HW3 (Programming Part) Report

> In case VSCode does not render this markdown file correctly, please refer to `report.pdf` for correctly rendered content.

Design

`ALU` Module

The value of `data_o` is defined in behavioral level, that is, the output signal is event-driven. Whenever any of the signals `ALUCtrl_i`, `data1_i`, or `data2_i` changes, `data1_i (op) data2_i` is calculated based on the value of `ALUCtrl_i` and passed to register `data_o`.
It is worth noting that wires and registers are unsigned by default; to apply an arithmetic right shift (`>>>` in Verilog) to `data1_i` in the case `ALUCtrl_i == 110`, `data1_i` (unsigned) has to be casted to its signed value first. That's why in line 24 of `ALU.v` I use `$signed` function to cast `data1_i`.
In addition, a default case is added in lines 27~28 of `ALU.v` to ensure `data_o` is assigned some value in case the value of `ALUCtrl_i` is not covered by cases above. However, all the 8 possible values of `ALUCtrl_i` are listed in the `case` statement so the default case is redundant.

The value of `zero_o` is defined in register transfer level (RTL). Whenever register `data_o` changes its value, that value is compared with `0`, and the resultant bit (indicating whether `data_o` is zero or not) is passed to wire `zero_o`.

`Registers` Module

The functionality of `Registers` module can be split into reading from registers and writing to a register.

Read from registers

First, I define a macro `read_reg(addr)` to check whether the register to read from is register 0. If `addr` is `0`, meaning a read from register 0, then it returns `32'b0`, which is hard-coded in the program as the register should be hardwired to zero. Otherwise, we are reading from a register that is not register 0, so we can simply obtain the value by `registers[addr]`.
Then, I define the values of wire `RS1data_o` and `RS2data_o` in RTL by connecting each wire

to the corresponding wire that stores the address of the register to read (namely `RS1addr_i` and `RS2addr_i`) via `read_reg` macro.

Write to register

This functionality is defined in behavioral level. On the rising edge of the clock signal (`clk_i`), if `RegWrite_i == 1` and the register to write to is not register 0, then store the value of `RDdata_i` into register `registers[RDaddr_i]`. Note that we check the register address here before performing the write to ensure that register 0 is not written.

Testing

`ALU` Module

I used the module `ALU_tb` in `ALU_tb.v` as the testbench for `ALU` module. `ALU_tb` will first instantiate the `ALU` module and then repeatedly test `ALU` using a loop.
In each iteration of the loop, the values for the ALU inputs `data1_i` and `data2_i` are randomly generated and the correct result (the expected value of `data_o`) is calculated using Verilog code; then, after waiting for some time units for the signals in `ALU` to propagate, the values of `data_o` and `zero_o` are compared to their expected values.
If there are no errors found in the end, `all correct!` is printed.

Testing Commands

1. Compile the testbench:

```
iverilog -o ALU_tb.vvp ALU_tb.v ALU.v
```

2. Execute the testbench:

```
vvp ALU_tb.vvp
```

3. View the resultant waveforms:

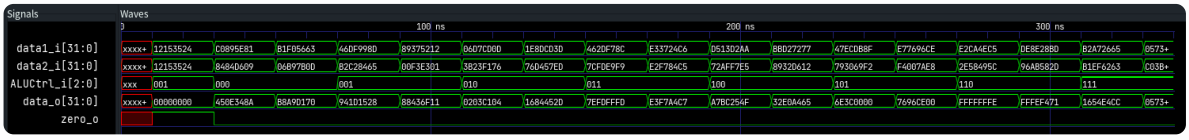
```
gtkwave ALU_tb.vcd --rcvar 'do_initial_zoom_fit yes'
```

