Alejandro A. Sánchez 804-10-7908

CCOM 4086 Computer Architecture

***16 Bit CPU***

***CPU Components:***

This CPU contains five different circuits: The CPU, the register file, The control unit, the ALU and a separate circuit for the ldhi instruction. The contents of this 16-bit will be discussed below.

**Datapath:**

The datapath is the component that brings every other component of the CPU together. It contains two RAM memories. The first RAM contains the instructions and the second is the memory that will be used to store information or read information, with the lw and sw instructions. The datapath also contains a Program Counter (PC). This special register is used to fetch the next instruction. To update the program counter we simply add one to the value of the PC. However if a branch or jump instruction occurs something different will happen to the PC. If a branch instruction occurs, the ALU will calculate the new address for the PC and this address will be summed to the PC instead of a 1. If a jump instruction occurs, the 1 will be ignored and the PC will move to whichever address was given by the jump instruction.

To implement our CPU, eleven multiplexers were used. The first one, from left to right, determines if a jal instruction is to be performed. If it is, then the control will send the signal to the multiplexor containing a 1. If it is a 1, it will then tell the Register File to write on the first register, if it is a 0, it will write on the register given by the instruction. The second multiplexer are used to distinguish if the first input of the ALU will be given by the rs or the rd register. The third multiplexer is used to determine if the second output for the ALU will come from the rt register or from an immediate value. The fourth multiplexer is used to determine if the data that will be written to the register will come from memory or from the ALU. The fifth multiplexer determines if the data will that will be written into the register will come from the output of the fourth multiplexer or from the jal instruction. The sixth multiplexer does the same thing as the fifth multiplexer, but instead it receives the input from the ldhi instruction. The remaning 5 multiplexers are used for the jump and branch instructions. A multiplexer is used to determine if the instruction is bnz, i for this we used a AND gate which takes as input the *is zero* bit from the ALU and the *branch* bit from the control unit. The second is a bz instruction, for this we also use a AND gate, but this time we negate the zero input. The next multiplexer verifies if it a jump instruction and the final multiplexer adds 1 to the program counter if it is not a jump or branch instruction. All of these instruction use the control unit to determine what to do.

This datapath also contains an ALU, register file and the control unit, which will be explained below. Sign extenders are used, these change the value bits from 8 to 16 bits for the i-type instructions and from 12 to 16 bits for the j-type instructions. A clock is used to simulate a real CPU. At each clock tick a new instruction will be executed. Also a ldhi box is used. This box contains the ldhi instruction, which receives the 8 bit immediate and returns it in the upper bits of a 16 bit value which then will be inserted into a register. Finally a reset button is used to erase the contents of both RAM memories.

**Register File:**

The register file contains the registers that are going to be used in the CPU, in this case 16 registers will be used. The register file can perform two operations on the registers: read and write. To read on the register, we need a demultiplexer or a decoder that will split the data for the 16 registers. Then we can access the data from the register and using a multiplexer which receives either rs, rt or rd as input we can determine from which register we want to read from. To implement write we need three more inputs: the clock, write enable and the data that we want to read. The clock and the write are of 1 bit and the data is of 16 bits. When the write enable and the clock are set to 1, we are going to put the data on the selected register. When either the clock or the write enable are 0, we cannot write on the register. To know this, we use various AND gates.

**Control:**

The Control Unit determines what the CPU will do when it receives an instruction. Depending on the opcode received, the control will determine if the instruction if r-type, i-type or j-type. The following table shows the input received by the control (the Opcode) and the data that will output depending on the opcode.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instr | Op | Jump | Branch | Mem  Read | MemtoReg | Mem  Write | ALU  Src | Reg  Write | rs or rd? | ldhi? | AluOP |
| add | 0000 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 011 |
| sub | 0001 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 100 |
| lw | 0010 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 011 |
| sw | 0011 | 0 | 0 | 0 | x | 1 | 0 | 0 | 0 | 0 | 011 |
| and | 0100 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 000 |
| or | 0101 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 001 |
| not | 0110 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 010 |
| addi | 1000 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 011 |
| ldhi | 1001 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | X |
| bz | 1010 | 0 | 1 | 0 | x | 0 | 0 | 0 | 1 | 0 | 101 |
| bnz | 1011 | 0 | 1 | 0 | x | 0 | 0 | 0 | 1 | 0 | 101 |
| jal | 1101 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | X |
| jr | 1110 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | X |
| j | 1111 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | X |

The Control Unit was implemented in logisim by using a two ROM memories which contain the outputs for each different type of instruction. The first ROM contains the outputs used by the datapath and the second ROM contains the outputs for the ALU that determine which operation it will perform.

**ALU:**

The ALU is the component of the CPU which performs arithmetic and logical operations. This ALU can perform the following operations: AND, OR, NOT, SUM, SUB. To know which of this instructions it has to do, the ALU uses an internal opcode. AND is 000, OR is 001, NOT is 010, SUM is 011, SUB is 100 and Zero is 101. This opcode is given to the ALU by the Control Unit. In our CPU the ALU receives a 3 bit opcode, two data inputs of 16 bits and returns the result and a bit which says if the result is zero or not. To test for zero a 16 bit NOR is needed. Each bit of the result is put on the NOR, and each bit is 0 it will display 1, if any of the bits is 0, it will display 1.

***How it Works:***

To use this CPU an image that contains a program written in hexadecimal must be loaded into the RAM memory that contains the instructions. This memory is called instruction memory. After the program is inserted, we can start the program by clicking on the clock. When the clock is one the next instruction will be fetched from the program counter and executed. To verify if the instructions are doing what they are supposed to do, we can check the wires that contain each data. For example if an R-Type instruction, like ADD comes in, we know that this instruction will need to read data from rs and rt and then send the content of this registers to the ALU and finally put the result in the rd register. To verify that everything is behaving correctly we check if the the RF is receiving the correct inputs and outputting the correct information. Then we verify the control. For an ADD instruction the control is supposed to send a signal for writing to a register, also it must send a sum opcode to the ALU. If the first signal does not turn green, then we have a problem.

To verify and i-type instruction, we must make sure that the instruction is using the 8 bit immediate instead of the rs and rt registers. We must also verify that the sign bit extender if extending the bit correctly. For j-type instructions the only thing that we must do is just go to the desired address. For lw and sw we must make sure that the memory is receiving and displaying the correct information. To verify if the CPU is doing everything correctly we must verify that it displays the fibonacci sequence in memory like this 1 in 0000, 1 in 0001, 2 in 0010, etc. If this is done correctly, then we the CPU is implemented correctly. Once the program has completed we can reset the contents of the memory using the reset button.