Computer Architecture - Project

1st Juan Galvez
Universidad de Ingenieria y Tecnologia
UTEC
Lima, Peru
juan.galvez@utec.edu.pe

***Abstract*—This article will be about the final project of the computer architecture course. As part of the project, we were commissioned to make a report detailing all the steps that were taken to carry out the project. The main task was to create a Datapath or processor that complies with certain instructions. The MIPS architecture will be used and finally the Datapath will be programmed in verilog (HDL).**

***Index Terms*—Datapath, Mips, Risc, Verilog, CPU, ISA**

I. Introduction

First we need to know that it is a Datapath. MIPS-Datapath is a graphical MIPS CPU simulator. The program is intended to be used as a teaching aid for computer architecture courses involving MIPS. One that we know is a Datapath, we need to know with what language we are going to program it. It is usually programmed in Verilog, Verilog is a hardware description language and supports the design, testing and implementation of analog, digital and mixed signal circuits at different levels of abstraction. It is not very difficult to understand since it has a syntax very similar to C. For this project Icarus was used, which is a verilog implementation for linux / unix operating systems.

II. Methodology

To know that our instructions work correctly, we had to investigate what each instruction did, that is why we reviewed books, videos and research articles on the subject. Most of the information was taken by the webpage MIPS Instruction Reference. It was a good source of information although it did not have data on some instructions, so it was sought from other sources such as the website MIPS 101. These two sources of information complemented each other very well, so many other sources of information were not needed. We use as device a MacbookPro with macOS Mojave The information collection process took about two days, although apparently it was not enough time (See "Experimental Setup").

III. Experimental Setup

All the instructions were tested, first of all, the results of some instructions were not the expected ones, that's why we had to investigate more thoroughly what specific things each instruction needed to be able to work. After several modifications and tests, the instructions worked correctly (the results were as expected). One of the problems was that, the Datapath optimization had to be lowered because more hardware was required to satisfy the needs (more parameters) that our instructions demanded for its proper functioning. We will further specify how it was done to resolve the unexpected results.

IV. Evaluation

Three programs were created to test the instructions, the first program performs: Add, Sub, And, Nor, Or, Slt, Addi, Subbi, Andi, Ori, Slt. The second: Lb, Lh, Lw, Sb, Sh, Sw, Lui. The third: Beq, Bneq, Bgez, J, Jr, Jal.

We use in total 10 modules (not including the testbench), these modules are:

1) PC: The PC is were we determine what instruction we are going to execute, also is where jumps and branches instructions are performed. It has many inputs signals. When a cycle is completed the we add 1 to the PC, thats how all the instructions are executed. His output will be the number of the instruction to be executed. For example if the output is 10, we are going to execute the instruction 10.

The PC of a "normal" Datapath uses PC + 4 because the memories are byte-addressable. In this project we use PC + 1, that could be a big mistake in terms of optimization, but if we only want to perform the instructions, we won't had so many problems since the main task of the problem is not to evaluate performance, is to evaluate/test if all the instructions work correctly without bugs/errors.

2) InsMem: In the Instruction Memory we receive the pc signal of the PC, the pc signal decides what instruction we are going to execute.

3) Control: The Control receives as inputs the bits [3:0] and [31:26] of the instruction that its being executed. Theses bits decide what outputs are going to be 1 or 0. There are 10 outputs: RegDst, Jump, Branch, MemRead, MemtoReg, AluOp, MemWrite, Alusrc, RegWrite, Jal.

4) Mux1: The Mux1 recieves as inputs RegDst, the bits [20:16] and [15:11] of the instruction that its being executed. If RegDst = 1, the ouput signal will be the bits [15:11] otherwise the output will be the this [20:16].

5) Regfile: The Regfile recieves as inputs RegWrite, JAL, the bits [25:21] of the instruction thats its beign executed, the output from Mux1, the output from Mux3 and the pc signal. Here is where the values are charged and stored. Depending on the inputs, the outputs will be to values.

6) Mux2: The Mux2 has as inputs Alusrc, one output of the Regfile and the bits [15:0] of the actual instruction previously sign-extended. If Alusrc = 0, the output will be the "one"

output of the Regfile otherwise the output will be the bits [15:0].

7) Alu Control: Alu Control receives as inputs Aluop and the bits [5:0] of the instruction that its being executed. The output will be the signal that determines the type of operation to do.

8) Alu: The Alu has the output of Mux2, the "other" output of the Regfile and the output of the Alu Control as inputs. In the Alu, most of the instructions are executed. His output is the result of the operation performed.

9) DataMem: The DataMem recives as inputs the output of the Alu, the "one" output of the Regfile, MemRead, MemWrite and the bits [31:26] of the actual instruction. The DataMem is similar to the Regfile because you can charged and store values of extra registers. If MemRead = 1 we are going to charge the value of an "extra" register (of DataMem) and put it to a register of the Regfile, if MemWrite = 1 we are going to store the value of a register of the Regfile to a register of the DataMem. His output will be the value of one of his registers. from the DataMem

10) Mux3: Mux3 has as inputs the output of the DataMem, the output of the Alu and MemtoReg. If MemtoReg = 1, his output will be the output of the Data and if MemtoReg = 0, the output will be the output of the Alu.

A. Add, Sub, And , Nor, Or, Slt, Addi, Subbi, Andi, Ori, Slti

These instructions were the easiest to do, they did not present any problems except SUB and SUBBI since we had to find a way to express the result of the operation in binary if this is negative, and that the program differentiates when to make the change of 1 and 0 (one of the ways of converting a positive number to a negative is changing the 1 with the 0). Apart from that, there were no more problems.

B. Lb, Lh, Lw, Sb, Sh, Sw, Lui

For these instructions (except Lui) one more parameter had to be passed to the DataMem module to differentiate between Lb, Lh, Lw and Sb, Sh, Sw. As all the load‘s and store‘s are very similar, their opcodes were used as parameters, so they could differentiate between them. For Lui there was no problem at the time of implementing it.

C. Beq, Bneq, Bgez, J, Jr, Jal

These instructions were the most difficult since some arrangements were made for the modules. The modules that were edited were Control, PC and Regfile. Control and PC were edited so that Jr could work, to each one, more parameters were added. Regfile was edited for Jal to work because we needed to save the value of our "jump" in register 31 or better known as "ra".

V. Conclusions

In conclusion, it can be affirmed that the main objective of the project was fulfilled. However, it is not recommended to implement this Datapath for large evaluations and / or tasks since a lot of hardware would be spent implementing them (difficult tasks). In addition, some variants of the "normal" Datapath were made, increasing the number of modules caused the use of hardware to increase which would lead to a decrease in optimization. If it is only used to perform simple operations like the ones we were commissioned, this project could be used. If we would try to modify our Datapath so that it becomes a pipeline, we would have several problems because a pipeline seeks to optimize the Datapath, since our Datapath is not highly optimized, many changes would have to be made in it. For this case, it would be better to use the common Datapath.

VI. Comments

Throughout the process of programming the Datapath several difficulties were encountered, such as the implementation of more modules to perform an instruction or pass more parameters to the different modules, for example, so that the load byte, halfword and store byte can be performed, halfword, one more parameter had to be passed to the DataMemory to be able to differentiate between lw, lb, lh or sw, sb, sh. Like this there were several cases where more parameters had to be passed to the other modules. This was a problem because when doing this type of "adjustment" the use of hardware increases and that was not the idea since the Datapath was also sought to be as optimized as possible, but it was not possible to "optimize" it at all. With respect to the report, the implementation of LATEX should have been made optional because not all of us feel weary using it.