RTL MIPS-like Processor Design

Group 5

Members:

M.B Sushanth - 230101061
Akshay Bobbala - 230101009
Aditya Bhawsar - 230101006
G.Mani Prakash - 230101043

Problem Statement

Implement a subset of the MIPS processor.

Detailed Module Descriptions

1. signextend

This module performs sign extension on a 16-bit input value to convert it into a 32-bit signed value, preserving its sign.
The most significant bit of the input (bit 15) is replicated 16 times to form the other more significant 16 bits of the 32-bit output.
This operation is essential when dealing with immediate values in instructions like ‘lw’, ‘sw’, ‘beq’ and ‘addi’.

2. d_flip_flop

Implements a D-type flip-flop with asynchronous reset. The flip-flop samples the input 'd' on the rising edge of the clock and outputs the value on 'q'. If reset is set to 1, the output is reset to zero immediately, regardless of the clock.

3. register32

This module uses 32 instances of the ‘d_flip_flop’ module to implement a 32-bit register. Each bit is stored and updated independently. The register updates on the rising edge of the clock and supports asynchronous reset. Used extensively in the register file and memory-

mapped registers for storing 32-bit values.

4. instruction_memory

A combinational memory array with 256 32-bit words. It is used to store and fetch instructions. On assertion of the `inst_reset` signal, all memory locations are cleared.

When `load_memory` is enabled, a specific instruction word is written to a specified address. The output instruction is read combinationally using the program counter (`pc_output`) as the address. This module mimics an instruction ROM in hardware.

5. data_memory

A 256 x 32-bit synchronous read/write memory. This module simulates the data segment of the processor.

The address for accessing data is taken from the lower 8 bits of the ALU result. The `Memread` and `Memwrite` signals control read and write operations respectively. The `data_reset` signal initializes all locations to zero during reset.
This module supports runtime data loading as well as memory-mapped I/O emulation during simulation.

In both the memory modules, we are using a base block of 32-bits.

6. register_file

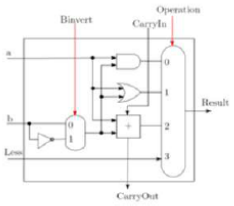
A bank of 32 general-purpose registers, each 32 bits wide. Register 0 is hardwired to zero as per MIPS convention.
The register file supports two read ports and one write port. Writes occur synchronously on the rising edge of the clock and only when the 'reg_write' signal is asserted. Reads are combinational and give immediate output based on input addresses.

7. alu_i and alu_msb

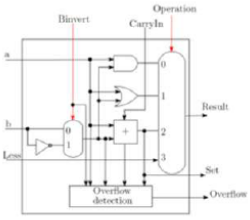
These modules implement 1-bit ALU slices. The 'alu_i' is used for bits 0-30 and handles operations like AND, OR, XOR and basic arithmetic. The 'alu_msb' handles the most significant bit operation and includes logic for overflow detection
and SLT (set-less-than) evaluation. These instantiations of modules are connected in a ripple-carry configuration to form the full 32-bit ALU.

8. alu

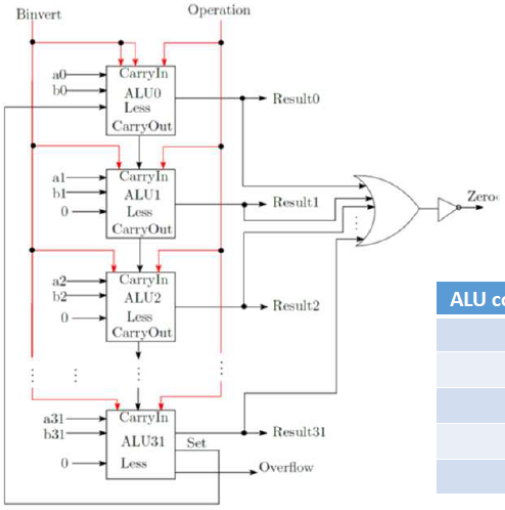
Combines 31 instances of ‘alu_i’ and 1 ‘alu_msb’ to form a 32-bit arithmetic logic unit. The ALU supports multiple logical and arithmetic functions as determined by the 4-bit `alu_control_out` signal. It also produces the ‘zero’ output flag used for branching and an `overflow` flag for signed arithmetic. The design is structurally pipelined using carry chains.

