation over the Mersenne prime field $\mathbb{F}_{2^{31}-1}$, referred to as M31.

Field and Limb Representation

All registers are represented as 32-bit values, split into four 8-bit limbs:

$$x = x_0 + 2^8 x_1 + 2^{16} x_2 + 2^{24} x_3 \quad \text{with } x_i \in [0, 255]$$

Arithmetic is performed modulo 2^{32}, and all constraints are enforced within $\mathbb{F}_{2^{31}-1}$.

1 ADD: Register Addition

Semantics

$R[rd] = R[rs1] + R[rs2] \mod 2^{32}$

Constraints

Two-limb addition with carry:

$$\begin{aligned} a_L &= a_0 + 2^8 a_1, & b_L &= b_0 + 2^8 b_1, & c_L &= c_0 + 2^8 c_1 \\ a_H &= a_2 + 2^8 a_3, & b_H &= b_2 + 2^8 b_3, & c_H &= c_2 + 2^8 c_3 \end{aligned}$$

Carries $h_1, h_2 \in \{0, 1\}$:

$$b_L + c_L = a_L + h_1 \cdot 2^{16}$$
$$b_H + c_H + h_1 = a_H + h_2 \cdot 2^{16}$$

Test Cases

Case 1: $b = 1, c = 2$

- Expected result: $a = 3$
- Limb values: $b_0 = 1, c_0 = 2, a_0 = 3$, all other limbs and carries are 0

Case 2: $b = 0xFFFFFFFF, c = 1$

- Expected result: $a = 0x00000000$, carry propagates
- Limb-wise: $b_i = 255, c_0 = 1$, expect all carries = 1, all $a_i = 0$

Soundness

Each sum is $\leq 2^{16} + 2^{16} + 1 < 2^{18}$. All expressions remain within M31. Carries ensure correctness modulo 2^{32}.

2 ADDI: Register-Immediate Addition

Semantics

$R[rd] = R[rs1] + \texttt{sext}(imm) \mod 2^{32}$

Constraints

Same structure as `ADD`, with the immediate sign-extended to 32 bits and decomposed into 4 limbs. The constraints apply identically.

Soundness

Field-safe, with identical carry structure and constraints as `ADD`.

3 SLTU: Unsigned Register Comparison

Semantics

$R[rd] = (R[rs1] < R[rs2])_{\text{unsigned}}$

Constraints

Decompose into two 16-bit limbs:

$$b_L = b_0 + 2^8 b_1, \quad c_L = c_0 + 2^8 c_1$$
$$b_H = b_2 + 2^8 b_3, \quad c_H = c_2 + 2^8 c_3$$

Borrow flags $h_1, h_2 \in \{0, 1\}$:

$$b_L + h_1 \cdot 2^{16} = c_L + r_L$$
$$b_H + h_2 \cdot 2^{16} + h_1 = c_H + r_H$$
$$a_0 = h_2, \quad a_1 = a_2 = a_3 = 0$$

Test Cases

Case 1: $b = 0x00000001, c = 0x00000002$

- $b < c \Rightarrow a_0 = 1$
- Borrow propagates: $h_1 = 1, h_2 = 1$

Case 2: $b = 0x00000005, c = 0x00000003$

- $b > c \Rightarrow a_0 = 0$
- No borrow needed

Soundness

Correct unsigned comparison via borrow propagation. All intermediate values $\leq 3 \cdot 2^{16}$.

4 SLTIU: Unsigned Immediate Comparison

Semantics

$R[rd] = (R[rs1] < \texttt{sext}(imm))_{\text{unsigned}}$

Constraints

Same as SLTU, with c representing a 32-bit immediate split into 4 limbs.

5 SLT: Signed Register Comparison

Semantics

$R[rd] = (R[rs1] < R[rs2])_{\text{signed}}$

Constraints

Borrow comparison:

$$b_L + h_1 \cdot 2^{16} = c_L + r_L$$
$$b_H + h_2 \cdot 2^{16} + h_1 = c_H + r_H$$
$$h_1(1 - h_1) = 0, \quad h_2(1 - h_2) = 0$$

Sign bit extraction:

$$b_4 = \texttt{h-rem-b} + 2^7 h\_sgn\_b$$
$$c_4 = \texttt{h-rem-c} + 2^7 h\_sgn\_c$$
$$h\_sgn\_b, h\_sgn\_c \in \{0, 1\}$$

Final result logic:

$$a\_val(1) = h\_sgn\_b(1 - h\_sgn\_c) + h_2((1 - h\_sgn\_b)(1 - h\_sgn\_c) + h\_sgn\_b h\_sgn\_c)$$
$$a\_val(2) = a\_val(3) = a\_val(4) = 0$$

Test Cases

Case 1: $b = -5, c = 3$

- Sign bit comparison: $h\_sgn\_b = 1, h\_sgn\_c = 0 \Rightarrow a_0 = 1$

Case 2: $b = 3, c = -5$

- $h\_sgn\_b = 0, h\_sgn\_c = 1 \Rightarrow a_0 = 0$

Soundness

Covers all sign combinations:

- $b < c$ when signs differ: $h\_sgn\_b = 1, h\_sgn\_c = 0 \Rightarrow a = 1$
- $b > c$ when signs differ: $h\_sgn\_b = 0, h\_sgn\_c = 1 \Rightarrow a = 0$
- Matching signs: fallback to unsigned comparison (via borrow)

Field Safety: All expressions $\leq 2^{17} \ll 2^{31}$.

