The American University in Cairo

School of Sciences and Engineering

CSCE 3301-02 Computer Architecture

Project 1: femtoRV32 RISC-V FPGA Implementation and Testing

Milestone 2

Submitted By:

Dalia Elnagar	ID: 900191234
Kirolos Mikhail	ID: 900191250
Kareem A. Mohammed Talaat	ID: 900192903

Submitted Under the Supervision of Dr:

Cherif Salama

Date: 2nd of November 2021

Table of Contents

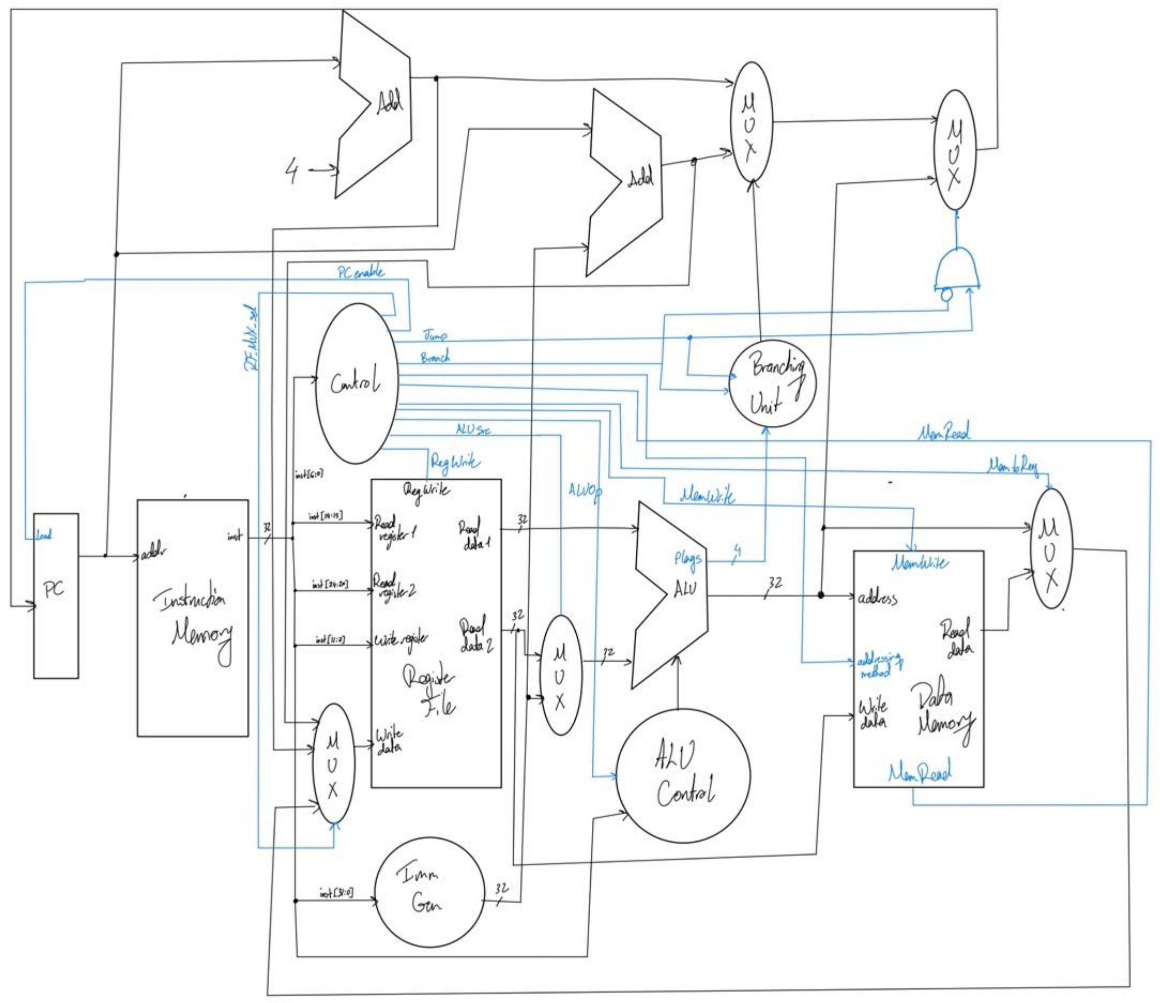
Milestone Description

In this milestone, we have implemented all of the 40 RV32I instructions.

GitHub link

https://github.com/daliawk/RISCV_Processor.git

Schematic



Modules Description

PC

Inputs and outputs

rst: a reset button to reset the value of the program counter to 0

pc_en: an enabler which enables the program count register to get incremented or decremented or obtain its current value in case we need to halt the program.

clk: the system's global clock.

PC_input: the program counter input which is the value we desire the program counter to be next (for example, for a normal instruction (no jumps or anything) should be the old PC value + 4). This input is connected to the output of an MUX such that if the control branch signal is zero and the jump signal is one, then the PC is updated with the output of the ALU (for JALR instructions).

Inst_read_address: the output of the program counter which is the instruction address to be used to fetch the instruction 32 bits.

Purpose

The program counter component is used to store the program counter values, and depending on the current program counter value, the instruction address is produced as an output.

Instruction Memory

Inputs and outputs

Inst_read_address[7:2]: the 6 bits input which the component uses to fetch the instruction from the memory.

inst: the 32 bit instruction which is the fetched instruction from the memory produced as an output.

Purpose

The instruction memory holds all the instructions in the memory, and depending on the input address, an instruction is fetched and produced as an output

Control Unit

Inputs and outputs

rst: a reset button to reset all the values to 0

inst: the 32 bit instruction to be used an input.

pc_en: an enabler which enables the program count register to get incremented or decremented or obtain its current value in case we do not want to load the next instruction produced as an output.

Branch: a boolen enabled when the instruction being currently executed is a branching instruction.

Jump: a boolen which is enabled when the instruction being currently executed is a jumping instruction (jal, jalr).

Mem_read: a boolen which is enabled when the instruction being currently executed requires a memory read (lw for example)

Mem_to_reg: a boolen which is enabled when the instruction being currently executed requires saving a value from the memory to the register file (add for example).

Mem_write: a boolen which is enabled when the instruction being executed requires writing into the memory.

ALU_Src: a select which is used to select the input value into the ALU.

Reg_write: a boolen which is enabled when writing to the register file is requires.

Signed_inst: a boolen which is enabled when the instruction being executed is a signed instruction.

AU_inst_sel: 2 bit selection used to select the type of the loading or storing instruction (LH, LB, LBU, SW, etc…)

ALUOp: 2 bit selection used to select the type of operation the ALU is going to be executed.

RF_MUX_sel: a 2 bit selection used to select the value to be used as the 32 bits value to be written into the register file (write_data) based on the instruction at hand.

Purpose

The control unit is using the instruction bits to identify the instruction at hand, and therefore produce controlling outputs (like selects, branch, jump, etc) which regulate the other components depending on the instruction at hand.

Register File

Inputs and outputs

rst: a reset button to reset the values of the registers to 0

clk: the system's global clock.

read_reg1: the read register 1 address from the instruction (5 bits from bit 15 to bit 19 in the instruction)

read_reg2: the read register 2 address from the instruction (5 bits from bit 20 to bit 24 in the instruction)

write_reg: the destination register address from the instruction (5 bits from bit 7 to bit 11 in the instruction)

write_data: the 32 bits register value to write into register file in the destination register address. The input to this register is the output of a 3x1 MUX to support R-instructions, AUIPC, and JAL/JALR

reg_write: the 1 bit select which enables/disables writing in the register file, specifically in the destination register.

read_data1: the 32 bits register value produced as an output from the register file which represents the first source register in the instruction.

read_data2: the 32 bits register value produced as an output from the register file which represents the second source register in the instruction.

Purpose

The register file is used to store the registers values, and there can be multiple operations executed on it:

We can read registers values based on the input address.

We can modify the register value (the write_address is used to identify that register), and write_data is the value to be written into the register, and all of this is enabled by the reg_write, so if it is 1 then we can change that register's value, otherwise we cannot edit it.

Immediate generator

Inputs and outputs

IR (inst): the 32 bit instruction as an input

Gen_out (immediate): the immediate value produced depending on the instruction.

Purpose

The immediate generator is used to generate the immediate value based on the instruction to be used later in other components depending on the instruction at hand, for example, in a beq instruction, the immediate generator is used to generate the immediate value from the instruction bits, then shift it to the left (multiply by 2) to be the offset to be added to the current program counter to represent the new program counter value.

The ALU Control Unit

Inputs and Outputs

ALUOp: The ALUOp is the operation determined by the control unit and is passed to the ALU control unit to decide the instruction format.

Inst: This is the full instruction, which used by the ALU Control unit to decide the ALU operation. (Mainly function three, from bits 12 to 14 and bit 30 are used)

ALU_selection: This output decides the operation to be done by the ALU.

Purpose

The ALU Control unit tells the ALU what to do in terms of operations, which can be narrowed down to a group of simple operations. It takes the ALUOp variable from the Control Unit, which determines the type of the instruction passed to it. Mainly, the I and S type and the Jalr instruction are grouped under (00), the branching and the Jal instruction under (01), the R-format instructions under (10) and the AUIPC and LUI under (11) (Later on where removed from the ALU completely).

Depending on the ALUOp, the Alu Control Unit checks function three and bit number 30, if needed. Even though there will be no other cases than the 4 ALUOp cases, a default case is added to pass ALU ADD.

The ALU

Inputs and Outputs

A and B: These inputs are the inputs the operation is going to be done on. A is just the read data one from the register file and B is the result of the multiplexing of read data 2 and the output of the immediate generator.

alu_control: This input is the control produced by the Alu Control unit and it decides the operation of the ALU.

ALUout: This output is the result of the operations done on A and B based on the ALU Control.

Z, V, C, S: These are the flags mainly used by the branching unit to decide on whether to branch or not.

Purpose

This component takes three main inputs and gives out just a single output.

The ALU takes the two operands; the read data one and the result of the multiplexing of the read data two and the immediate values generated from the immediate generator. Regardless of the type of instruction passed to the ALU, the same operation is done, and the only difference is the second input (whether the immediate generator output or the read data two. That is because in all cases, the addition process is the same whether it is an immediate function or addition of registers, so from the perspective of the ALU it is the same instruction.

Branching Unit

Inputs and Outputs

B and jump: These are the flags set by the Control Unit based on the instructions sent to the control unit.

Z, C, V and S: These are the flags calculated inside the ALU and are used to decide whether to branch or not.

Funct3: These are the three bits of function 3 and are used to know which branching condition should we look for and how to handle them.

Branch: This is a one-bit output and it is only set to one if the jump bit is one or if the branching condition is true.

Purpose

This unit decides on whether to branch, jump or neither. It does so by checking function 3 and based on it checks the flags set for said branching condition. If the jump flag is one, then automatically the branch bit is one. To sum up, if jump is one or if function 3 and the condition opposite to it is true, then the branching bit is true. In any other case, branching is set to zero.

Data Memory

Inputs and Outputs

mem_read: This input signal to read from memory

mem_write: This input signal to write to memory

AU_inst_sel: The 2-bit input signal that decides the addressing method (word, half-word or byte)

signed_inst: This input signal to signify that whether the addressing mode is signed or unsigned

addr: The input address in the memory

data_in: The 32 bit input data to be written to the memory

data_out: The 32 bit output data to be read from the memory

Purpose

This component has an array of bytes as the data memory of the processor. It does 2 main operations:

1- Reads from memory by setting the data_out to be the concatenation of a group of 32 bits which differ according to the load instruction. LW loads the concatenation of 4 bytes. LH loads the concatenation of 2 bytes while sign-extending the most significant bit of the byte with the bigger address. LHU concatenates the 2 bytes and fills the uppermost 16 bits with zeros. LB and LBU are the same as LH and LHU with the difference being loading 1 byte.

2- Writes to memory by setting the concatenation of some bytes in the memory array to be either all or some bytes from the 32 bit data_in. The number of bytes concatenated is decided according to the AU_inst_sel