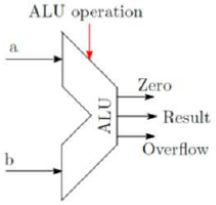


ALU 0 – ALU 30



ALU 31





ALU control lines	Function
000	and
001	or
010	add
110	subtract
111	Set on less than

We are using a separate module for MSB, because we need overflow which we are checking with most significant bits. Overflow tells whether we got the correct sign or not of the add or sub result.

Overflow = (carry_in (XOR) carry_out) of MSB.

It is important to note that carry out of MSB is different from overflow.

And then we give assign ‘less’ in LSB , the ‘set’ of MSB and all other ‘less’ will be default ‘0’, because for slt operation the result is either 0 or 1 , but in 32-bits.

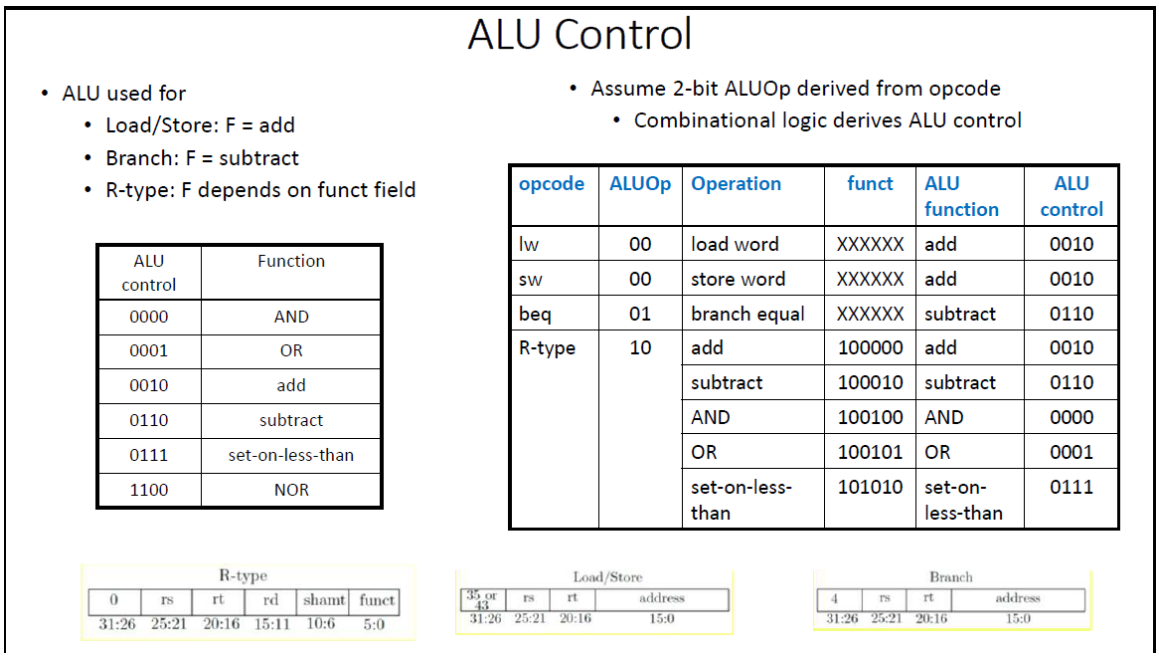
9. control

The main control unit decodes the 6-bit opcode field from the instruction and generates all necessary control signals for the datapath. These include RegDst, ALUSrc, MemtoReg, RegWrite, MemRead, MemWrite, Branch, Jump, Jal and ALUop.
Each instruction type (R, I, J) triggers a specific combination of these signals to drive the datapath correctly.

10. alucontrol

The ALU control unit generates a 4-bit control signal for the ALU based on the function field

of R-type instructions
and the 2-bit ALUop control signal from the main control unit. It enables fine-grained operation selection for instructions
like `add`, `sub`, `and`, `or`, and `slt`.



11. program_counter

An 8-bit program counter that holds the current instruction address. It updates on every clock cycle. Supports three types of updates:
sequential (PC + 1), branch (PC + offset), and jump (direct target). The priority order is: reset > jump > branch > sequential.
It ensures correct instruction flow across conditional and unconditional transfers.