6 SLL: Shift Left Logical

Semantics

$R[rd] = R[rs1] \ll (R[rs2] \& 0x1F)$

Constraints

- Shift amount `shamt` $\in [0, 31]$ is extracted from $c\_val(1)$ using bits $sh1$ through $sh5$, with Boolean constraints.
- The value is partially shifted using a multiplier `exp3`, and byte shifts are controlled via $sh4$ and $sh5$.
- Shift propagation is tracked via auxiliary variables $qt1, ..., qt4$, and final output limbs $a_1, ..., a_4$ are determined based on shift bits.

Test Cases

- `shamt = 0`: $a = b$
- `shamt = 1`: verifies bit-level shift via $exp3$
- `shamt = 8`: 1-byte shift using $sh4$
- `shamt = 13`: bit + byte shift combo
- `shamt = 31`: MSB propagation only
- `shamt = 32`: masked to 0 via $shamt\&0x1F$

Soundness

Supports all $shamt \in [0, 31]$. No overflows due to maximum byte size (≤ 255).

7 SRA Instruction

Semantics

$R[rd] = R[rs1] \gg_{\text{arith}} (R[rs2]\&0x1F)$

Constraints

Same decomposition as SLL. Extract sign bit:

$$b_3 = h\_rem\_b + 2^7 h\_sgn\_b$$

Mask: $sra\_mask = h\_sgn\_b \cdot (exp3 - 1) \cdot exp3\_aux$ Shifted bits + mask yield final a_i.

Test Cases

Case 1: $b = -1, shamt = 1$

- Arithmetic shift right maintains sign: result remains $-1 = 0xFFFFFFFF$

Case 2: $b = 0x7FFFFFFF, shamt = 31$

- Result: $a = 0x00000000$, logical behavior, MSB not propagated

Soundness

Correctly applies sign extension when $R[rs1] < 0$. All intermediates are field-safe.

8 MUL Instruction

Semantics

$R[rd] = R[rs1] \cdot R[rs2] \mod 2^{32}$

Constraints

Decomposition:

$$\begin{aligned} b &= b_0 + 2^8 b_1 + 2^{16} b_2 + 2^{24} b_3 \\ c &= c_0 + 2^8 c_1 + 2^{16} c_2 + 2^{24} c_3 \\ a &= a_0 + 2^8 a_1 + 2^{16} a_2 + 2^{24} a_3 \end{aligned}$$

Partial Products:

$$\begin{aligned} p_0 &= b_0 c_0 \\ p_1 &= b_0 c_1 + b_1 c_0 \\ p_2 &= b_0 c_2 + b_1 c_1 + b_2 c_0 \\ p_3 &= b_0 c_3 + b_1 c_2 + b_2 c_1 + b_3 c_0 \end{aligned}$$

Carry-Based Reduction:

$$\begin{aligned} a_0 + 2^8 h_1 &= p_0 \\ a_1 + 2^8 h_2 &= p_1 + h_1 \\ a_2 + 2^8 h_3 &= p_2 + h_2 \\ a_3 + 2^8 h_4 &= p_3 + h_3 \end{aligned}$$

Clock and PC Advancement: Same as in `ADD` and other arithmetic instructions:

- $clk\_next = clk + 1$, via two-limb carry-checked modular addition.
- $pc\_next = pc + 4$, also via two-limb carry logic.

Test Cases

We verify the constraints through a series of test cases with explicit intermediate breakdowns:

Case 1: $b = 3, c = 5$

- **Limb Decomposition:** $b = [3, 0, 0, 0]$, $c = [5, 0, 0, 0]$
- **Partial Products:** $p_0 = 3 \cdot 5 = 15$, $p_1 = p_2 = p_3 = 0$
- **Carries:** $h_1 = h_2 = h_3 = h_4 = 0$
- **Output Limbs:** $a = [15, 0, 0, 0]$

Case 2: $b = 0xFF, c = 0xFF$

- **Limb Decomposition:** $b = c = [255, 0, 0, 0]$
- **Partial Products:** $p_0 = 255 \cdot 255 = 65025$
- **Carries:** $h_1 = 254$, $h_2 = h_3 = h_4 = 0$
- **Output Limbs:** $a = [1, 254, 0, 0]$

Case 3: $b = 0x01020304, c = 1$

- **Limb Decomposition:** $b = [4, 3, 2, 1],\ c = [1, 0, 0, 0]$
- **Partial Products:** $p_0 = 4,\ p_1 = 3,\ p_2 = 2,\ p_3 = 1$
- **Carries:** $h_1 = h_2 = h_3 = h_4 = 0$
- **Output Limbs:** $a = [4, 3, 2, 1]$

Case 4: $b = 0xFFFFFFFF, c = 2$

- **Limb Decomposition:** $b = [255, 255, 255, 255],\ c = [2, 0, 0, 0]$
- **Partial Products:** $p_0 = 510,\ p_1 = 510,\ p_2 = 510,\ p_3 = 510$
- **Carry Reductions:**
  - $a_0 = 254, h_1 = 2$
  - $a_1 = 0, h_2 = 2$
  - $a_2 = 0, h_3 = 2$
  - $a_3 = 254, h_4 = 1$
- **Final Output:** $a = [254, 0, 0, 254] = 0xFFFFFFFE$

These test cases illustrate that the carry-based decomposition correctly tracks overflow and distributes values across limbs without exceeding the field size bound.

Soundness

- All intermediates $\leq 4 \cdot 255^2 + 255 = 261375 < 2^{31} - 1$.
- Padding-safe: constraints are wrapped in $(1 -$ `is-local-pad`$)$.
- Final a is $b \cdot c \mod 2^{32}$.