Test Results

- Testbench output:

```
>>> ~/ntu/computer-architecture/hw/hw3-programming vvp ALU_tb.vvp
VCD info: dumpfile ALU_tb.vcd opened for output.
@0: data1 = xxxxxxxx, data2 = xxxxxxxx, ALUCtrl = xxx, data = xxxxxxxx, zero = x
@10: data1 = 12153524, data2 = 12153524, ALUCtrl = 001, data = 00000000, zero = 1
@30: data1 = c0895e81, data2 = 8484d609, ALUCtrl = 000, data = 450e348a, zero = 0
@50: data1 = b1f05663, data2 = 06b97b0d, ALUCtrl = 000, data = b8a9d170, zero = 0
@70: data1 = 46df998d, data2 = b2c28465, ALUCtrl = 001, data = 941d1528, zero = 0
@90: data1 = 89375212, data2 = 00f3e301, ALUCtrl = 001, data = 88436f11, zero = 0
@110: data1 = 06d7cd0d, data2 = 3b23f176, ALUCtrl = 010, data = 0203c104, zero = 0
@130: data1 = 1e8dcd3d, data2 = 76d457ed, ALUCtrl = 010, data = 1684452d, zero = 0
@150: data1 = 462df78c, data2 = 7cfde9f9, ALUCtrl = 011, data = 7efdfffd, zero = 0
@170: data1 = e33724c6, data2 = e2f784c5, ALUCtrl = 011, data = e3f7a4c7, zero = 0
@190: data1 = d513d2aa, data2 = 72aff7e5, ALUCtrl = 100, data = a7bc254f, zero = 0
@210: data1 = bbd27277, data2 = 8932d612, ALUCtrl = 100, data = 32e0a465, zero = 0
@230: data1 = 47ecdb8f, data2 = 793069f2, ALUCtrl = 101, data = 6e3c0000, zero = 0
@250: data1 = e77696ce, data2 = f4007ae8, ALUCtrl = 101, data = 7696ce00, zero = 0
@270: data1 = e2ca4ec5, data2 = 2e58495c, ALUCtrl = 110, data = fffffffe, zero = 0
@290: data1 = de8e28bd, data2 = 96ab582d, ALUCtrl = 110, data = fffef471, zero = 0
@310: data1 = b2a72665, data2 = b1ef6263, ALUCtrl = 111, data = 1654e4cc, zero = 0
@330: data1 = 0573870a, data2 = c03b2280, ALUCtrl = 111, data = 0573870a, zero = 0
all correct!
ALU_tb.v:85: $finish called at 34000 (10ps)
```

- Waveforms:



`Registers` Module

I used the module `Registers_tb` in `Registers_tb.v` as the testbench for `Registers` module. `Registers_tb` will first instantiate the `Registers` module and then repeatedly test `Registers` using a loop.
In each iteration of the loop, 2 registers, say, registers i and j, are chosen among all 32 registers. Then on the falling edge of the clock signal, we write a random 32-bit value to register i. On the next falling edge, we read from register i and check whether the value read matches the value previously written to register i, and then we write another random value to register j. On the next falling edge, we read from register j and check whether the value read matches the value previously written to register j. In addition, we also set `RegWrite_i` to `0` and try to write a value different from the one currently stored in register j, and then check if the new value has been successfully written by reading register j at the next falling edge.
If there are no errors found in the end, `all correct!` is printed.

