Computer Architecture Project 1 Report

Ousswa Chouchane ID: 900225937
Michael Henein ID: 900212900

March 28, 2024

1 Schematics

2 Modules Description

2.1 Top Module

This is the main module that constitutes the CPU. It integrates various components of a processor. It orchestrates the flow of data and control signals between these components to execute instructions fetched from memory. The ALU performs arithmetic and logic operations, while the control unit manages the control signals based on the instruction opcode.

Inputs:

- `clk`: Clock signal
- `rst`: Reset signal

Modules Used:

- `N_bitReg.v`: N-bit register module
- `InstMem.v`: Instruction memory module
- `regFile.v`: Register file module
- `immGen.v`: Immediate generator module
- `NbitMUX.v`: N-bit multiplexer module
- `NbitALU.v`: N-bit ALU module
- `DataMem.v`: Data memory module
- `ctrl_unit.v`: Control unit module
- `nBitShifter.v`: N-bit shifter module
- `alu_ctrl.v`: ALU control module
- `Branch_Unit.v`: Branch unit module

2.2 DataMem

This module simulates a memory array with 64 entries, each capable of storing 32-bit data.

Inputs:

- `clk`: Clock signal
- `MemRead`: Memory read control signal
- `MemWrite`: Memory write control signal
- `inst`: Instruction opcode (3 bits)
- `addr`: Memory address (5 bits)
- `data_in`: Data to be written to memory (32 bits)

Outputs:

- `data_out`: Data read from memory (32 bits)

Functionality:

- **Storing Functionality:** Upon a positive clock edge (`posedge clk`), if the `MemWrite` signal is asserted, the module writes data to memory based on the provided instruction opcode (`inst`) and address (`addr`).
- **Reading:** The data read from the memory is assigned to the output `data_out`.

2.3 InstMem

The instruction memory unit within a processor.

Inputs:

- `addr`: Address for instruction memory access (5 bits)

Outputs:

- `data_out`: Data output from instruction memory (32 bits)

Functionality:

- Upon a request for an instruction at a specific address (`addr`), the module retrieves the corresponding instruction from the memory array.
- The retrieved instruction is assigned to the output `data_out`.

Testing Case Instructions: [Add the testing case instructions here]

3 Alu_ctrl

ALU control unit within a processor. It takes as inputs various control signals and instruction opcodes to generate the appropriate ALU control signal (`alusel`).

Inputs:

- `aluop`: ALU operation code (2 bits)
- `inst1`: Instruction opcode bits [2:0] (3 bits)
- `inst2`: Additional instruction bit (1 bit)
- `lui`: Load upper immediate signal

Outputs:

- `alusel`: ALU control signal (4 bits)

The `aluop` input specifies the ALU operation type, while `inst1`, `inst2`, and `lui` provide additional information about the instruction being executed. Consider the following scenario for the I instruction:

```
3'b110: begin
           case (inst1)
             3'b000: begin
                 alusel = 4'b0010;/*addi*/
             end
             3'b010: begin
               alusel = 4'b0111; /* slti*/
             end
             3'b100: begin /*xori*/
               alusel = 4'b1001;
             end
             3'b110: begin /*ori*/
               alusel = 4'b0001;
             end
             3'b111: begin /*andi*/
               alusel = 4'b0000;
             end
             3'b001: begin /*shift left i */
               alusel = 4'b0100;
             end
             3'b101: begin /*shift left i instructions*/
               alusel = 4'b0101;
             end
           endcase
```

This belongs to a bigger case statement that takes into account `aluop` and `lui`.

4 Cntrl_unit

The module takes the instruction opcode (`inst`) as input and generates various control signals to coordinate the operation of other modules within the processor.

Inputs:

- `inst`: Instruction opcode (5 bits)

Outputs:

- `Branch`: Branch control signal
- `MemRead`: Memory read control signal
- `MemtoReg`: Memory to register data transfer control signal
- `MemWrite`: Memory write control signal
- `ALUSrc`: ALU source control signal
- `RegWrite`: Register write control signal
- `lui`: Load upper immediate control signal
- `ALUOp`: ALU operation code (2 bits)

Control Logic: The module uses a case statement to evaluate the instruction opcode and set the appropriate control signals accordingly.

ALU Operation Code: The `ALUOp` output determines the operation to be performed by the ALU based on the instruction type. It is set according to the ALU operation required for the instruction.

Load Upper Immediate (lui) Signal: The `lui` signal is set to indicate whether the instruction is a load upper immediate instruction (`lui`) or not.

Example Cases:

- For R-type instructions (`inst = 5'b01100`), only the `RegWrite` control signal is asserted to enable register write.
- For load instructions (`inst = 5'b00000`), memory read, memory to register transfer, ALU source, and register write control signals are asserted to facilitate data loading from memory to a register.

5 Regfile

The module stores 32 registers, each capable of holding a 32-bit value. The module facilitates reading data from registers (`r1` and `r2` outputs) and writing data to registers based on control signals and instruction opcodes.

Some changes have been made to accommodate for the LOAD instructions, they go as follows:

- Standard write operation for regular write instructions (`3'b010`).
- Load byte with sign extension (`3'b000`).
- Load halfword with sign extension (`3'b001`).
- Load byte unsigned (`3'b100`).
- Load halfword unsigned (`3'b101`).

6 Branch_Unit

This module evaluates the condition flags (`cf`, `zf`, `vf`, `sf`) and the branch control signal (`Branch`) based on the instruction opcode (`inst`) to determine whether a branch should be taken.

Branch Logic: The module uses a case statement to evaluate the instruction opcode and set the flag output accordingly. Each case corresponds to a specific branch instruction type.

Flag Calculation: Based on the instruction opcode, the module calculates the value of the flag output using condition flags and the branch control signal.

Handling Default Case: A default case is included to handle unexpected values of the instruction opcode. In such cases, the flag output is set to 0.

Example Cases:

- For a branch equal instruction (`inst = 3'b000`), the flag output is set to the logical AND of the zero flag (`zf`) and the branch control signal (`Branch`). This indicates that the branch should be taken if the zero flag is asserted.
- For a branch not equal instruction (`inst = 3'b001`), the flag output is set to the logical AND of the complement of the zero flag (`zf`) and the branch control signal (`Branch`). This indicates that the branch should be taken if the zero flag is not asserted.

7 Shifter module

```
 This is the same module provided in the source files.  However, one
more case have been added to accommodate for the LUI instruction
2'b11:  r = {16'b0, a[31:16]}; // LUI: Load Upper Immediate (Extract
upper 16 bits)
```

8 Remarks

The implementation of the jump instruction could not be fixed in time, which is why you will notice that it is missing from both the diagram and the implementation; however, we are planning to implement them closely in the branching.

We will have a control coming out from the control unit and anded with the branching getting out of the branching unit, it should give us the right destination of PC.

We have also met some issues with implementing AUIPC, mainly adding hardware and accommodation to other previously constructed functions. This function implementation is also not included in both the diagram and the source files.

Edge cases have not been tested. Further enhancement to the project will be done before MS3.