Computer Organización

Iván de Jesús Deras Tábora

March 13, 2019

Period 1 2019

MIPS32SOC Part 4

Objetives

In this part of the project we will add suport for more branch, bit shift and procedure management instructions. We will add the following instructions: `bgez`, `bgtz`, `blez`, `bltz`, `nop`, `sll`, `srl`, `sllv`, `srlv`, `sra`, `srav`, `jal`, `jr`

What you will need

In order to developed this part of the project you'll need the following:

1. The third part of the project completed.
2. The test cases

The last item will be provided to you.

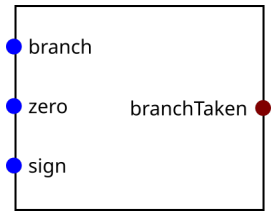
Your task

To achieve your goal you'll have to do the following

1. Add a module called Branch Resolver, this will be developed in Verilog.
2. Add support in the ALU for shift left and shift right.
3. Add some more control signals
4. Add support for procedure management instructions (`jal`, `jr`)
5. Change the `RESET` signal to use inverted logic (**Active Low**). That is a `0` will reset the system and a `1` will keep the system running.

The Branch Resolver

The branch resolver circuit will decide, based on the branch type if taken or not. The circuit has the following diagram:



Signal	Type	Description	Bits
branch	Input	Indicates the type of branch in execution. 0 = No branch, 1 = beq, 2 =bne, 3 = bgez, 4 = bgtz, 5 = blez, 6=bltz	3
zero	Input	Zero flag	1
sign	Input	Sign flag	1
branchTaken	Output	Branch taken. 1 if the branch is taken, 0 otherwise.	1

All the instructions `bgez, bgtz, blez, bltz` assumes a signed number as an input, so to implement them you just have to look at the sign of the input value. You DON'T need to change the ALU to execute this instructions.

Bit shifting instructions

The bit shift instructions to add are `sll, srl, sllv, srlv, sra, srav`. For these instructions, you have to add 3 operations to the ALU, shift left, shift right and shift right arithmethic (in verilog use the operator `>>>` for this). Take into account that shift operations uses `rt` field (not `rs`) as first argument, ex: `sllv rd, rt, rs`, `sll rd, rt, sa`. Remember `sa` is the **shift amount**.

Notes:

1. Shift right arithmethic interprets the input to shift as a signed number, the instruction keeps the sign. That is if the number to shift is negative It will preserve the negative sign after shifting.
2. `sll`, `srl` and `sra` use `rt` field of the instruction as first argument, that means that you have to add a MUX for the first ALU operand. Also for the second argument they use the shift amount, you have to increase the size of the ALU second operand MUX. ex: `sll rd, rt, sa`.
3. `sllv, srlv`, `srav` use `rt` as first argument and `rs` second argument. This can be easily solved by reversing the operands the ALU.

Procedure management instructions

We'll implement the instructions `jal` and `jr`. Remember `jal` is a jump instruction but it saves the address of the next instruction in the register `ra` . The `jr` is also a jump, whose target address comes from a register (the `rs` field from the instruction).

Control Unit

You'll have to remove the beq and bne control signals and use the branch signal instead. The following output signals will be added to the control unit:

Signal	Description
branch	Indicates the type of branch in execution. 0 = No branch, 1 = beq, 2 =bne, 3 = bgez, 4 = bgtz, 5 = blez, 6=bltz
jr	1 if the instruction is a `jr`, 0 otherwise
jal	1 if the instruction is a `jal`, 0 otherwise

Take into account that instructions `bgez` and `bltz` has the same opcode. To differentiate them you have to look at the field `rt` of the instruction (Add this as an input to the control unit). In the case `bgez` this field will be 1 and for `bltz` it will be 0.

Testing

For testing you'll receive a series of **test cases** for every instruction added to the processor.