The Chinese University of Hong Kong, Shenzhen

CSC 3050

Computer Architecture

Project 3: Implementing MIPS Arithemetic Logic Unit (ALU) using Verilog

Author:
Qiu Weilun

Student Number:
120090771

April 7, 2022

Contents

1 Big Picture of ALU **2**

2 Underlying Logic of the Implementation **3**

3 Running the Program **4**

1 Big Picture of ALU

The Arithemetic Logic Unit (ALU) is an important computation component in the Central Processing Unit (CPU) of a computer. Genrally speaking, ALU is used to perform calculation during the computer program is running. For a MIPS ALU, six basic arithemetic operation is supported. The operations and their correponding ALU control signals are shown in the following figure.

Operation	**Control Signal**
and	0000
or	0001
add	0010
subtract	0110
set-on-less-than	0111
nor	1100

Table 1: Basic Functions of MIPS ALU and the Control Signals

Other instructions in MIPS architecture can be implemented in the ALU based on this six basic instructions. The input of an ALU would be the values stored in the two corresponding registers. The output of the ALU would be the result of the calculation as well as values of the three flags. The zero flag is used to indicates the case when the final result is zero. The negative flag is used to indicate that the result is a negative number. The overflow flag is ued when overflow issue occurs in the calculation.According to the requirement of this project, the overflow flag will be used only in **add**, **addi** and **sub** instruction. The zero flag will only be used in the **beq** and **bne** instruction. The negative flag will be used in **slt**, **slti**, **sltiu** and **sltu** instruction.

The major task of this project is to simulate the process of parsing the instruction, fetching the memory and calculating the result and status of flags. The program would first divide the entire instruction into several segments. From these machine code segments, the program is able to recognize the type of the instruction and the registers used in the operation. The values in the corresponding registers will be fatched from the memory to the input of ALU. The ALU will then carry out the specific operation based on the ALU operation signal.

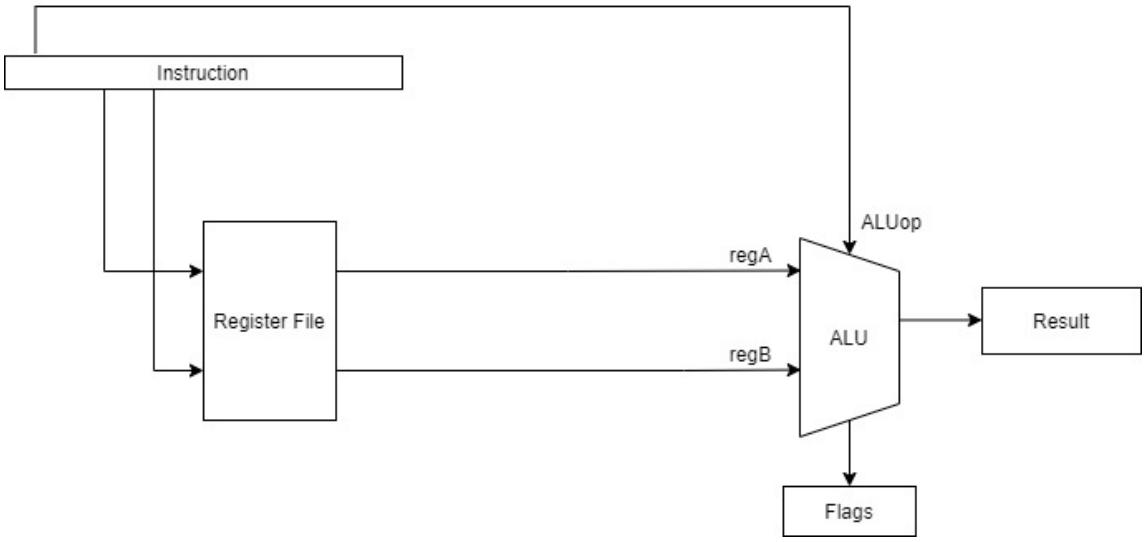

Figure 1: The Process of Instruction Parsing, Register Fetching and Arithemetic Operation

In this project, not all the MIPS instructions are implemented. The implemented instructions are shown in the following table.

Instruction Type	Instructions
R-type	add, addu, and, nor, or, sll, sllv, slt, sltu, sra, srav, srl, srlv, sub, subu, xor
I-type	addi, addiu, beq, bne, lw, ori, slti, sltiu, sw, xori

Table 2: Instructions Implemented in the Project

2 Underlying Logic of the Implementation

To achieve the goal of identifying each instruction, a case statement in verilog is used. The program first checks whether the operation code of the instruction is 000000. If it is, that means the instruction is a R-type instruction. Therefore, to determine the corresponding function of the instruction, the function code of the instruction is needed. If it is not, the instruction is an I-type instruction. In this case, the addresses of the rs and rt registers can be directly accessed from the instruction. The immediate field can also be found. For zero extension or sign extension of the immediate field, it will be determined by the instruction itself. The following data flow chart exactly shows how the entire program work.