Testing Commands

1. Compile the testbench:

```
iverilog -o Registers_tb.vvp Registers_tb.v Registers.v
```

2. Execute the testbench:

```
vvp Registers_tb.vvp
```

3. View the resultant waveforms:

```
gtkwave Registers_tb.vcd --rcvar 'do_initial_zoom_fit yes'
```

Test Results

- Testbench output (excerpt):

```
>>> ~/ntu/computer-architecture/hw/hw3-programming vvp Registers_tb.vvp
VCD info: dumpfile Registers_tb.vcd opened for output.
@0: RS1addr =  x, RS2addr =  x, RDaddr =  x, RDdata = xxxxxxxx, RegWrite = x, RS1data = xxxxxxxx, RS2data = xxxxxxxx
@10: RS1addr =  0, RS2addr =  1, RDaddr =  0, RDdata = 12153524, RegWrite = 1, RS1data = 00000000, RS2data = xxxxxxxx
@20: RS1addr =  0, RS2addr =  1, RDaddr =  1, RDdata = c0895e81, RegWrite = 1, RS1data = 00000000, RS2data = xxxxxxxx
@25: RS1addr =  0, RS2addr =  1, RDaddr =  1, RDdata = c0895e81, RegWrite = 1, RS1data = 00000000, RS2data = c0895e81
@30: RS1addr =  0, RS2addr =  1, RDaddr =  1, RDdata = c0895e82, RegWrite = 0, RS1data = 00000000, RS2data = c0895e81
@50: RS1addr =  0, RS2addr =  2, RDaddr =  0, RDdata = 8484d609, RegWrite = 1, RS1data = 00000000, RS2data = xxxxxxxx
@60: RS1addr =  0, RS2addr =  2, RDaddr =  2, RDdata = b1f05663, RegWrite = 1, RS1data = 00000000, RS2data = xxxxxxxx
@65: RS1addr =  0, RS2addr =  2, RDaddr =  2, RDdata = b1f05663, RegWrite = 1, RS1data = 00000000, RS2data = b1f05663
@70: RS1addr =  0, RS2addr =  2, RDaddr =  2, RDdata = b1f05664, RegWrite = 0, RS1data = 00000000, RS2data = b1f05663
@90: RS1addr =  0, RS2addr =  3, RDaddr =  0, RDdata = 06b97b0d, RegWrite = 1, RS1data = 00000000, RS2data = xxxxxxxx
@100: RS1addr =  0, RS2addr =  3, RDaddr =  3, RDdata = 46df998d, RegWrite = 1, RS1data = 00000000, RS2data = xxxxxxxx
@105: RS1addr =  0, RS2addr =  3, RDaddr =  3, RDdata = 46df998d, RegWrite = 1, RS1data = 00000000, RS2data = 46df998d
@110: RS1addr =  0, RS2addr =  3, RDaddr =  3, RDdata = 46df998e, RegWrite = 0, RS1data = 00000000, RS2data = 46df998d
@130: RS1addr =  0, RS2addr =  4, RDaddr =  0, RDdata = b2c28465, RegWrite = 1, RS1data = 00000000, RS2data = xxxxxxxx
@140: RS1addr =  0, RS2addr =  4, RDaddr =  4, RDdata = 89375212, RegWrite = 1, RS1data = 00000000, RS2data = xxxxxxxx
@145: RS1addr =  0, RS2addr =  4, RDaddr =  4, RDdata = 89375212, RegWrite = 1, RS1data = 00000000, RS2data = 89375212
@150: RS1addr =  0, RS2addr =  4, RDaddr =  4, RDdata = 89375213, RegWrite = 0, RS1data = 00000000, RS2data = 89375212
@170: RS1addr =  0, RS2addr =  5, RDaddr =  0, RDdata = 00f3e301, RegWrite = 1, RS1data = 00000000, RS2data = xxxxxxxx
@180: RS1addr =  0, RS2addr =  5, RDaddr =  5, RDdata = 06d7cd0d, RegWrite = 1, RS1data = 00000000, RS2data = xxxxxxxx
