# CSCE 22104

## Lab Report

Brent Marcus Orlina

ID: 011019116

Lab Section 001

Lab 10

## Introduction

This lab's goal was to support a new instruction set-less-than, or `slt`, with the opcode `0111`. The instruction is an R-type which sets the RD register to a 1 if the content of register RS is less than the content of register RT. This is done by subtracting the contents of the two registers, `R[RS]` − `R[RT]`, and looking at the sign bit. If the content of RT is greater than the content of RS, then the difference must be negative and thus the sign bit must be 1. Otherwise, the sign bit is 0. This can be written back to the register file by concatenating fifteen zeroes to the left of only the sign bit of the ALU output.

## Approach

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity Control is
    port(
        op : in std_logic_vector(3 downto 0);

        ctrl_alu_op  : out std_logic_vector(1 downto 0);
        ctrl_alu_src : out std_logic;

        ctrl_reg_src   : out std_logic_vector(1 downto 0); -- changed!
        ctrl_reg_dst   : out std_logic;
        ctrl_reg_write : out std_logic;

        ctrl_mem_read  : out std_logic;
        ctrl_mem_write : out std_logic
     );
end Control;
```

Listing 1: The control block component's ports.

Listing 1 show the ports of the control block component. The output port `ctrl_reg_src` is extended to have another bit. This is because the ALU will simply perform a subtraction operation, with which the sign bit gets written back to the register. Thus, the sign bit can be zero extended to sixteen bits, which can be directly be connected to the write back multiplexer, as shown in listing 3, which is controlled by the `ctrl_reg_src`.

```vhdl
architecture Datapath of Control is
begin
    ctrl_alu_op  <= "01" when op = "0111" else op(1 downto 0); -- changed!
    ctrl_alu_src <= '1'  when op = "1000" else                 -- changed!
                    '0'  when op = "0111" else op(2);

    ctrl_reg_src   <= "00" when op = "1000" else               -- changed!
                      "10" when op = "0111" else "01";
    ctrl_reg_dst   <= '1'  when op = "1100" else '0';
    ctrl_reg_write <= '0'  when op = "1100" else '1';

    ctrl_mem_read  <= '1' when op = "1000" else '0';
    ctrl_mem_write <= '1' when op = "1100" else '0';
end Datapath;
```

Listing 2: The control block component's implementation.

Listing 2 shows the implementation of the control block component to support the new instruction. For the output port `ctrl_alu_op`, a special case has been added for the new instruction `slt`, which has the opcode of `0111`. The original implementation simply looked at the last two bits of the opcode, `11` in this case. The ALU opcode `11` is for the OR operation, which does not match what the `slt` instruction need, a subtraction operation, and thus a special case for this instruction needs to be made.

The output port `ctrl_alu_src` also has a special case for the new instruction. The new instruction is an R-type, however its second bit, index-zero, in the opcode is a `1`, which signals that the ALU should use the last four bits of the instruction to be an immediate value instead of a register address. Thus, a special case is made for the new instruction so that it can be interpreted as an R-type.

Finally, the `ctrl_reg_src` has been changed. Notably, it is now two bits long since the multiplexer for the writeback must support three connections: the memory read output, the ALU output, and the output specifically for the new `slt` instruction. The order of the connections to the write back multiplexer is somewhat arbitrary and simply follows the given lab circuit as shown in figure [INSERT FIGURE REF HERE].