12. full_adder_8bit

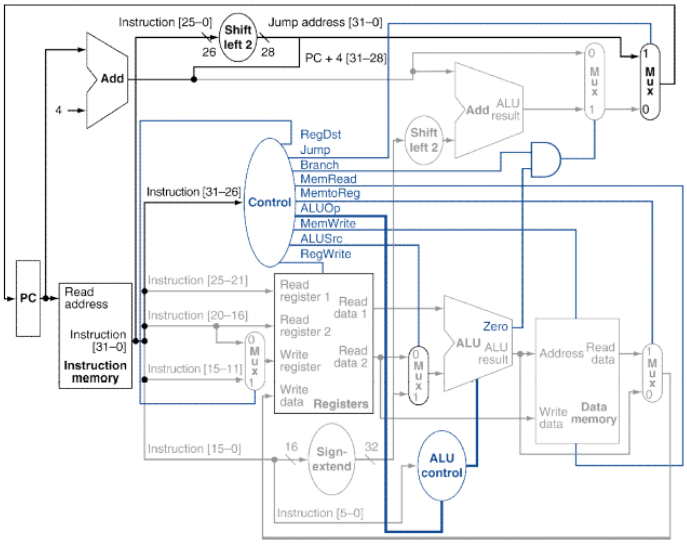
Implements an 8-bit ripple-carry adder used primarily for incrementing the PC and computing branch targets.
This module consists of simple XOR and AND gates chained to propagate the carry bit, simulating realistic adder delays and logic.

13. finall module

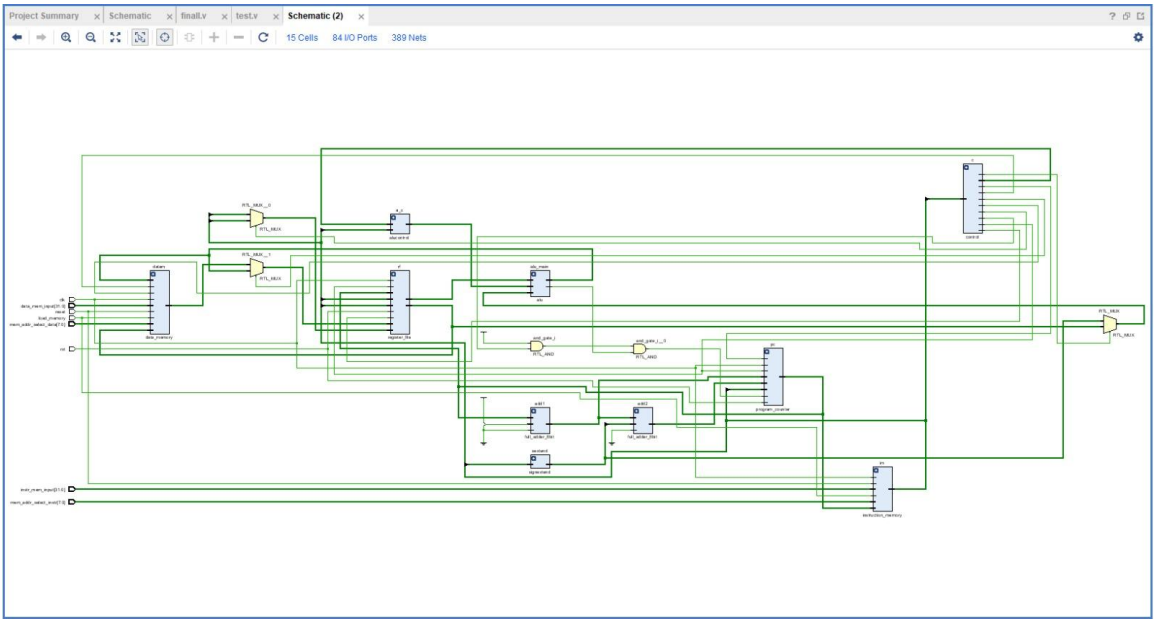
This is the main processor module that ties together all submodules into a single cohesive

unit. It handles the five classical
stages of instruction execution: Fetch, Decode, Execute, Memory Access, and Write Back. The datapath is driven entirely by the
control signals generated by the `control` and `alucontrol` modules. All updates are synchronous, and no behavioral constructs
are used. It supports instruction loading, memory initialization, and execution of R-type, I-type, and J-type instructions.

MIPS processor:



Schematic diagram:



Simulation diagram: