B09902053
何碩宸

CA HW3 Report

Computer Architecture
November 7, 2022

Module Explanation

- MUX32.v: MUX32 module reads data1, data2, and a selector as input, and outputs the data chosen by the selector. It is implemented with a ternary operator.
- Sign_Extend.v: Sign_Extend module reads 12-bit data as input and outputs 32-bit data. The module achieves sign extension by copying the 11^{th} bit input data as the remaining 20-bit prefix for the output 32-bit data.

funct7	rs2	rs1	funct3	rd	opcode	function
0000000	rs2	rs1	111	rd	0110011	and
0000000	rs2	rs1	100	rd	0110011	xor
0000000	rs2	rs1	001	rd	0110011	sll
0000000	rs2	rs1	000	rd	0110011	add
0100000	rs2	rs1	000	rd	0110011	sub
0000001	rs2	rs1	000	rd	0110011	mul
imm[11:0]		rs1	000	rd	0010011	addi
0100000	imm[4:0]	rs1	101	rd	0010011	srai

- Control.v: Control module takes the 7-bit opcode as input, and outputs 2-bit ALUOp, ALUSrc, RegWrite. From the instructions given in the spec, there are characteristics in those instructions:
  1. All the instructions above need to write the data back to the ***rd*** register, so RegWrite is always set to 1.
  2. Only ***addi*** and ***srai*** apply immediate value, and their Op[5] is different from others. Therefore, ALUSrc is set to be $\sim$ Op[5].
  3. ***mul*** and ***addi*** may share the same value in funct7|funct3, so it is necessary to pass the Op[5] value down. For convenience, simply choose Op[6:5] as ALUOp.
- ALU_Control.v: ALU_Control module takes funct7|funct3 and 2-bit ALUOp as input, and outputs 3-bit ALU Control. For most instructions, the operator can be identified with funct3 solely, so funct3 can simply be the ALUCtrl for most cases. However, ***add***, ***addi***, ***sub*** and ***mul*** share the same funct3. Therefore, it is necessary to take funct[8], funct[3] and ALUOp[0] (i.e. Op[5]) into consideration when the funct3 is 000. ALUControl identifies ***sub*** and ***mul*** and assigns special ALUOp that is not identical to any funct3 in the above chart, while ALUCtrl for ***add*** remains 000.

- ALU.v: ALU module takes data1, data2, ALUCtrl as input and outputs data, Zero. It identifies the operator with ALUCtrl and outputs the arithmetic result to data. Zero is not used in this assignment, so I don't use any operation on Zero.
- Adder.v: Adder module takes data1, data2 as input and output the result of data1 + data2. It is used to add PC. In this assignment, data1 is the old PC and data2 is 4 because there is no branch/jump operation.

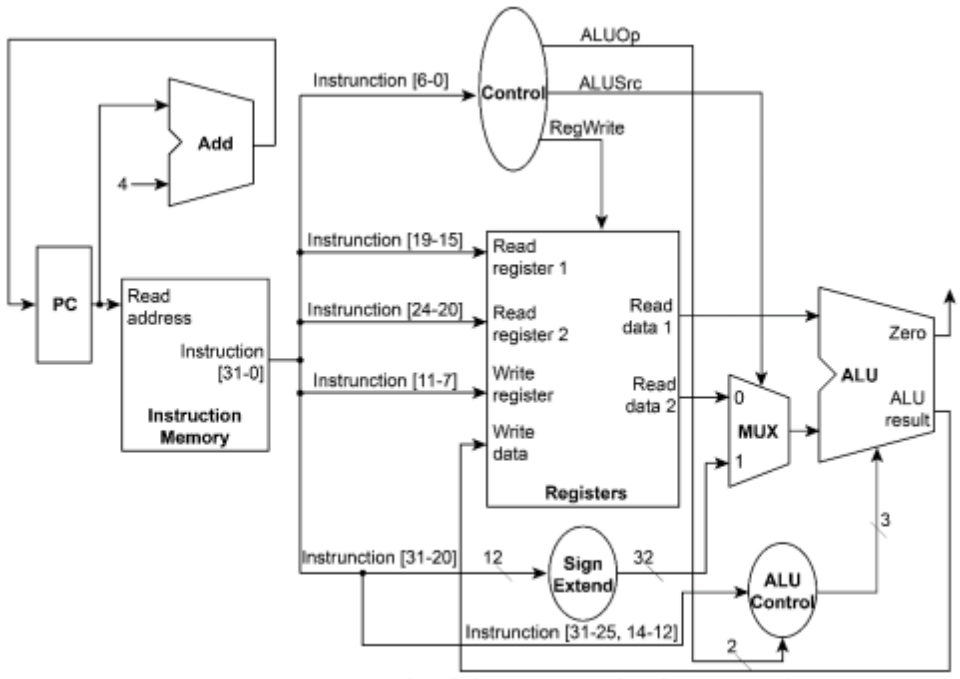

Figure 1 Data path of the CPU in this homework

- CPU.v: CPU module is the core module in this assignment. It integrates different components and specifies data flows to be the above diagram. CPU module reads clock signals, reset bit, start bit and sends those to PC module. PC module then determines the current PC and it is sent to instruction memory to fetch the instruction. Then the instruction is performed and the PC is updated. It will do this over and over to execute the instructions sequentially.

Development Environment

- OS: Ubuntu20.04
- Compiler: iverilog