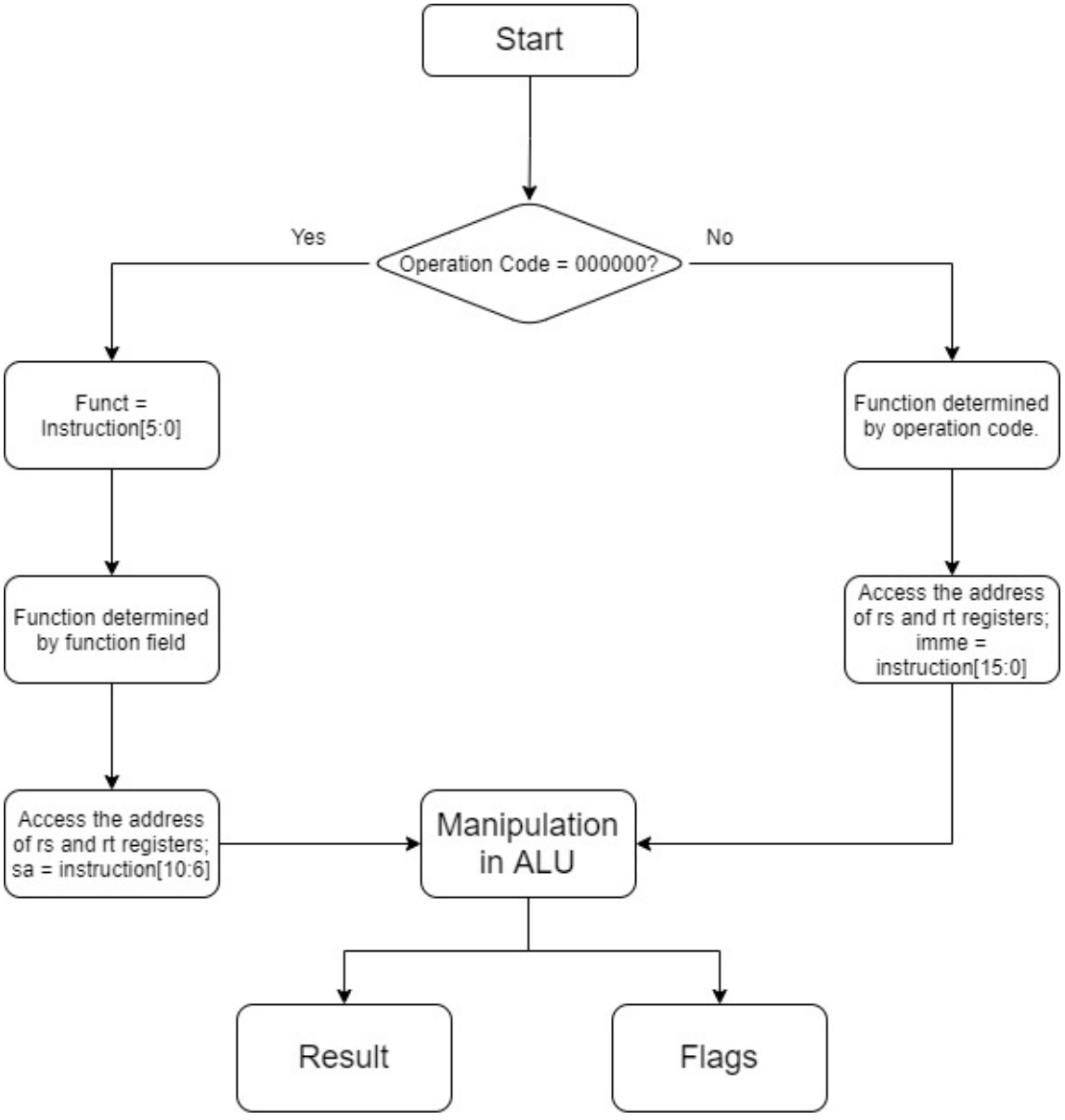

Figure 2: Data Flow Chart of the Implementation

3 Running the Program

To execute the program, enter the ”make” instruction in the terminal first to compile the ALU.v file and the test_ALU.v file. In the ALU.v file, the core operations in the ALU is implemented. The ALU test bench is written in the test_ALU.v file. After that, enter the ”vvp test_ALU.vvp” command to execute the program.