# Appendix: Instruction Set Details

This appendix provides a detailed description of the arithmetic and logic instructions specified for the processor design.

---

## Arithmetic and Logic (ALU) Instructions

The following is a list of the ALU instructions to be supported, along with their functions and examples. Immediate addressing versions of instructions typically have the suffix "I".

### ADD – Add

**Description:** This instruction adds the values from two source registers and stores the result in a destination register. The immediate version adds a sign-extended immediate value to a source register.

**Examples:**

- `ADD R1, R2, R3` performs the operation $R1 \leftarrow R2 + R3$.
- `ADDI R3, #25` performs the operation $R3 \leftarrow R3 + 25$.

### SUB – Subtract

**Description:** This instruction subtracts the value of the second source register from the first and stores the result in a destination register. The immediate version subtracts a sign-extended immediate value from a source register.

**Examples:**

- `SUB R4, R5, R6` performs the operation $R4 \leftarrow R5 - R6$.
- `SUBI R5, #10` performs the operation $R5 \leftarrow R5 - 10$.

### AND – Bitwise AND

**Description:** This instruction performs a bitwise AND operation between two source registers and stores the result in a destination register. The immediate version performs a bitwise AND with a zero-extended immediate value.

**Examples:**

- `AND R1, R2, R3` performs the operation $R1 \leftarrow R2 \ \& \ R3$.
- `ANDI R2, #0xFF` performs the operation $R2 \leftarrow R2 \ \& \ 0xFF$.

**OR – Bitwise OR**

**Description:** This instruction performs a bitwise OR operation between two source registers and stores the result in a destination register. The immediate version performs a bitwise OR with a zero-extended immediate value.

**Examples:**

- `OR R1, R2, R3` performs the operation $R1 \leftarrow R2 \mid R3$.
- `ORI R2, #0xFF` performs the operation $R2 \leftarrow R2 \mid 0xFF$.

**XOR – Bitwise Exclusive OR**

**Description:** This instruction performs a bitwise XOR operation between two source registers and stores the result in a destination register. The immediate version performs a bitwise XOR with a zero-extended immediate value.

**Examples:**

- `XOR R4, R5, R6` performs the operation $R4 \leftarrow R5 \oplus R6$.
- `XORI R5, #0xABCD` performs the operation $R5 \leftarrow R5 \oplus 0xABCD$.

**NOR – Bitwise NOR**

**Description:** This instruction performs a bitwise NOR operation between two source registers and stores the result in a destination register.

**Example:** `NOR R7, R8, R9` performs the operation $R7 \leftarrow \neg(R8 \mid R9)$.

**NOT – Bitwise NOT**

**Description:** This instruction performs a bitwise NOT (inversion) on the source register's value and stores it in the destination register.

**Example:** `NOT R1, R2` performs the operation $R1 \leftarrow \neg R2$.

**SL – Shift Left**

**Description:** This instruction performs a logical left shift on the value in a register. The shift amount can be specified by another register or an immediate value.

**Examples:**

- `SLA R5, R7` shifts register R5 left by the amount specified in the least significant bit of R7 ($R5 \leftarrow R5 \ll R7[0]$).
- `SLAI R5, #1` shifts register R5 left by 1 bit ($R5 \leftarrow R5 \ll 1$).

**SRL – Shift Right Logical**

**Description:** This instruction performs a logical right shift on the value in a register, filling the most significant bits with zeros. The shift amount can be an immediate value or specified by another register.

**Example:** `SRLI R3, #4` performs the operation $R3 \leftarrow R3 \gg 4$.

### SRA – Shift Right Arithmetic

**Description:** This instruction performs an arithmetic right shift on the value in a register, preserving the sign bit by filling the most significant bits with copies of the original sign bit.

**Example:** `SRAI R3, #4` performs an arithmetic right shift: $R3 \leftarrow R34$.

### INC – Increment

**Description:** This instruction adds one to the value in the specified register.

**Example:** `INC R4` performs the operation $R4 \leftarrow R4 + 1$.

### DEC – Decrement

**Description:** This instruction subtracts one from the value in the specified register.

**Example:** `DEC R4` performs the operation $R4 \leftarrow R4 - 1$.

### SLT – Set on Less Than

**Description:** This instruction compares two registers. If the first is less than the second, the destination register is set to 1; otherwise, it is set to 0.

**Example:** `SLT R1, R2, R3` performs the operation $R1 \leftarrow (R2 < R3)?\ 1 : 0$.

### SGT – Set on Greater Than

**Description:** This instruction compares two registers. If the first is greater than the second, the destination register is set to 1; otherwise, it is set to 0.

**Example:** `SGT R1, R2, R3` performs the operation $R1 \leftarrow (R2 > R3)?\ 1 : 0$.

### LUI – Load Upper Immediate

**Description:** This instruction loads a 16-bit immediate value into the upper 16 bits of a register. The lower 16 bits are set to 0. This is useful for forming 32-bit constants.

**Example:** `LUI R1, #0xABCD` performs the operation $R1 \leftarrow 0xABCD0000$.

### HAM – Hamming Weight

**Description:** This instruction calculates the Hamming weight (or population count) of the value in the source register. It counts the number of set bits (1s) and stores the result in the destination register.

**Example:** `HAM R1, R2` counts the number of 1s in R2 and stores the count in R1. If $R2$ was $0x00000007$ (binary ...0111), the result would be $R1 \leftarrow 3$.

---

## Implementation Note for ALU Design

When completing the first step of the assignment (ALU design), please adhere to the following coding styles:

- For **logical instructions** such as `AND`, `OR`, `XOR`, `NOR`, and `NOT`, you may use **behavioral Verilog** code.
- For **arithmetic instructions** such as `ADD`, `SUB`, `INC`, and `DEC`, you must use **structural Verilog** code (e.g., by instantiating full-adder modules to create a ripple-carry adder).