```vhdl
 1  architecture Behavioral of CPU is
 2      -- Components + Signals
 3      ...
 4  begin
 5      -- Instruction Fetch + Instruction Decode + Execute + Memory
 6      ...
 7
 8      -- WriteBack
 9      WriteBack <= MemoryOutput when RegisterSource = "00" else
10                   ALUOutput    when RegisterSource = "01" else
11                   "000" & x"000" & ALUOutput(15);
12
13  end Behavioral;
```

Listing 3: The writeback phase of the CPU implementation

Listing 3 shows the writeback phase of the CPU implentation, since that section is the only part that has changed. The first two cases `RegisterSource` equaling to `00` and `01` are the memory and ALU outputs. The only other case possible that the control block component can output for `RegisterSource` is `10`, thus the output for the `slt` instruction can be added without a condition as seen on line 11. The concatention of `"000" & x"000"` is indeed fifteen zero bits since each zero of `x"000"` is actually four zeroes because it is in hex. Thus, $3 + 3 \cdot 4 = 15$.

## Experimentation

The components were tested by writing a testbench for the CPU component. The written testbench tested for given instructions for the CPU and were manually verified to be correct.

## Results & Discussion

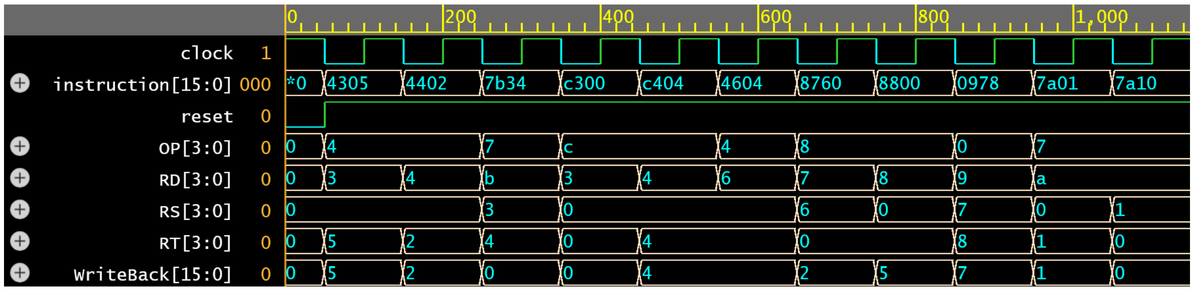

Figure 1: The waveform for the CPU component.

The CPU component works as expected. Figure 1 shows the waveform of the CPU component, correctly outputting the results of each operation. The waveform is in hex radix to easily show the

| Instruction | op | rd | rs | rt | value (of rd) |
|---|---|---|---|---|---|
| `ADDI R3,  R0,  5` | 0x4 | 0x3 | 0x0 | 0x5 | 5 |
| `ADDI R4,  R0,  2` | 0x4 | 0x4 | 0x0 | 0x2 | 2 |
| `SLT  R11, R3, R4` | 0x7 | 0xB | 0x3 | 0x4 | 0 |
| `SW   R3,   0(R0)` | 0xC | 0x3 | 0x0 | 0x0 | 0 |
| `SW   R4,   4(R0)` | 0xC | 0x4 | 0x0 | 0x4 | 4 |
| `ADDI R6,  R0,  4` | 0x4 | 0x6 | 0x0 | 0x4 | 4 |
| `LW   R7,   0(R6)` | 0x8 | 0x7 | 0x6 | 0x0 | 2 |
| `LW   R8,   0(R0)` | 0x8 | 0x8 | 0x0 | 0x0 | 5 |
| `ADD  R9,  R7, R8` | 0x0 | 0x9 | 0x7 | 0x8 | 7 |
| `SLT  R10, R0, R1` | 0x7 | 0xA | 0x0 | 0x1 | 1 |
| `SLT  R10, R1, R0` | 0x7 | 0xA | 0x1 | 0x0 | 0 |

Table 1: Results of the CPU component testbench in table form.

instruction in the current clock cycle and the fact that the ALU does not have any negative results. Table 1 shows the waveform results in table form.

## Conclusions

The new `slt` instruction was implemented into the CPU correctly, shown through the testbench for the CPU. The knowledge from this lab was learning how to support the new `slt` instruction correctly. This was done by realizing that subtraction can be used for comparison by looking at the difference's sign bit. Then, the control block component was adjusted to correctly supply the ALU opcode, ensuring that the new instruction is seen as an R-type, and allowing the write back multiplexer to support another connection.