16-Bit Processor

Jawad Ahmed(533047)
Hassan Ali(512606)

January 12, 2025

—

Information and Communication Technology

—

Prof. Asad Mansoor

PROJECT OVERVIEW

The processor is based on a Harvard Architecture with the following components:

Key Components:

Instruction Memory (ROM):
A read-only memory module to store program instructions.
Data Memory:
A read-write memory module for handling data operations.
Main Datapath:
Includes 8 general-purpose registers.
Special-purpose registers for operations such as LO,HI,SR.
A robust ALU capable of executing various instructions.
Control Unit:
Manages the execution of instructions and interacts with the datapath via control and status signals.

Objectives:

Our project aims to build an instruction set processor in Verilog, leveraging the Harvard architecture with separate instruction and data memories. It will feature a register file, a versatile ALU, and execute instructions defined in the main Datapath. All of the instructions are stored in ROM and are incremented using PC. This processor promises efficient control and execution through dedicated control and status signals.

PROCESSOR DESIGN

Contribution/Work Division:

Team Member Name	Contribution
Hassan Ali	Compilation and Linking Of MOdules Removing Errors from Code
Jawad Ahmed	Writing Modules and implementing Different Types Of Instructions

Processor Design:

The processor supports three types of instruction formats, derived from the MIPS instruction set:

Instruction Types:

Register Type (R):

Format: Opcode | Rd | Rs | Rt | Shamt

Example: add Rd, Rs, Rt

Immediate Type (I):

Format: Opcode | Rd | Rs | Immediate

Example: addi Rd, Rs, Constant

Jump Type (J):

Format: Opcode | Address

Example: j Address

MODULES USED

Verilog Modules

The processor design is divided into modular components, each implemented in Verilog HDL for functionality and clarity.

Instruction Memory

Purpose: Stores the program's instructions in a Read-Only Memory (ROM) format.

Implementation:

The ROM is initialized with pre-defined instructions.

The Program Counter (PC) fetches instructions sequentially from the ROM.

Example Functionality:

When the PC points to a specific address, the corresponding instruction is fetched and sent to the control unit for decoding.

Data Memory

Purpose:

Acts as a Read-Write memory, storing and retrieving data used during program execution.

Implementation:

Contains multiple memory locations (16-bit wide) for storing intermediate or final results.

Data is accessed using memory addresses provided by instructions like lw (load word) and sw (store word).

Example Functionality:

For the instruction lw $Rd, offset($Rs), the memory address is calculated using $Rs + offset, and the value at that memory address is loaded into $Rd.

ALU (Arithmetic Logic Unit)

Purpose: Performs all arithmetic and logical operations specified in the instruction set.

Implementation:

Inputs: Operands from the registers and control signals from the control unit.

Outputs: Results of operations and status flags (e.g., Zero, Negative, Overflow).

Operations Supported:

- Arithmetic: add, sub, addi
- Logical: and, or
- Shifts: sll (shift left logical), srl (shift right logical)
- Multiplication: Generates results in special-purpose registers Hi and Lo.

Example Functionality:
For add $Rd, $Rs, $Rt, the ALU adds the values from $Rs and $Rt and stores the result in $Rd.

Control Unit

Purpose:
The brain of the processor, responsible for decoding instructions and generating control signals to guide the datapath components.

Implementation:
Decodes the opcode and identifies the type of instruction (R, I, or J). Generates control signals to enable specific operations in the ALU, data memory, and register file.

Example Functionality:
For the lw instruction, the control unit enables:
Memory read signal for data memory.
Write signal for the destination register.
Address calculation logic using $Rs and the offset.

Register File

Purpose:
A collection of 16-bit registers used for storing temporary data during execution.

Implementation:
Consists of 8 general-purpose registers.
Special-purpose registers:
Hi and Lo: Store results from multiplication operations.
Status Register (SR): Stores flags like Zero, Negative, Overflow, and Carry.

Example Functionality:
The add $Rd, $Rs, $Rt instruction retrieves values from $Rs and $Rt, performs the addition in the ALU, and writes the result to $Rd

WORKING AND FLOW OF DATA

The execution of an instruction in the processor follows a systematic flow:

1. Fetch

The Program Counter (PC) points to the memory address of the next instruction.
The instruction is fetched from the ROM and sent to the control unit.

2. Decode

The control unit decodes the fetched instruction to determine:
The operation (e.g., add, load, store).
The type of instruction (R, I, or J).
The operands involved (registers or memory addresses).

3. Execute

The ALU performs the operation based on the decoded instruction and control signals.
For example:
add $Rd, $Rs, $Rt: ALU adds the values from $Rs and $Rt.
sll $Rd, $Rs, shamt: ALU shifts the value in $Rs left by shamt.

4. Memory Access

For instructions like lw and sw:
The memory module is accessed to load or store data.
Example:
lw $Rd, offset($Rs): Data is loaded from Memory[$Rs + offset] into $Rd.

