**LAB 2 REPORT: DECODER FOR A BASIC RISC-V (RV32I) CORE**

**BY. EYOSIYAS ENDALAMAW**

**Introduction**

The primary goal of this lab was to design and simulate an instruction decoder for a basic RISC-V (RV32I) processor core. The decoder must accurately interpret all valid RV32I instructions and identify illegal ones, producing a control word that interfaces with the datapath. By implementing the decoder in VHDL and verifying its functionality through simulation, this lab reinforces core concepts in instruction classification, control signal generation, and hardware description language design.

**Design Overview**

The design features a combinational RV32I instruction decoder that converts a 32-bit instruction into the control word used by the datapath from Lab 1. Initially, the decoder extracts key components of the instruction, such as the opcode, funct3, funct7, rs1, rs2, and rd. It then classifies the instruction into one of several types (RR, RI, LD, ST, BR, JAL, JALR, LUI, AUIPC) and constructs the appropriate immediate value using a helper function that reconstructs and sign-extends the various instruction formats (with B/J types ensuring that bit 0 is set to zero).

Based on the identified instruction type, the decoder generates all necessary control signals, including register selects (Asel, Bsel, Dsel), write enable (Dlen), ALU source selects (PCAsel, IMMBsel), write-back select (PCDsel), and PC control signals (PCie and PCle), as well as branch flags (isBR, BRcond), the 4-bit ALU function, and the 32-bit immediate value (IMM).

For branch instructions, the ALU calculates the target address as PC + IMM(B), while the Branch Test Unit evaluates the outputs from the register file (rs1 compared to rs2) based on BRcond. A registered branch-taken signal is OR’d with the PCle control to determine the next PC load in the following cycle. For jump instructions, the targets are computed as PC + IMM(J) for JAL or rs1 + IMM(I) for JALR, ensuring that bit 0 is cleared for RV32I alignment.

In cases where opcodes or encodings are undefined, the decoder sets an illegal flag and provides safe default values. This design closely follows the “decoder spreadsheet” approach, effectively separating instruction classification, immediate generation, and control word synthesis, which enables seamless integration with the datapath designed in Lab 1.

I put my table here

A screenshot of a computer

AI-generated content may be incorrect.

A computer screen shot of a computer

AI-generated content may be incorrect.Fig 1: The top-level schematic of the decoder + datapath implementation. Each module (U\_REG, U\_ALU, U\_BRANCH, and U\_PC) is connected according to the RISC-V datapath, which in turn is connected to the decoder specification. Multiplexers manage data path selection based on control signals such as PCAsel, IMMBsel, and PCDsel.

**Simulation and verification**

I verified the decoder using an integrated connector testbench that combined both the decoder and the datapath. This testbench ran various typical RV32I instructions, including U, I, R types, branch instructions, JAL, JALR, and an illegal instruction. It used lightweight assertions and the read-back technique (with the instruction ADDI x0, rs1, 0) to verify register writes, while also monitoring the program counter (PC) behavior for cases like PC+4, branch taken/not taken, and JAL/JALR instructions. The waveforms show key signals, including ALU\_y, the immediate value (IMM), isBR/BRcond, the branch\_taken signal affecting PC loading, and the register file’s Dlen/Dsel, confirming correct control and timing across all scenarios.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

*Fig 2: Waveform result for each test pass*.

**Discussion and Conclusion**

The final decoder adheres to the RV32I specification and accurately interprets the Lab-1 control word for U/I/R instructions, branches, and jumps. By connecting the Branch Unit to compare register values (instead of ALU buses) and masking bit-0 during PC loads, the timing for branches and the alignment for JALR are functioning properly. Illegal encodings are properly identified, and all tested scenarios correspond with the anticipated PC transitions and register write-backs. Overall, the design is robust and prepared for integration with memory to enable complete load/store execution.