@185: RS1addr =  0, RS2addr =  5, RDaddr =  5, RDdata = 06d7cd0d, RegWrite = 1, RS1data = 00000000, RS2data = 06d7cd0d
@190: RS1addr =  0, RS2addr =  5, RDaddr =  5, RDdata = 06d7cd0e, RegWrite = 0, RS1data = 00000000, RS2data = 06d7cd0d
@210: RS1addr =  0, RS2addr =  6, RDaddr =  0, RDdata = 3b23f176, RegWrite = 1, RS1data = 00000000, RS2data = xxxxxxxx
@220: RS1addr =  0, RS2addr =  6, RDaddr =  6, RDdata = 1e8dcd3d, RegWrite = 1, RS1data = 00000000, RS2data = xxxxxxxx
@225: RS1addr =  0, RS2addr =  6, RDaddr =  6, RDdata = 1e8dcd3d, RegWrite = 1, RS1data = 00000000, RS2data = 1e8dcd3d
@230: RS1addr =  0, RS2addr =  6, RDaddr =  6, RDdata = 1e8dcd3e, RegWrite = 0, RS1data = 00000000, RS2data = 1e8dcd3d
@19710: RS1addr = 28, RS2addr = 31, RDaddr = 31, RDdata = e4824cca, RegWrite = 0, RS1data = b7f4306f, RS2data = e4824cc9
@19730: RS1addr = 29, RS2addr = 30, RDaddr = 29, RDdata = e4d820c9, RegWrite = 1, RS1data = 4ad39595, RS2data = be43ea7c
@19735: RS1addr = 29, RS2addr = 30, RDaddr = 29, RDdata = e4d820c9, RegWrite = 1, RS1data = e4d820c9, RS2data = be43ea7c
@19740: RS1addr = 29, RS2addr = 30, RDaddr = 30, RDdata = 5a3761b4, RegWrite = 1, RS1data = e4d820c9, RS2data = be43ea7c
@19745: RS1addr = 29, RS2addr = 30, RDaddr = 30, RDdata = 5a3761b4, RegWrite = 1, RS1data = e4d820c9, RS2data = 5a3761b4
@19750: RS1addr = 29, RS2addr = 30, RDaddr = 30, RDdata = 5a3761b5, RegWrite = 0, RS1data = e4d820c9, RS2data = 5a3761b4
@19770: RS1addr = 29, RS2addr = 31, RDaddr = 29, RDdata = 6d48a5da, RegWrite = 1, RS1data = e4d820c9, RS2data = e4824cc9
@19775: RS1addr = 29, RS2addr = 31, RDaddr = 29, RDdata = 6d48a5da, RegWrite = 1, RS1data = 6d48a5da, RS2data = e4824cc9
@19780: RS1addr = 29, RS2addr = 31, RDaddr = 31, RDdata = d6aea8ad, RegWrite = 1, RS1data = 6d48a5da, RS2data = e4824cc9
@19785: RS1addr = 29, RS2addr = 31, RDaddr = 31, RDdata = d6aea8ad, RegWrite = 1, RS1data = 6d48a5da, RS2data = d6aea8ad
@19790: RS1addr = 29, RS2addr = 31, RDaddr = 31, RDdata = d6aea8ae, RegWrite = 0, RS1data = 6d48a5da, RS2data = d6aea8ad
@19810: RS1addr = 30, RS2addr = 31, RDaddr = 30, RDdata = 6fcff1df, RegWrite = 1, RS1data = 5a3761b4, RS2data = d6aea8ad
@19815: RS1addr = 30, RS2addr = 31, RDaddr = 30, RDdata = 6fcff1df, RegWrite = 1, RS1data = 6fcff1df, RS2data = d6aea8ad
@19820: RS1addr = 30, RS2addr = 31, RDaddr = 31, RDdata = 06b0e30d, RegWrite = 1, RS1data = 6fcff1df, RS2data = d6aea8ad
@19825: RS1addr = 30, RS2addr = 31, RDaddr = 31, RDdata = 06b0e30d, RegWrite = 1, RS1data = 6fcff1df, RS2data = 06b0e30d
@19830: RS1addr = 30, RS2addr = 31, RDaddr = 31, RDdata = 06b0e30e, RegWrite = 0, RS1data = 6fcff1df, RS2data = 06b0e30d
all correct!
Registers_tb.v:97: $finish called at 1984000 (10ps)
```

- Waveforms (excerpt):