5. Write Back

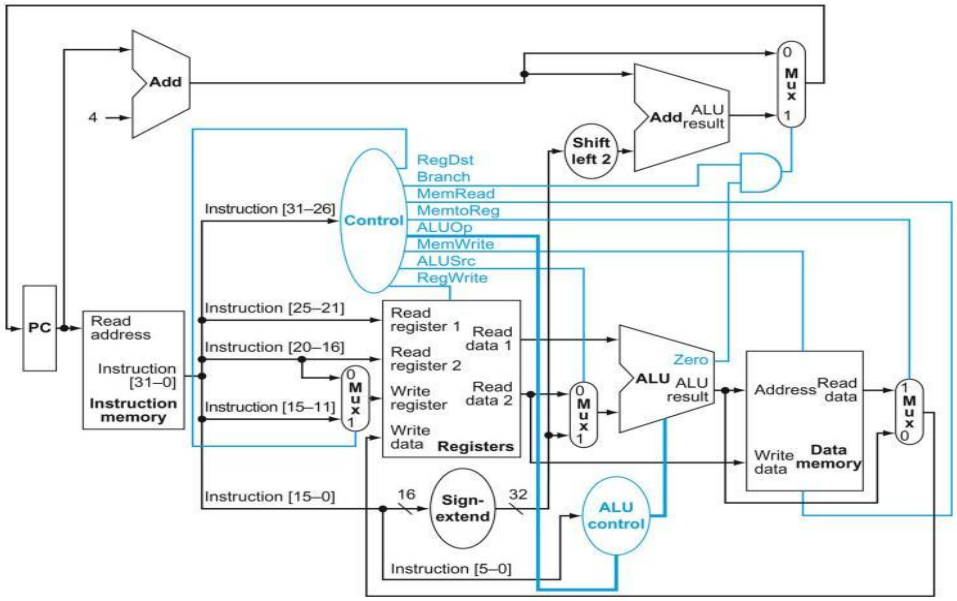
Results are written back to the destination register.
Special-purpose registers (e.g., Status Register) are updated with flags like Zero, Negative, Overflow, or Carry when applicable.

6. PC Update

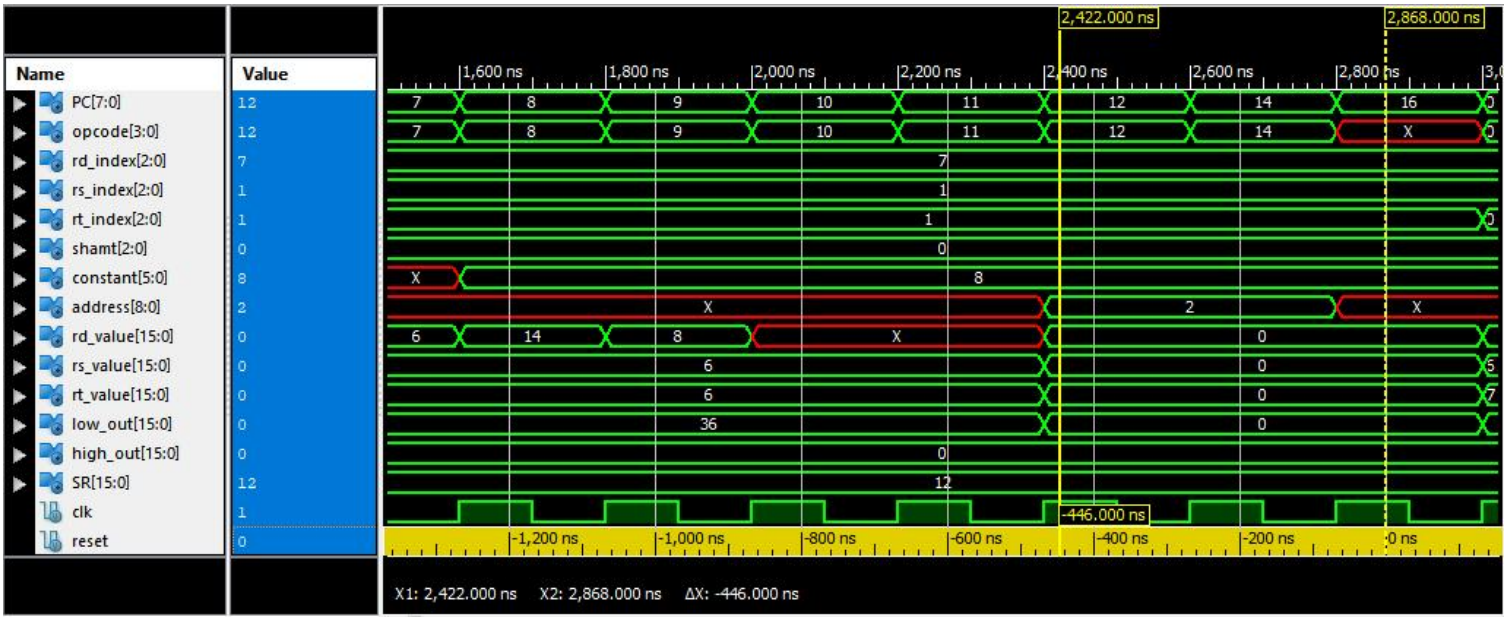
For sequential instructions, the PC increments to point to the next instruction.
For jump and branch instructions, the PC is updated based on the address or condition.

OVERVIEW

The Control Unit acts as the processor's mastermind, analyzing instructions retrieved from memory. When an arithmetic or logic operation arises, it doesn't directly perform the task. Instead, it functions as a dedicated communicator, dispatching specific control signals to the Datapath module. Think of the Datapath as the execution unit, obediently fulfilling the Control Unit's instructions. It performs the designated operation, whether calculations or comparisons, and promptly transmits the outcome back to the Control Unit. This seamless collaboration ensures precise and controlled execution of every instruction within the processor's architecture.



OUTPUT:



CONCLUSION

The custom 16-bit processor project successfully implements a simplified yet functional processor architecture. By integrating various components and designing a robust instruction set, the project demonstrates the practical application of Verilog HDL in computer system design.