CSE 3055 Digital Logic Project Report - Iteration 1

Prepared By: Musa Özkan, Barış Giray Akman, Hakkı Kokur
Instructor: Betül Demiröz Boz

December 1, 2023

1 Students

Musa ÖZKAN - 150121058
Barış Giray AKMAN - 150121822
Hakkı KOKUR - 150120033

2 Introduction

This iteration of the project involves the design of a processor with a specific instruction set architecture (ISA). The processor has a defined width for addresses (10 bit) and registers (4 bits), and it is composed of several parts including a Register File, Instruction Memory, Data Memory, Control Unit, and an Arithmetic Logic Unit (ALU).

The processor supports a variety of operations, such as arithmetic operations, logical operations, memory access, and conditional jumps based on flags.

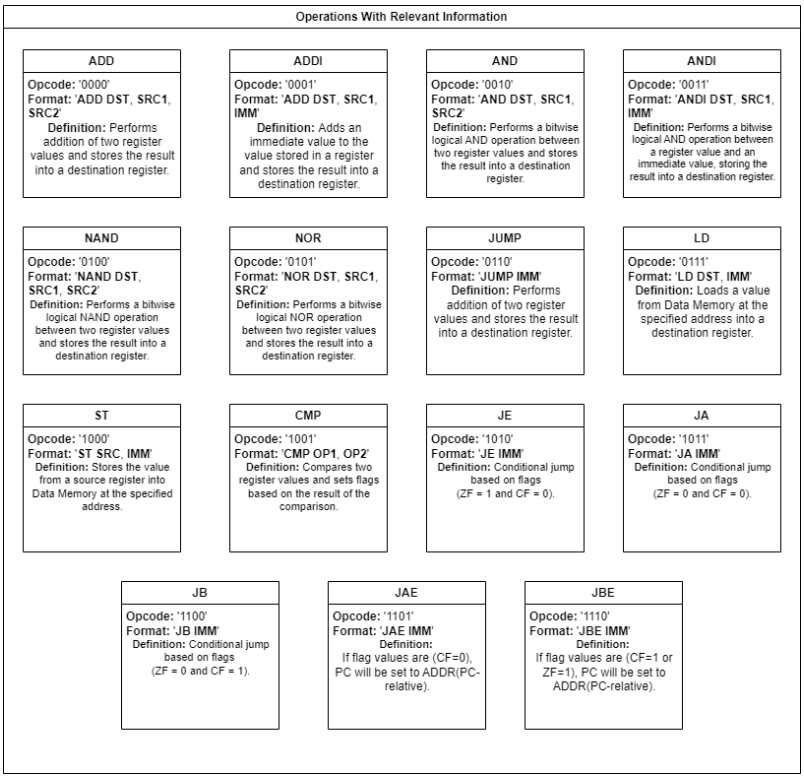

Figure 1: Table fo Instructions and Their Explanations

3 Instruction Set Architecture

The ISA was created by defining the opcode, operand registers, immediate values, and other necessary fields for each instruction. The ISA below has the details of format for each instruction type. For example, the ADD operation uses opcode '0000', specifying two source registers (SR1, SR2) and one destination register (DR).

The ISA design includes binary representations for each part of the instruction, which allows for direct translation from mnemonic to machine code.

	[17:14](opcode)	[13:10]	[9:6]	[5:2]	[1:0]
ADD	0000	DR	SR1	SR2	00
ADDI	0001	DR	SR1	IMM6	
AND	0010	DR	SR1	SR2	00
ANDI	0011	DR	SR1	IMM6	
NAND	0100	DR	SR1	SR2	00
NOR	0101	DR	SR1	SR2	00
JUMP	0110	ADDR10			0000
LD	0111	DR	ADDR10		
ST	1000	SR	ADDR10		
CMP	1001	OP1	OP2	000000	
JE	1010	ADDR10			0000
JA	1011	ADDR10			0000
JB	1100	ADDR10			0000
JAE	1101	ADDR10			0000
JBE	1110	ADDR10			0000

Figure 2: Instruction Set Architecture

4 Code Explanation

The python script we wrote serves as an assembly language-to-machine code converter for a processor design.

4.1 Workflow Summary:

The script reads assembly-like instructions from a file, parses them to categorize each instruction, and applies specific conversion functions based on their types. By utilizing these functions, the script generates binary representations for every parsed instruction. Subsequently, it transforms these binary representations into their corresponding hexadecimal format before writing the output into a designated file. This comprehensive process allows for the translation of diverse assembly-like instructions into a machine-readable hexadecimal format, essential for testing and validating the functionality of a processor according to its specific design requirements.

4.2 Functionality Overview:

4.2.1 File Operations:

- Imports the OS module for file-related operations.
- Defines a function (read_file) to read lines from a file and strip white space.

4.2.2 Instruction Parsing:

- Functions extract operations and operands from assembly-like instructions.

4.2.3 Conversion Functions:

Handles conversion of various instruction types:
• Arithmetic and logical instructions (ADD, ADDI, AND, ANDI, NAND, NOR).
• Conditional jump instructions (JE, JA, JB, JAE, JBE) based on flag conditions.
• Load and store instructions (LD, ST).
• Unconditional jump instruction (JUMP).
• Compare instruction (CMP).

4.2.4 Binary and Hexadecimal Conversion:

• Converts extracted instructions into their respective binary representations.
• Converts binary strings to hexadecimal format.

4.2.5 Output File Generation:

Creates an output file and writes the converted hexadecimal representations into it, line by line.

5 Input & Output Formats Explained

All of the inputs written adhere to the Instruction Set Architecture (ISA) specifications, maintaining consistent formatting and structure. Each line within the input file represents a distinct instruction, reflecting the defined operations compatible with the processor's ISA. The guidelines ensure clarity and coherence in the assembly-like instructions. Operations are delineated by spaces, while registers are separated by commas, providing a clear distinction between operands. Immediate values, when required, follow registers and are also separated by commas, enhancing readability and precise interpretation of each instruction by the processor according to the ISA guidelines.

The output of the conversion process yields a series of hexadecimal numbers, each representing machine code derived from the assembly-like instructions. Structured line-by-line, every hexadecimal value corresponds to a binary representation of an instruction from the sequence processed. This output conforms to the Instruction Set Architecture (ISA) and is machine-readable, facilitating interpretation and execution by the processor. These hexadecimal values serve as crucial input for the processor's logic, enabling the execution and testing of operations precisely defined by the assembly-like instructions, ensuring their accurate implementation according to the ISA. Ultimately, this formatted output is pivotal in guiding the processor's execution flow as it interprets and carries out the specified operations.