Christian A. Rodríguez Encarnación

801-11-6705

CCOM 4086-002

CPU Documentation

For our final project, we implemented a 16 bit CPU with Logisim. I’ll first give a brief overview of my design, and then explain how the instruction set carries out the instructions using my design. First note that the instruction are 16 bits that are broken down into 4 parts, used differently in some instructions.

For the Program Counter I used a simple Logisim register file. This PC is updated after every clock cycle, by adding 1 for most instruction, or by changing to a branch address for branches.

For the instruction memory I used RAM from Logisim, this memory takes as input the PC, and outputs the 16 bits in that location, it is also updated on every clock cycle.

The Register file is an altered version of one implemented by Felipe Torres, also for this project. It contains 16 16-bit registers. It receives 3 memory locations from the instruction memory, the clock, a result that comes from either the data memory, or the ALU operation, and a register write control bit. It outputs the 3 registers in the locations it received.

I implemented the ALU myself. The ALU carries out a 16-bit operation. This ALU receives 4 control bits and carries out 5 different operations depending on that control bit.

0000: OR

0001: AND

0010: add

0011: sub

0100: NOT

My ALU also verifies if the top input is equal to 0. This is used for the branch operations. My ALU has two outputs, the 16 bit operation output, and the 1-bit comparison output.

For Data Memory I also used RAM from Logisim. This RAM has different load/store ports that I use for load and store instructions. It receives as input the ALU operation result and two control bits for load and store instructions. It outputs the data on the memory location indicated by the ALU operation result. It also receives the clock as input and is updated on every clock cycle.

I implemented the Control for my CPU using a ROM from Logisim. The Control receives the 4 instruction bits and read from the ROM the13 bits stored in that location. I wrote all 12 instructions in the ROM, such that it activates the necessary output control bits to carry out each instruction. The Control outputs the following bits:

1. 4 bits for the ALU operation
2. Branch bit (Not Used)
3. Memory Read for load instruction
4. Memory Write for store instruction
5. Memory to Register for load instruction
6. Register Write for writing data into a register
7. Register Destination (Not Used)
8. ALU Source used in different instructions
9. PC Source used for branches

The CPU implementation has the following instructions:

0000 add

0001 sub

0010 lw (load word)

0011 sw (store word)

0100 and

0101 or

0110 not

1000 addi (add immediate)

1010 bz (branch if zero)

1011 bnz (branch if not zero)

1111 j (jump to address)

My CPU has multiplexers that determine the input and output of some of the circuits, in order to carry out each instruction. To explain how my CPU carries out these instructions I divide them into 4 different classes:

1. add, sub, and, or, not
2. addi
3. lw, sw
4. bz, bnz, j

In the first type of instructions the CPU activates the Register Write control bit and the ALU operation control bit varies in each operation. After the instruction is read from memory and decoded, the first 8 bits represent the two registers that are to be operated on, and the next 4 bits the register where we are to store the operation result. The 8 bits are sent to the ALU and the operation result is sent back to the Register File, which writes this data into a register. Adding 1 updates the PC and data memory is not used.

Ex. add $1, $2, $3

The addi instruction is very similar to the first class of instruction, but the ALU Source bit is turned on, such that the ALU receives a 8 to 16 bit sign extended offset value, and the register we are to write the result in. The rest is exactly the same.

Ex. addi $1, 16

For lw and sw instructions the Control activates either the memory read or write bit, the memory to register bit, and the register write bit for load. These instructions calculate the data memory address by adding the first 8 bits in the instruction, then either fetches the data in that address or stores the data from a register in that address.

Ex. lw $1 (0)$4

Branches activate the ALU source and the branch control bits. Each branch also has a different PC source code: 01 for bz, 10 for bnz and 11 for jump. For bz and bnz we compare the first 4 bits to 0 and use the ALU’s if 0 bit. This bit is sent to two different multiplexors, representing both instructions. For bz if the bit is 1 then it sends the address indicated in the instruction to the PC source multiplexor, or else it sends the constant 1 and the PC is updated normally. For bnz it does the exact inverse of bz. Since jumps are always taken, then we just send the 12 to 16 sign extended target address to the PC source. In these three instruction the PC is updated accordingly.

Ex. bz $4, loop